Secure Hardware Accelerators for Post Quantum Cryptography

Timo ZIJLSTRA

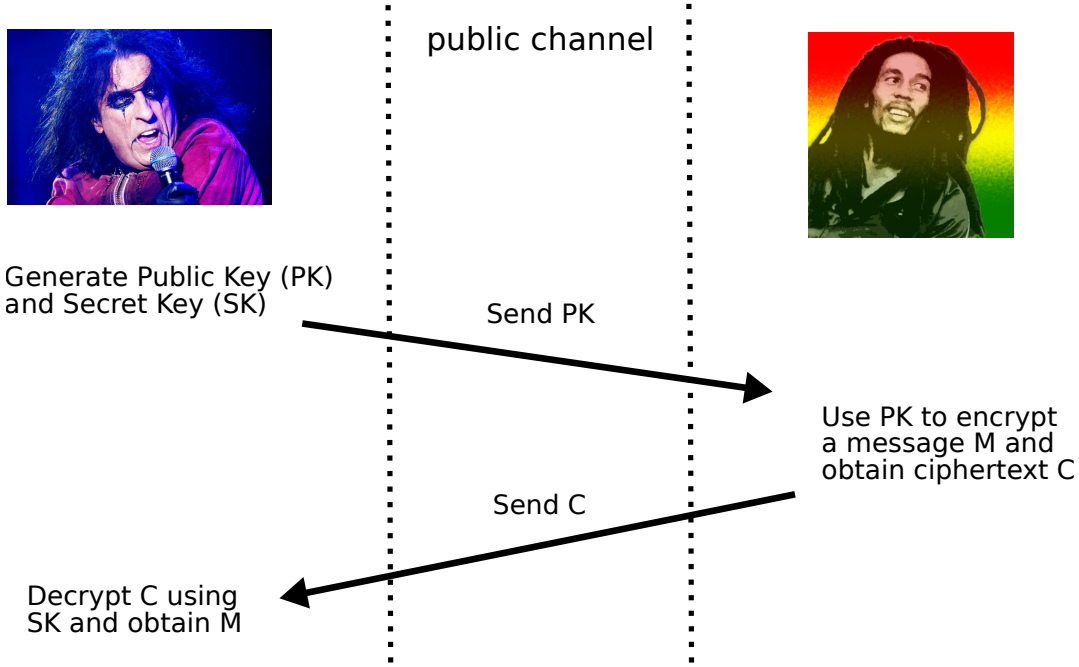
CNRS Lab-STICC, UBS

September 28, 2020

Context: Public Key Encryption (PKE)

Goal: share a secret *message*

1. Alice generates pair of keys and sends the **public key**
2. Bob encrypts his message and sends the **ciphertext**
3. Alices decrypts the ciphertext using the **secret key**



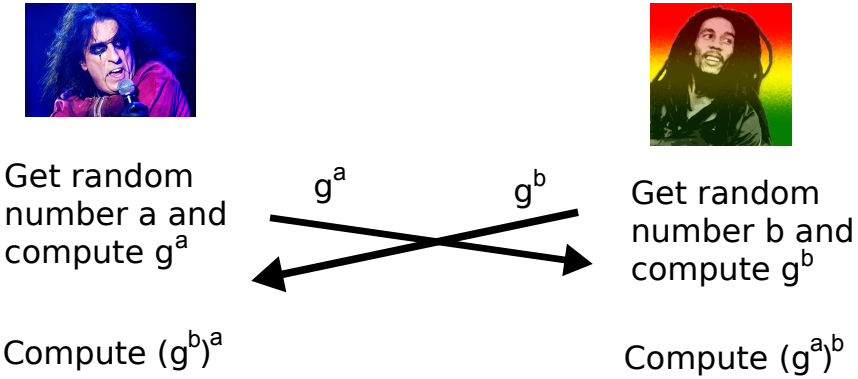
- The data sent over the public channel (C, PK) must not reveil any information about M
- How to generate (PK, SK), C?

Context: Diffie-Hellman

Hypothesis: Discrete Logarithm (DL)

Let $p > 2^{2000}$ be some (public) prime number and g and s integers.[1]
Given only g and g^s mod p, it is practically impossible to compute s.

The security of the key exchange protocol is based on the hardness of DL:



- Shared secret: $(g^a)^b = (g^b)^a$ mod p
- Public: g, g^a, g^b mod p
- DL hypothesis true $\implies$ secure
- Hypothesis false $\implies$ big problem

The DL hypothesis is **false** in the presence of quantum computers [Sho97].

[1] g must also be a generator and s should be large ($> 2^{200}$)

Context: Post Quantum Cryptography (PQC)

- Quantum computers do not yield any generic exponential speed-up
  $\rightarrow$ only some specific mathematical problems are solved

Idea

Find computationally hard problems for which there does not exist any fast quantum algorithm.

Quantum hard computational problems:

- Shortest vector problem
  $\rightarrow$ Lattice cryptography (LWE/NTRU)
- Decoding a random linear code
  $\rightarrow$ Code based cryptography (McEliece)
- Solving a system of multivariate quadratic equations
- Discrete logarithm over elliptic curve isogeny groups

Overview of the presentation

Introduction

NIST Post Quantum Standardization Project

Goal

Select post quantum cryptographic algorithms for standardization.

Timeline:

- 12/2017: Candidates for 1st round announced ($>$ 60 PKE candidates)
- 01/2019: 2nd round candidates announced (around 20 remaining)
- 06/2020: Start of round 3
- 2022-2024: Draft standards available

Round 3 PKE finalists:

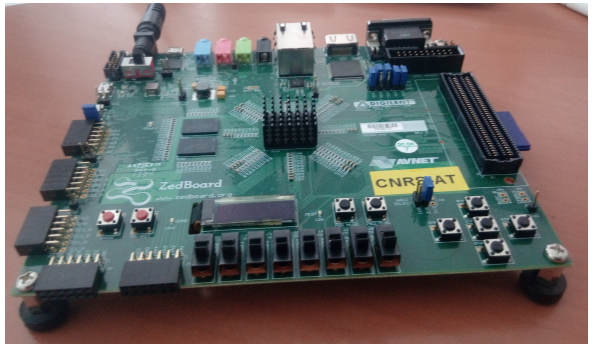
- 3 out of 4 based on lattice problems

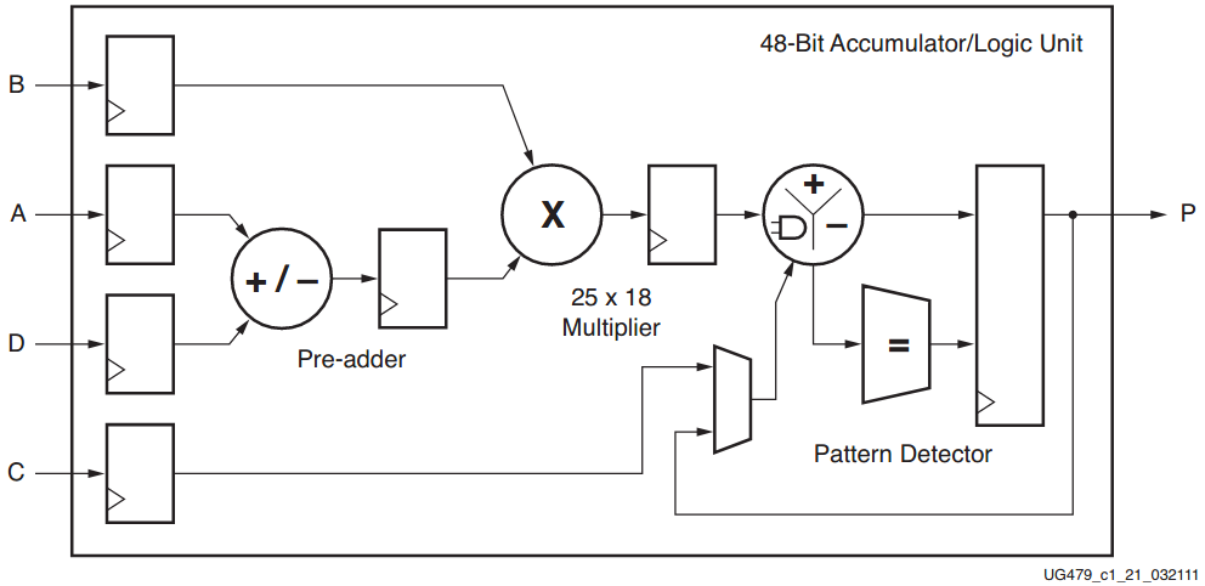
choose the best algorithms (most secure, most efficient, etc)

- Security evaluation by cryptographers around the world
- We focus on secure and optimized hardware implementations

FPGAs

- Programmable hardware devices
- Contain a number of elements:
  - Look up tables (LUT)
  - DSP blocks
  - BRAM





Xilinx DSP48E1 (from UG479)

Important metrics: **computation time**, **resource utilization**

High-Level Synthesis (HLS)

- FPGAs can be programmed using VHDL/Verilog language
  $\rightarrow$ a time consuming process!
- Or using HLS:
  1. Program in C
  2. HLS transforms it to VHDL description
     $\rightarrow$ faster, easier and compiler assisted optimizations

  example:

```
for(i=0; i<N; i++){
#pragma HLS pipeline
#pragma HLS unroll factor=2
    t = A[i]*B[i];
    C[i] = reduce_mod(t);
}
#pragma HLS allocation instances=mul limit=1 reduce_mod
```

- Explore many different *directives* combinations
- Generic implementations for complex algorithms

Side-Channel Attacks (SCAs)

SCA method

Obtain secret data by measuring physical quantities of the device.

1. Power consumption
   $\longrightarrow$ *SPA*/*DPA* and variants
2. Computation time
   $\longrightarrow$ *Timing analyisis*
3. Electromagnetic radiation
4. Also: fault attacks

- Security against SCA of RSA/ECC is well studied
- For new PQC, not so much

Example

function(*secret_bit*):
    if *secret_bit* $==$ 1 **then**
        compute $g()$
    else
        compute $f()$

$\longrightarrow$ need to study countermeasures against SCA for PQC

Lattice based Cryptography

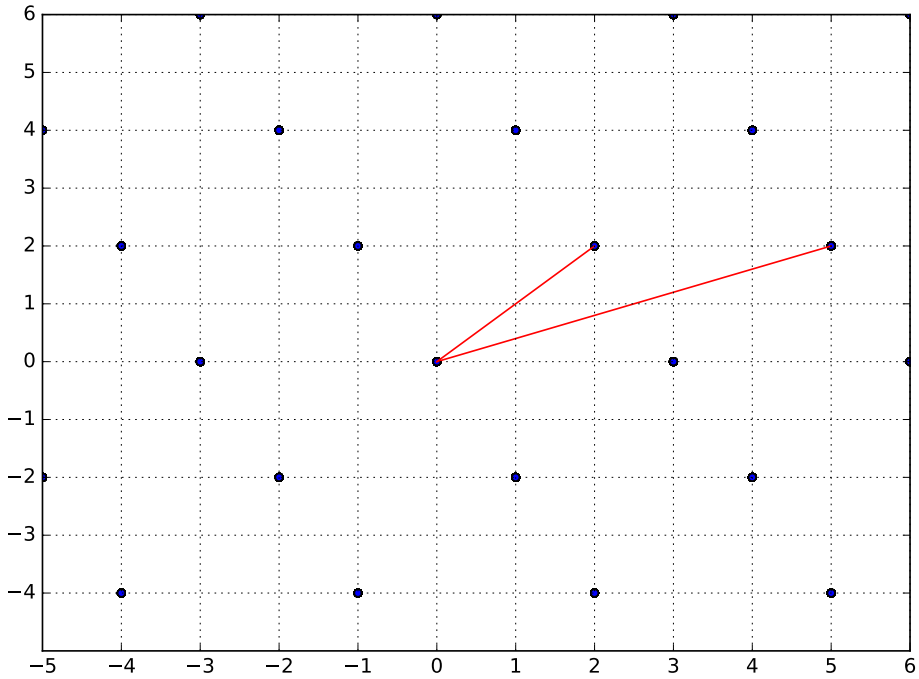
Lattice Problems

Lattice (definition)

A lattice $\mathcal{L}$ is an additive subgroup of $\mathbb{Z}^n$ for some n. It is generated by n basis vectors $\mathbf{b}_1, \ldots, \mathbf{b}_n$:

$$\mathcal{L} = \left\{ \sum_{i=1}^{n} x_i \mathbf{b}_i : x_i \in \mathbb{Z} \right\} \tag{1}$$

- Example in $\mathbb{Z}_{12}^2$
- Generated by 2 basis vectors
- Shortest vector problem (SVP): find smallest vector $\neq 0$



Learning With Errors (LWE)

- LWE problem definition:
  1. Let $\bar{s}$ be some secret vector in some finite field
  2. Let $A = (a_{i,j})$ be some public random matrix
  3. Given A and $\bar{b} \approx A\bar{s}$, find $\bar{s}$.
- Equation ($=$) instead of ($\approx$) $\implies$ Easy, using Gaussian elimination
- With added noise ($\approx$) $\implies$ Very hard
- [Reg05] LWE is at least as hard as SVP using quantum computers $\rightarrow$ useful for PQC

Encryption Scheme ([Reg05][LPR10])

Plaintext: μ

Alice $(\mathit{KeyGen}, \mathit{Decrypt})$:		**Bob** $(\mathit{Encrypt}\ \mu)$:
$\mathbf{A} \xleftarrow{\$}$ random matrix		
$\mathbf{s}, \mathbf{e} \xleftarrow{\$}$ vector w/ small coefficients		
$\mathbf{b} \leftarrow \mathbf{A} \cdot \mathbf{s} + \mathbf{e}$	$\xrightarrow{\mathbf{A},\mathbf{b}}$	$\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3 \xleftarrow{\$}$ small
		$\mathbf{c}_1 \leftarrow \mathbf{A} \cdot \mathbf{e}_1 + \mathbf{e}_2$
$\mu' \leftarrow \mathcal{D}(\mathbf{c}_2 - \mathbf{c}_1^\mathsf{T}\mathbf{s})$	$\xleftarrow{\mathbf{c}_1,\mathbf{c}_2}$	$\mathbf{c}_2 \leftarrow \mathbf{b}^\mathsf{T}\mathbf{e}_1 + \mathbf{e}_3 + \mathcal{E}(\mu)$

Possible SCA vulnerability: computation of $\mathbf{c}_1^\mathsf{T}\mathbf{s}$

Structured and Unstructured Lattices

- Matrix multiplication is expensive (many finite field operations) $\rightarrow$ use structured lattices to reduce complexity [LPR10]
- Secret key: some LWE secret $\mathbf{s} \in \mathcal{R}_q^k := (\mathbb{Z}_q[x]/(x^n+1))^k$

Parameters and corresponding algorithms in NIST competition

scheme	based on	polynomial degree n	vector length k	modulus q
FrodoKEM	LWE	1	640/976/1344	$2^{15}/2^{16}$
NewHope	RLWE	1024	1	12289
Kyber	MLWE	256	$2/3/4 > 1$	7681

- Secret key is
  - a $8 \times k$ matrix over $\mathbb{Z}_q$ in FrodoKEM
  - a polynomial in $\mathbb{Z}_{12289}/(x^{1024}+1)$ in NewHope
  - a vector in $\left(\mathbb{Z}_{7681}[x]/(x^{256}+1)\right)^k$ in Kyber

Contributions of the thesis (yellow = presented in this talk)

1. HLS Implementation of finite field arithmetic for FPGAs
   - Analyse impact of various algorithms on HLS implementation results
   - Compas 2019 (national conference)
2. FPGA implementation of post-quantum cryptosystems
   - Based on LWE, RLWE and MLWE
   - Acceleration using parallelism
   - Timing / area trade-offs often better than state of the art
   - CPA and CCA2 secure solutions
   - Analysis of perfomance of various PRNGs used in the schemes
   - Submitted to **IEEE Transactions on Computers**, under major revision
3. Countermeasures against SCAs
   - Show that cryptoscheme is vulnerable to SCA (simulation)
   - Implement existing countermeasures
     - Masking, Shifting, Blinding
   - Propose new countermeasures
     - Secret key randomization using redundant number representation
     - Shuffling (using 2 different methods)
   - **IndoCrypt** 2019 (international conference)

HLS Implementation of LWE based PKE on FPGA

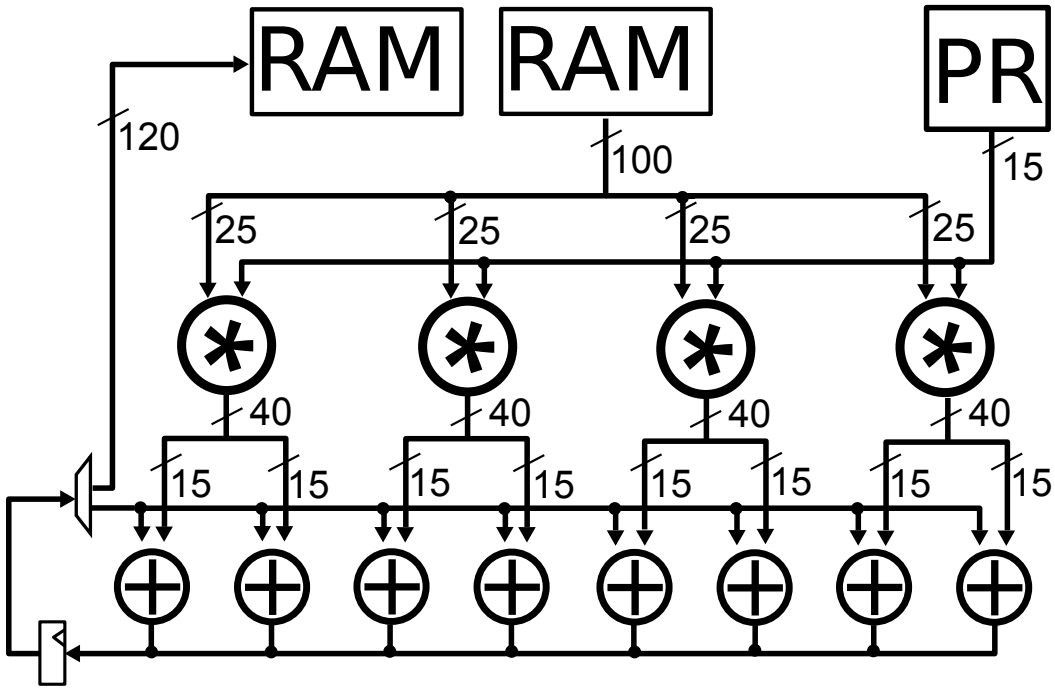
Implementation of LWE based Encryption

- Main part of encryption:
  1. Sample from binomial distribution: 640×8 matrix $\mathbf{E}_1$
  2. Generate public key 640×640 matrix $\mathbf{A}$
  3. Compute $\mathbf{E}_1^\mathsf{T}\mathbf{A}$
- Bottleneck: matrix multiplication ($640^2 \times 8$ 15-bit multiplications)
- Speed up using *error packing* technique from [LFK+19]
- Speed up by parallelization of operations:

$$\underbrace{\begin{pmatrix} e_{00}||e_{10} & e_{01}||e_{11} & \cdots \\ \vdots & \vdots & \\ e_{60}||e_{70} & e_{61}||e_{71} & \end{pmatrix}}_{k \text{ columns}} \times \underbrace{\begin{pmatrix} a_{00} & a_{01} & \cdots \\ a_{10} & a_{11} & \\ \vdots & & \\ \vdots & & \\ a_{(k-1)0} & & \end{pmatrix}}_{k \text{ columns}}$$

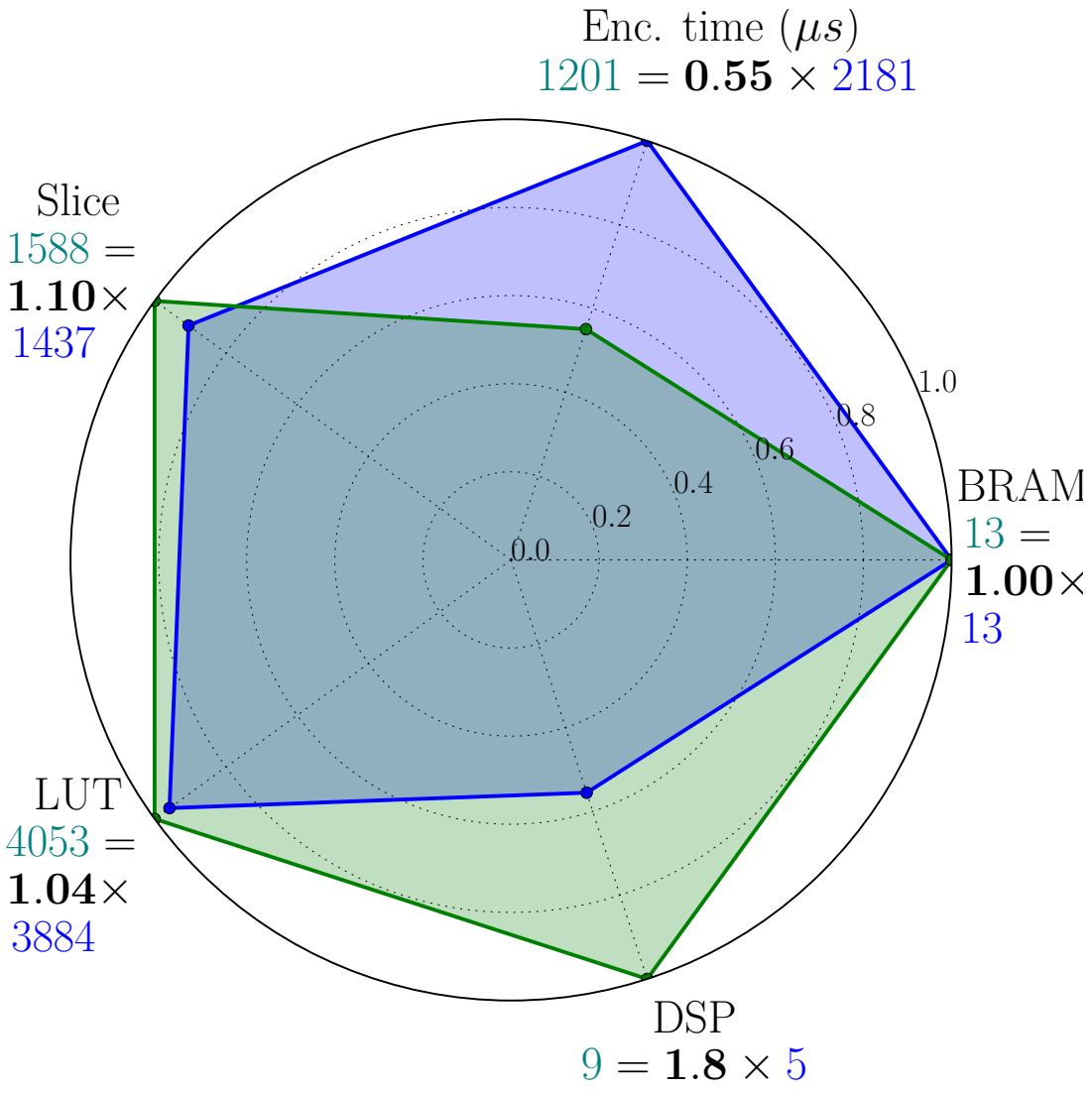
Matrix Multiplication Base Implementation Architecture

- *Base implementation* uses 4 DSP (encryption time: 2181μs)

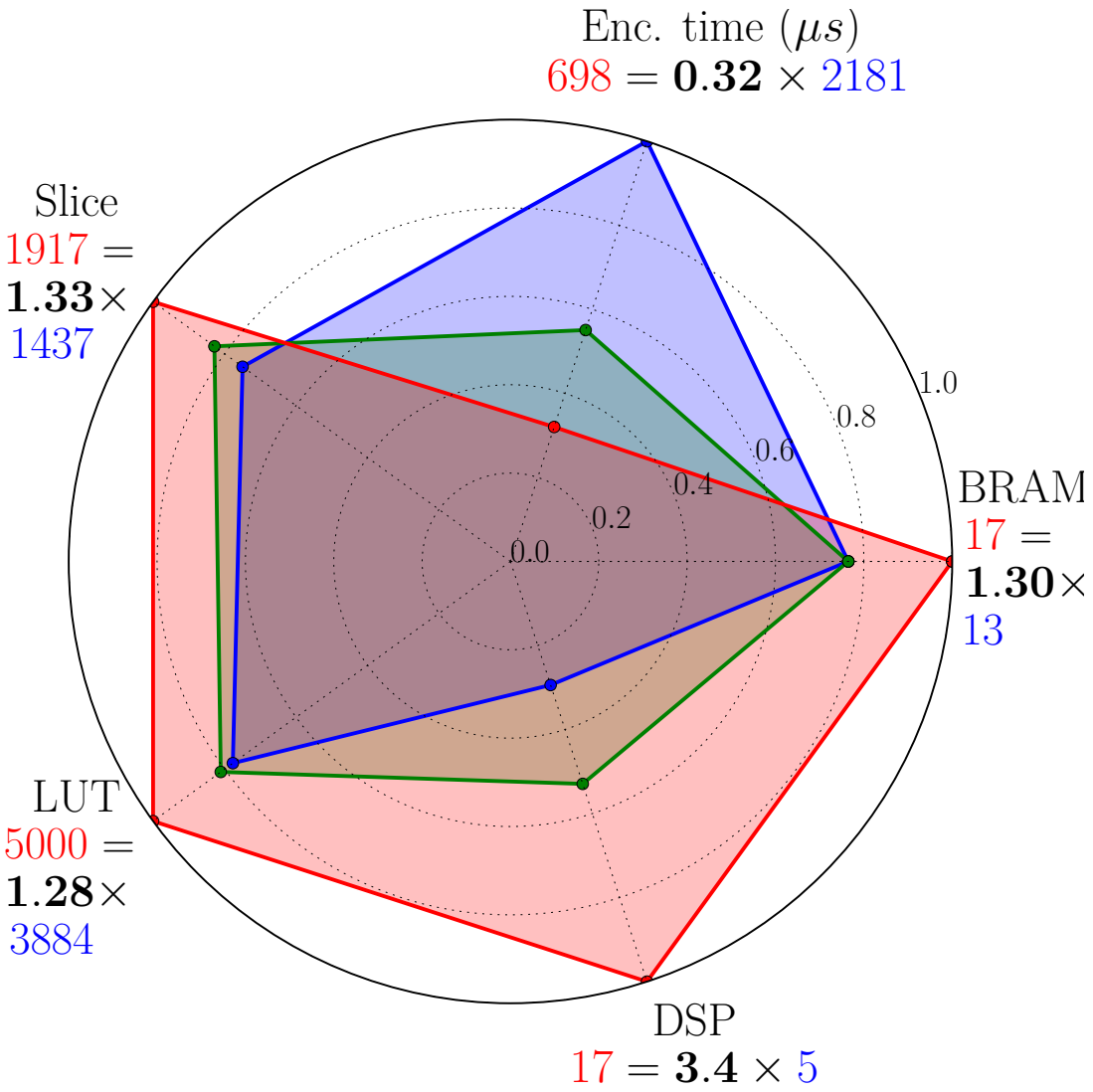


- Further speed up by increasing the level of parallelization
- Parallelization factor f: compute products between f PK coefficients and f columns of $\mathbf{E}_1$ simultaneously
- Write generic codes for f and parameters using HLS

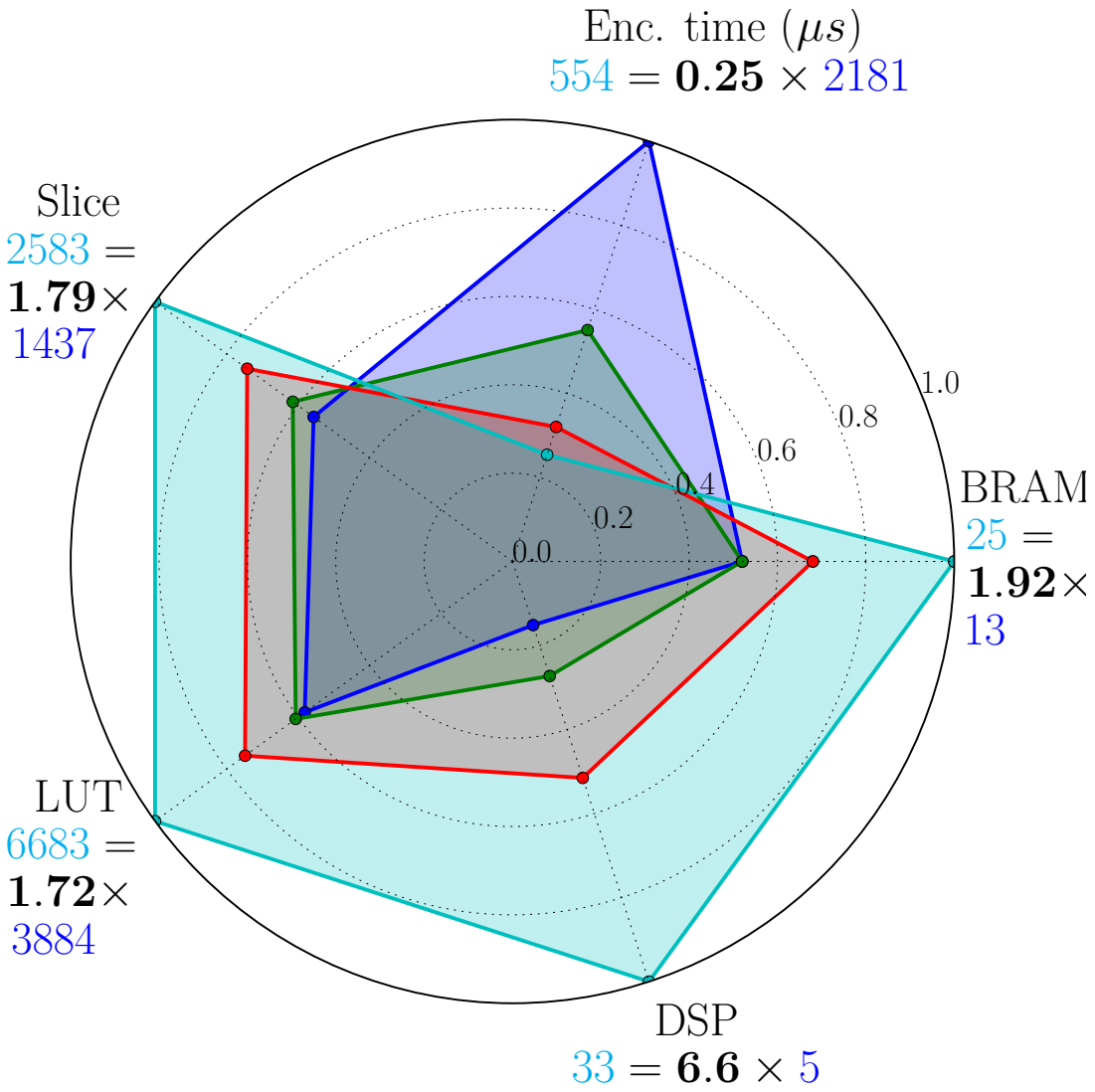
LWE $f = 2$


Enc. time (μs)
1201 = **0.55** × 2181
BRAM
13 = **1.00**× 13
DSP
9 = **1.8** × 5
LUT
4053 = **1.04**× 3884
Slice
1588 = **1.10**× 1437
0.0
0.2
0.4
0.6
0.8
1.0

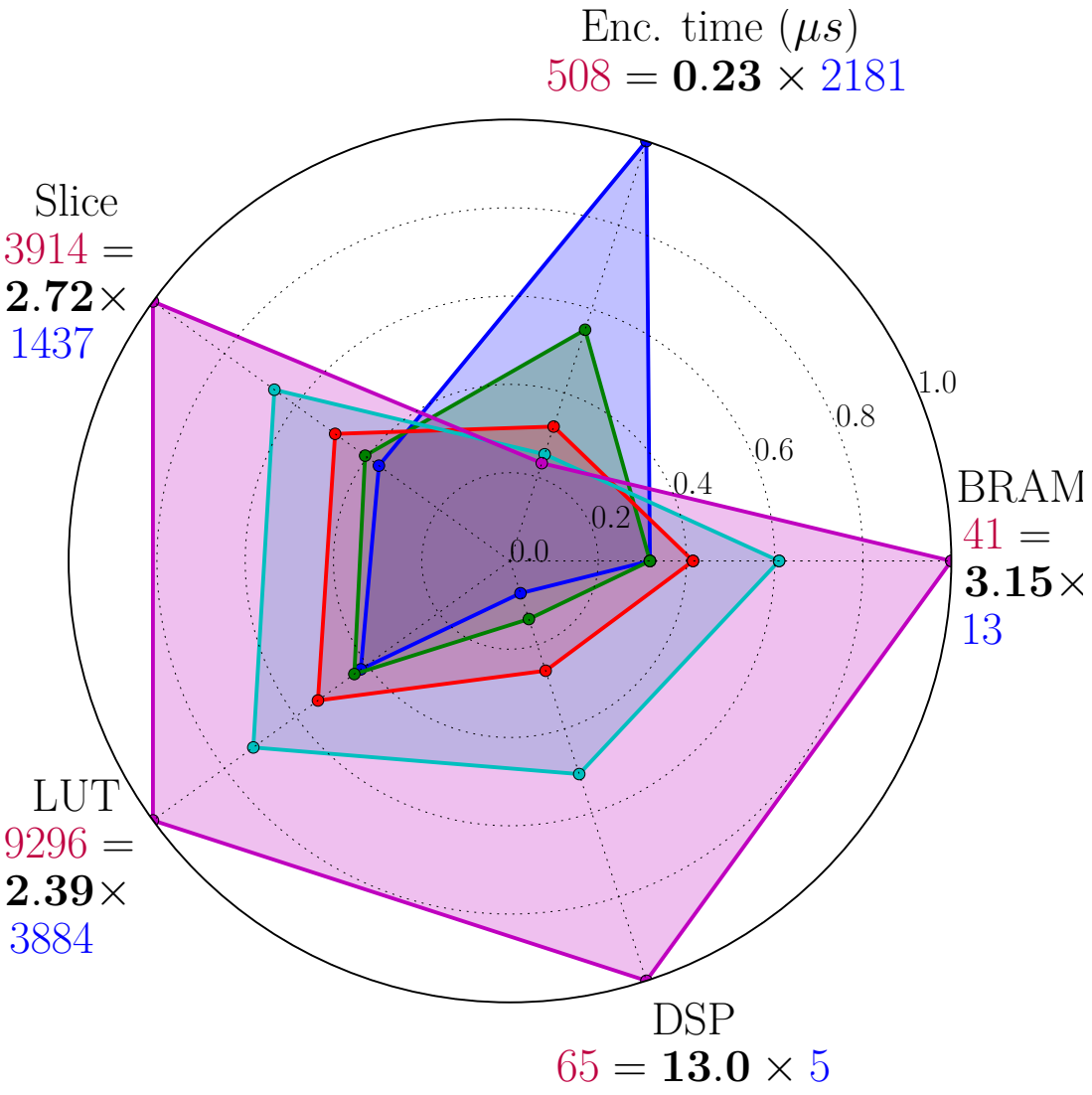
LWE $f = 4$


Enc. time (μs)
698 = **0.32** × 2181
BRAM
17 = **1.30**× 13
DSP
17 = **3.4** × 5
LUT
5000 = **1.28**× 3884
Slice
1917 = **1.33**× 1437
0.0
0.2
0.4
0.6
0.8
1.0

LWE $f = 8$


Enc. time (μs)
554 = **0.25** × 2181
BRAM
25 = **1.92**× 13
DSP
33 = **6.6** × 5
LUT
6683 = **1.72**× 3884
Slice
2583 = **1.79**× 1437
0.0
0.2
0.4
0.6
0.8
1.0

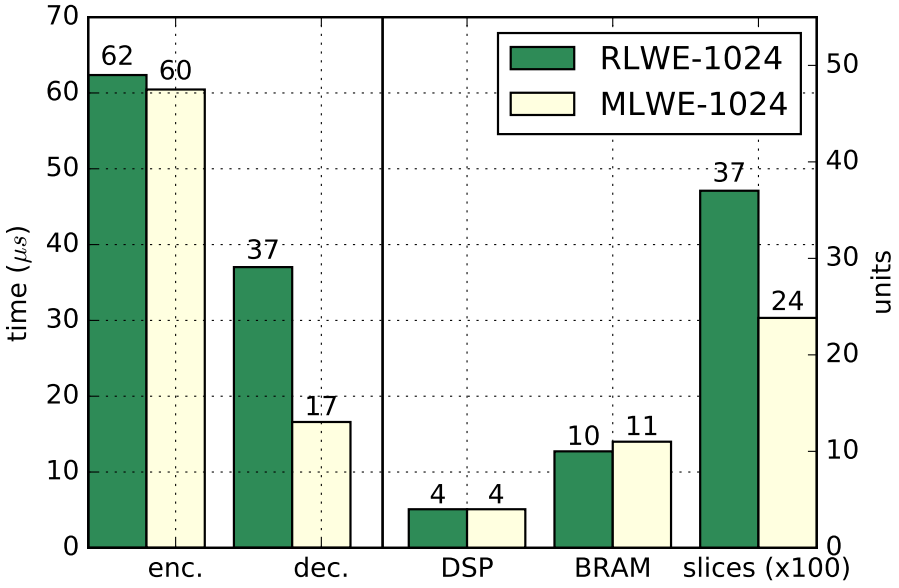
LWE $f = 16$


Enc. time (μs)
508 = **0.23** × 2181
BRAM
41 = **3.15**× 13
DSP
65 = **13.0** × 5
LUT
9296 = **2.39**× 3884
Slice
3914 = **2.72**× 1437
0.0
0.2
0.4
0.6
0.8
1.0

RLWE vs MLWE

RLWE	MLWE
polynomials	small matrices of polynomials
$n = 1024$	$n = 256$
$q = 12289$	$q = 7681$

- Polynomial multiplication uses NTT
  $\rightarrow$ large NTT (RLWE) vs multiple smaller NTTs (MLWE)
- Base implementation results:



Parallelization of RLWE/MLWE Encryption

- Tested many (> 100) different combinations of HLS directives

RLWE:

- compute NTT($\mathbf{e}_1$) and NTT($\mathbf{e}_2$) simultaneously
  $\rightarrow$ also works for MLWE

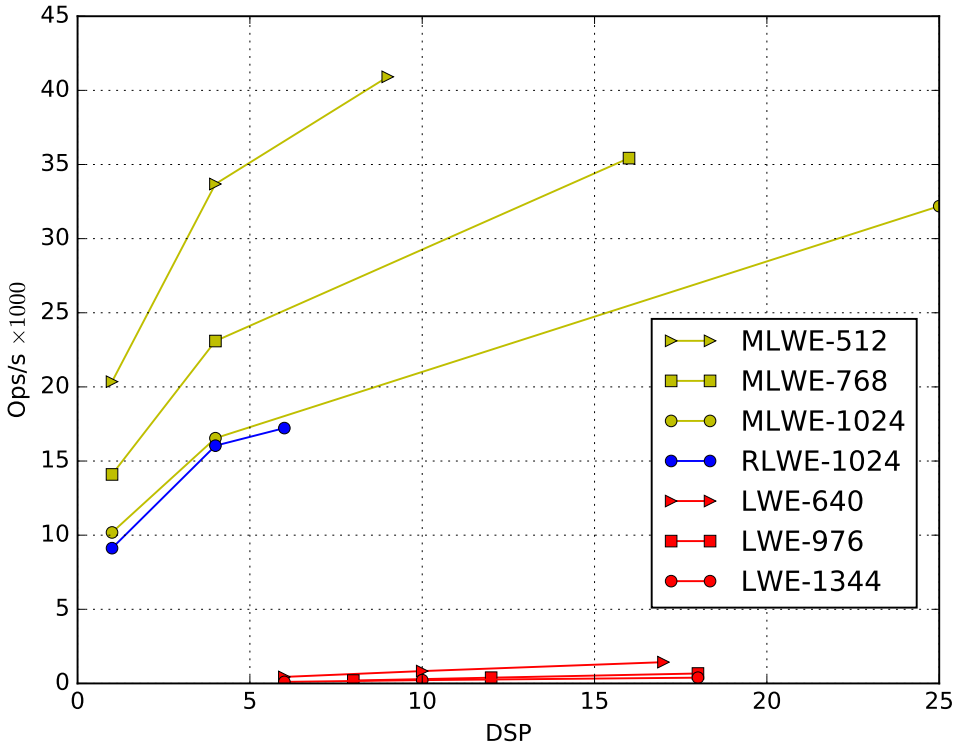
MLWE:

- compute vector multiplication $\mathbf{e}_1 \cdot \mathbf{a}$ in parallel. Instead of

$$\mathbf{e}_1^{(0)}\mathbf{a}^{(0)} + \mathbf{e}_1^{(1)}\mathbf{a}^{(1)} + \cdots + \mathbf{e}_1^{(k-1)}\mathbf{a}^{(k-1)}$$

  compute k products $\mathbf{e}_1^{(0)}\mathbf{a}^{(0)}, \mathbf{e}_1^{(1)}\mathbf{a}^{(1)}, \ldots, \mathbf{e}_1^{(k-1)}\mathbf{a}^{(k-1)}$ simultaneously
- compute the k NTT's of $\mathbf{e}_1^{(0)}, \mathbf{e}_1^{(1)}, \ldots, \mathbf{e}_1^{(k-1)}$ in parallel

Speeding up LWE/RLWE/MLWE Encryption



- Throughput for 3 degrees of parallelism (3 points of each curve)
- Parallelized LWE very slow w.r.t. RLWE/MLWE
- MLWE can be accelerated more than RLWE
  - Matrix/vector structures are easily parallelized in MLWE
  - In RLWE the NTT is the main bottleneck

Comparison with other Works

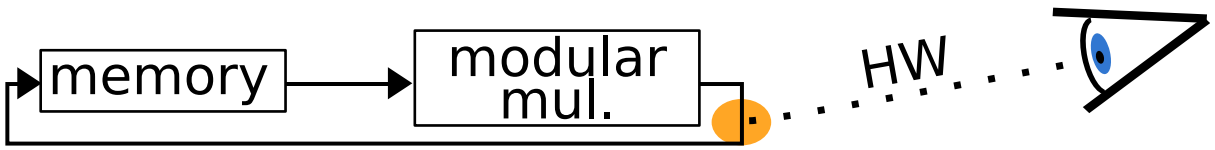
Source	Type	FPGA	μs	DSP, BRAM, Slices, LUT
RLWE-1024				
[KLC+17]	K-E	xc7z020	79	8, 14, n.a. 20826
[OG17]	K-E	xc7a35t	1532	2, 4, n.a., 4498
[ZYC+20]	CPA	xc7z020	62	2, 8, n.a, 6781 (*)
This work	CPA	xc7a200	63	4, 10, 3701, 10112 (*)
LWE-640				
[HMOR19]	CCA	Artix-7	4624	4, 0, 1338, 4620
This work	CCA	xc7a200	2972	5, 37, 12951, 39077 (*)
MLWE-1024				
[AEL+20]	CCA	xc7a35t	6900	4, 34, n.a., 1738
This work	CCA	xc7a200	170	11, 16, 11567, 33707 (*)

SCAs and Countermeasures

Side Channel Analysis Model

- Attack each modular multiplication separately during decryption
- Hypothesis: for each $c \cdot s \bmod q$, the power trace allows to guess:

$$\mathrm{HW}(c \cdot s \bmod q) + \mathcal{N}(0, \sigma)$$



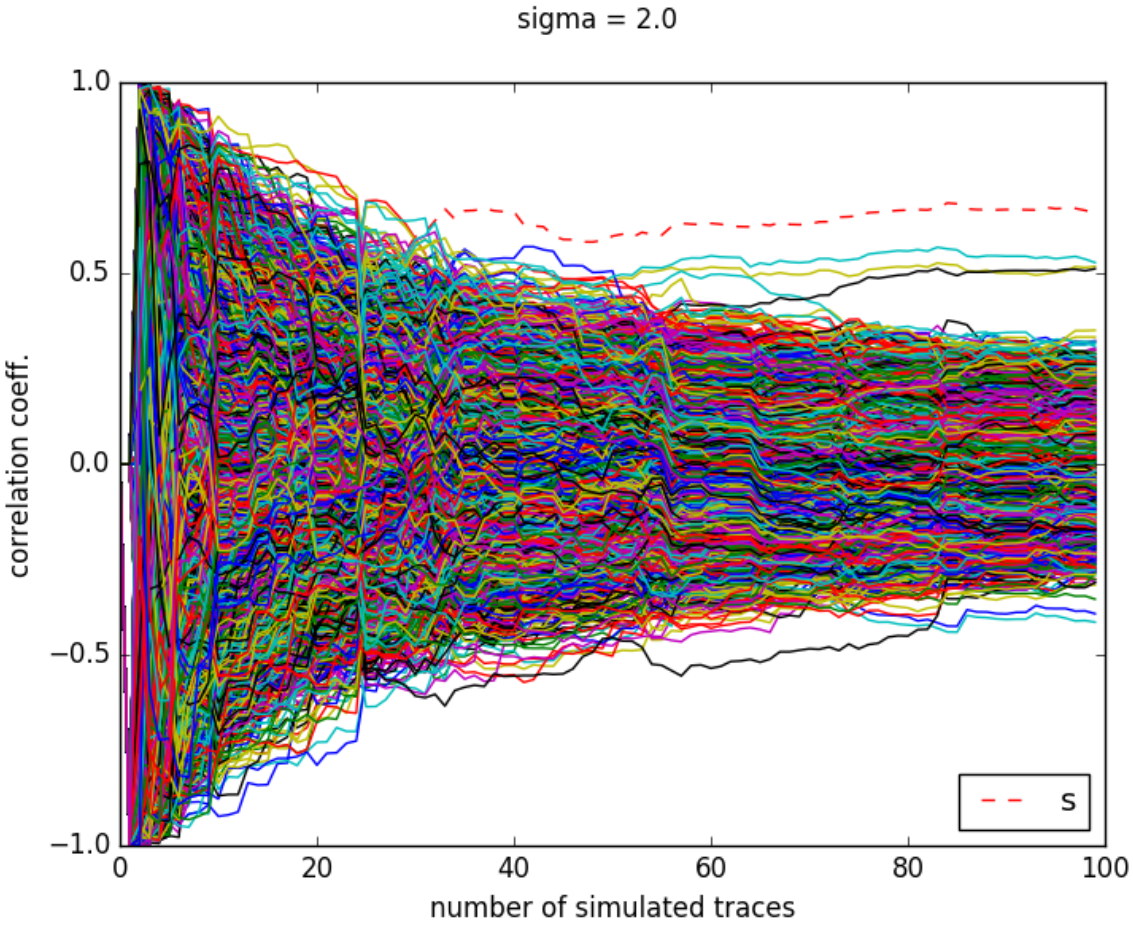
Correlation Power Attack:

1. Generate random ciphertexts
2. Predict power traces
3. Record power traces during decryption
4. Compute correlation between traces and predictions
5. Maximum correlation is obtained for the correct guess

Side Channel Attack Simulation

Simulate Correlation Power Attack in SageMath:

- Machine executing one instruction per cycle
- Attack 1 single coefficient of RLWE-256 decryption
- Correlations for all subkey guesses:



Countermeasures

- Randomize computations
- How to obtain correct results from randomized computations?
  1. Masking [RRVV15]
     - [RRVV15] reports FPGA implementation
     - Masked decryption 3 times slower than unmasked
     - We optimize and re-implement it on FPGA
  2. Blinding and Shifting at algorithmic level [Saa18]
     - We implement it on FPGA
  3. Shuffling (randomize the order of computations)
     - We propose 2 methods
  4. New redundant secret key representation
- Implemented for RLWE-256 but can be applied to MLWE as well

Countermeasure: Masked Decryption [RRVV15]

Use linearity: $\mathbf{a}(\mathbf{b}+\mathbf{c}) = \mathbf{ab}+\mathbf{ac}$

1. Generate a uniform random $\mathbf{s}'$ and let
   $\mathbf{s}'' \leftarrow \mathbf{s} - \mathbf{s}'$
   $\rightarrow$ then $\mathbf{s} = \mathbf{s}' + \mathbf{s}''$
2. Compute (part of) the decryption function for both shares:
   - $\mathbf{d}' \leftarrow \mathbf{c}_2 - \mathbf{c}_1\mathbf{s}'$
   - $\mathbf{d}'' \leftarrow -\mathbf{c}_1\mathbf{s}''$

   Then $\mathcal{D}(\mathbf{d}' + \mathbf{d}'') = \mu$
3. Use masked decoder to extract μ from $\mathbf{d}'$ and $\mathbf{d}''$
   - [RRVV15]: probabilistic decoder
   - We propose a deterministic decoder
   - Implement both solutions on FPGA

Reminder

$\textsc{Decrypt}(\mathbf{c}_1, \mathbf{c}_2) = \mathcal{D}(\mathbf{c}_2 - \mathbf{c}_1\mathbf{s})$

Countermeasure: Blinding [Saa18]

For all integers a, b:

$$a\mathbf{c}_1 \cdot b\mathbf{s} = (ab)(\mathbf{c}_1 \cdot \mathbf{s})$$

1. Pick some random $a, b \in \mathbb{Z}/q\mathbb{Z}$ and compute $(ab)^{-1}$
2. Compute $a\mathbf{c}_1 \cdot b\mathbf{s}$
3. Multiply by $(ab)^{-1}$ and subtract $\mathbf{c}_2$ to obtain correct $\mathbf{d}$
4. Decode
   $\rightarrow$ [Saa18]: use pre-computed roots of unity $\omega^i, \omega^j, \omega^{n-i-j}$

Reminder

$\textsc{Decrypt}(\mathbf{c}_1, \mathbf{c}_2) = \mathcal{D}(\mathbf{c}_2 - \mathbf{c}_1\mathbf{s})$

- Computation of $\mathbf{c}_1 \cdot \mathbf{s}$ randomized at each run
- $\mathbf{d} = \mathbf{c}_2 - \mathbf{c}_1\mathbf{s}$ is not randomized $\implies$ decoding algorithm is not protected
  $\rightarrow$ use blinding in combination with another countermeasure

Countermeasure: Shifting [Saa18]

1. Multiply $\mathbf{s}$ and $\mathbf{c}_1$ by x^i and x^j respectively, for random $i, j < n$
2. Obtain $\mathbf{c}_1 \mathbf{s} x^{i+j}$
3. Multiply by $x^{-(i+j)}$

- In $\mathbb{Z}_q[x]/(x^n + 1)$: multiply by $x^i \iff$ shift i positions to the right $\rightarrow$ easy to compute
- NTT domain: pointwise multiplication by $NTT(x^i) = (1, \omega^i, \omega^{2i}, \dots)$ $\rightarrow$ still easy to compute (since ω^i is pre-computed for all $i < n$)

Shifted decryption:

1. Get random indices $i, j < n$
2. Compute $NTT(x^i) \odot \mathbf{s}$, $NTT(x^j) \odot \mathbf{c}_1$ and $NTT(x^{i+j}) \odot \mathbf{c}_2$
3. Decrypt and shift $i + j$ positions to the left

Countermeasure: Shuffling

- Pointwise multiplications can be computed in any order
- Instead of:

$$\mathbf{a}_0 \cdot \mathbf{b}_0, \mathbf{a}_1 \cdot \mathbf{b}_1, \ldots$$

  compute:

$$\mathbf{a}_{\sigma(0)} \cdot \mathbf{b}_{\sigma(0)}, \mathbf{a}_{\sigma(1)} \cdot \mathbf{b}_{\sigma(1)}, \ldots$$

  for some random permutation σ
- Also works for pointwise multiplications in the NTT
  $\rightarrow$ protects against [PPM17] SPA attack

2 permutation methods:

1. LFSR counter
   - Cheap: initial state defines random permutation
   - Only 255 permutations possible for $n = 256$
2. Permutation network
   - Uses $\frac{n}{2}\log(n)$ random bits
   - $n^{n/2}$ different permutations possible

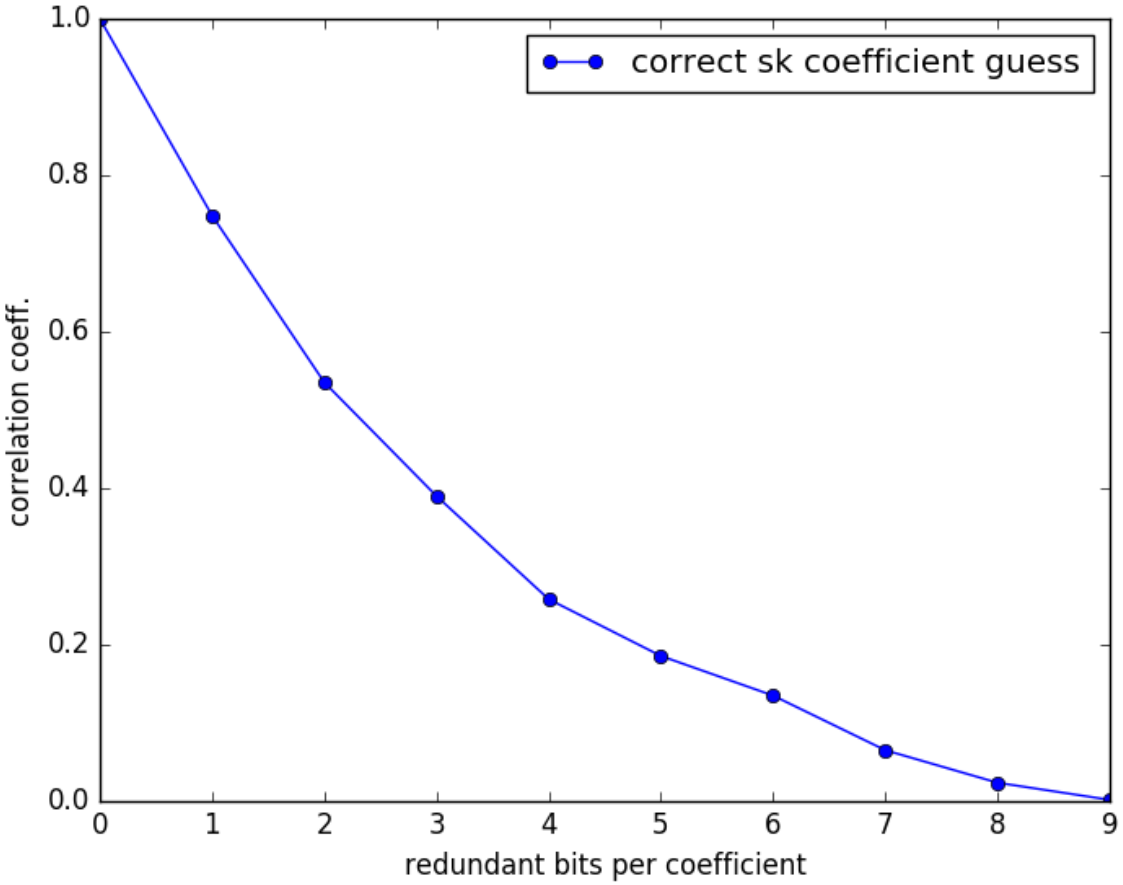
Countermeasure: Redundant Representation

Use a redundant representation to randomize secret key

- In RSA/ECC:
  1. Secret key is scalar or exponent in some group
  2. Randomize by adding multiples of the group order
     $\rightarrow$ new secret key is still a valid decryption key
- We apply this to RLWE crypto:
  1. Secret key: n coefficients in $\mathbb{Z}/q\mathbb{Z}$
  2. For each coefficient: add a small random multiple of q
  3. Perform computations in $\mathbb{Z}/(2^r q)\mathbb{Z}$
     $\rightarrow$ r is the redundancy parameter
- Increase $r \implies$ increase security against SCA
- Using HLS: easy to obtain implementations for various values of r
- Generic implementations for different security levels

Does it work?

- Correlation analysis for different redundancy levels
  - Attacking one single modular multiplication
  - Hamming Weight model assuming noiseless observations
- Simulations in SageMath ($\sigma = 0$):



Implementation on Artix-7 FPGA using HLS, RLWE-256

Protection	Source	Implem.	Time (μs)	Slice, LUT, DSP, BRAM
None	-	[RRVV15]	23.5	-, 1713, 1, -
Masking	[RRVV15]	[RRVV15]	75.2	-, 2014, 1, -
None	-	This work	7.8	483, 1163, 2, 3
Masking	[RRVV15]	This work	10.1	2187, 5500, 5, 6
Our Mask.	This work	This work	10.1	1722, 4269, 5, 6
Blinding	[Saa18]	This work	10.6	941, 2284, 3, 4
Shifting	[Saa18]	This work	14.8	832, 2150, 3, 4
Shift + Blind	[Saa18]	This work	14.7	1063, 2781, 3, 4
Permutation	This work	This work	11.4	3183, 7385, 2, 4
LFSR ctr.	This work	This work	10.3	1069, 2861, 2, 3
Redund. $r = 1$	This work	This work	8.5	629, 1599, 2, 3
$r = 2$	This work	This work	8.2	611, 1664, 2, 3
$r = 3$	This work	This work	8.9	807, 2067, 2, 3
$r = 4$	This work	This work	8.5	872, 2285, 2, 3
$r = 5$	This work	This work	9.0	990, 2677, 2, 6

Conclusion

Conclusion

- Uncertain when (or if) there will be a quantum computer
- PQC in stage of developement
  - Theoretic security
  - Cost-efficiency (speed/size of implementations)
  - Side channel security
- In this thesis:
  - Proposed fast and small FPGA implementations for LWE/RLWE/MLWE based encryption schemes, and modular arithmetic
  - Improved and implemented various countermeasures against SCAs
  - Proposed new countermeasures
- Perspectives:
  - Analyse security of proposed countermeasures
    $\rightarrow$ Side channel analysis
  - Optimize area utilization of HLS implementations
    $\rightarrow$ sharing of resources, BRAM efficiency

Publications and Presentations

- Publications:
  - T. Zijlstra, K. Bigou, and A. Tisserand. "FPGA Implementation and Comparison of Protections against SCAs for RLWE", presented at IndoCrypt 2019 (acceptance rate = 25%).
  - L. Djath, T. Zijlstra, K. Bigou, A. Tisserand. "Comparaison d'algorithmes de réduction modulaire en HLS sur FPGA", COMPAS 2019.
- Submitted work:
  - T. Zijlstra, K. Bigou, and A. Tisserand. "Lattice-based Cryptosystems on FPGA: Parallelization and Comparison", submission to IEEE Transactions on Computers (under major revision).
- Presentations:
  - T. Zijlstra. "Countermeasures against physical attacks on ring-LWE encryption schemes"presented at WRAC'H 2019.
  - T. Zijlstra. "Introduction to Post Quantum Cryptography", presented at the internal seminar of the MOCS team, 2019.
  - Various presentations related to post quantum cryptography at GT Sécu meetings.

Thank you for your attention!