Side-channel analysis countermeasures for lattice-based cryptography

Oscar Reparaz

PQCRYPTO mini-workshop
2016.06.28, Utrecht (NL)
side-channel attacks meets pqcrypto
side-channel attacks meets pqcrypto
side-channel attacks meets pqcrypto
side-channel attacks meets pqcrypto
side-channel attacks meets pqcrypto
side-channel attacks meets pqcrypto
Side-channel leakage example: SPA

SPA on FPGA ECC (Elke de Mulder)
Side-channel leakage example: DPA

Unprotected 8-bit microcontroller. @3 MHz. Power consumption curves of MOV instruction. Operands with different HW. (J. Balasch)
Side-channel leakage example: DPA

32-bit ARM Cortex-A8 @1 GHz
Linux OS, bitsliced AES

(Balasch, Gierlichs, Reparaz, Verbauwhede)
masking

• masking = countermeasure against DPA

• idea: secret sharing \( b = b_1 + b_2 \)

• individual shares tell you nothing about the intermediate
  • power consumption tells you nothing about the intermediate

• main difficulty: compute on masked data
  • AES / RSA / ECC / ring-LWE
ring-LWE review
ring-LWE review

- Works with polynomials in a ring \( R_q = \mathbb{Z}_q[x]/(f(x)) \)
- Key-gen
  - Sample two polynomials \( r_1, r_2 \)
    - public key: \( p = r_1 - g \ast r_2 \)
    - secret key: \( r_2 \)
- Encryption
  - lift the message to ring element \( \tilde{m} \)
  - compute ciphertext \( (c_1, c_2) \) \( c_1 = g \ast e_1 + e_2 \)
  - \( c_2 = p \ast e_1 + e_3 + \tilde{m} \)
- Decryption
  - recover message as \( m = \text{th}(c_1 \ast r_2 + c_2) \)
unprotected ring-LWE decryption

$$m = \text{th}[\text{INTT}(c_1 \ast r_2 + c_2)]$$
unprotected ring-LWE decryption

\[m = \text{th}[\text{INTT}(c_1 \times r_2 + c_2)]\]
unprotected ring-LWE decryption

\[ m = \text{th}[\text{INTT}(c_1 \ast r_2 + c_2)] \]
unprotected ring-LWE decryption

\[ m = \text{th} [\text{INTT}(c_1 \ast r_2 + c_2)] \]
th operation
masking ring-LWE

Two approaches:

1. Split the key
   “A masked ring-LWE implementation”, CHES 2015
   “Masking ring-LWE”, JCEN

2. Split the input
   “Additively Homomorphic ring-LWE Masking” PQCrypto 2016
CHES 2015: split the key

• Core idea: split the secret: \( r = r' + r'' \)

\[
\text{INTT}(r \cdot c_2 + c_1) = \text{INTT}(r' \cdot c_2 + c_1) + \text{INTT}(r'' \cdot c_2).
\]

\[
m = \text{th}[\text{INTT}(c_1 \cdot r_2 + c_2)]
\]
CHES 2015: split the key

- Core idea: split the secret: \( r = r' + r'' \)

\[
\text{INTT}(r \cdot c_2 + c_1) = \text{INTT}(r' \cdot c_2 + c_1) + \text{INTT}(r'' \cdot c_2).
\]
on the masked decoder
on the masked decoder
\[ a = a' + a'' \]
\[ a = a' + a'' \]
$a = a' + a''$
what happened?

• could decode \( \text{th}(a) \) from \( \text{quad}(a') \) and \( \text{quad}(a'') \)
  • \( \text{quad}() \) return only 2 bits, so it will be easy to perform masked computation.

• Idea: decode \( \text{th}(a) \) only from \( \text{quad}(a') \) and \( \text{quad}(a'') \)
  • large compression
decoding rules

• There are 7 other more cases ("rules")

• There are 8 cases that don’t allow inferring th(a)!
Cases where it fails

Solution: refresh the sharing and try again.

\[ a' := a' + \Delta \]

\[ a'' := a'' - \Delta \quad (\text{compute nice } \Delta) \]
Fig. 3: The masked decoder.
implementation costs

unprotected (CHES2014*)
• 1713 LUTs / 830 FFs / 1 DSP
• Fmax = 120 MHz
• 2.8 k cycles (23.5 us)

protected (this work)
• 2014 LUTs / 959 FFs / 1 DSP
• 100 MHz
• 7.5 k cycles (75.2 us)

Parameter set: (n,q,s)=(256,7681,11.32)
Xilinx Virtex-II xc2vp7 FPGA

ECC: Rebeiro et.al. (CHES2012): 289 kcycles * LUT
This work: 151 k cycles*LUTs

* Synthetized on Virtex-II
evaluation

\[ \rho = 0.25, \text{ intermediate: } r'[0] \cdot c_1[0] + c_2[0] \]

\[ \rho = 0.3, \text{ intermediate: } r''[0] \cdot c_1[0] \]

\[ \rho = 0.27, \text{ intermediate: } a'[0] \]

\[ \rho = 0.21, \text{ intermediate: } m'[0] \]
PRNG off
second order

mean curve

mask: $m'[254]$

masked value: $m''[254]$

unmasked value: $m[254]$, first order

unmasked value: $m[254]$, squared traces

unmasked value: $m[254]$, abs traces
second order
ARM masked bitsliced implementation

Apply (hardware) gate-level masking
Substitute 5 macros with secure versions
- SXOR, SMOV, SROTL, SNOT: trivial
- SAND: Trichina gate

```c
#define SAND(c, a, b){
t0 = a[0] & b[0];
t1 = a[0] & b[1];
c[0] = RAND();
t0 = t0 ^ c[0];
t0 = t0 ^ t1;
t1 = a[1] & b[0];
t0 = t0 ^ t1;
t1 = a[1] & b[1];
t0 = t0 ^ t1;
c[1] = t0;
}
```
pqcrypto 2016 masking:
split the inputs
pqcrypto 2016 masking: split the inputs

- ring-LWE decryption is additively homomorphic
pqcrypto 2016 masking: split the inputs

- ring-LWE decryption is additively homomorphict

\[ \text{decryption}(c_1, c_2) \oplus \text{decryption}(c'_1, c'_2) = \text{decryption}(c_1 + c'_1, c_2 + c'_2) \]
pqcrypto 2016 masking: split the inputs

- ring-LWE decryption is additively homomorphic

\[
\text{decryption}(c_1, c_2) \oplus \text{decryption}(c_1', c_2') = \text{decryption}(c_1 + c_1', c_2 + c_2')
\]

- Procedure:
pqcrypto 2016 masking: split the inputs

• ring-LWE decryption is additively homomorphic

\[
\text{decryption}(c_1, c_2) \oplus \text{decryption}(c'_1, c'_2) = \text{decryption}(c_1 + c'_1, c_2 + c'_2)
\]

• Procedure:

1. Internally generate a random message \( m' \) unknown to the adversary
2. Encrypt \( m' \) to \( (c'_1, c'_2) \)
3. Perform decryption\( (c_1 + c'_1, c_2 + c'_2) \) to recover \( m \oplus m' \).
pqcrypto 2016 masking (2)

• Essentially ciphertext blinding
  • adversary looses control over input, can’t place predictions

• We don’t need a masked decoder! 😍

• Need to keep private + **public** keys on decryption

• Not provable secure
experiments
# Comparison of SCA Security, Overhead, and Implementation Difficulty

<table>
<thead>
<tr>
<th></th>
<th>SCA security</th>
<th>overhead</th>
<th>implementation difficulty</th>
</tr>
</thead>
<tbody>
<tr>
<td>unprotected</td>
<td>none</td>
<td>x1</td>
<td>med</td>
</tr>
<tr>
<td>CHES 2015</td>
<td>provable</td>
<td>x3</td>
<td>high</td>
</tr>
<tr>
<td>PQCrypto 2016</td>
<td>not provable</td>
<td>x4</td>
<td>med</td>
</tr>
</tbody>
</table>
Conclusion

CHES 2015 (split key)

• Fully masked ring-LWE decryption
  • outputs Boolean shares
• Manageable overhead: x2.6 cycles wrt unprotected
• Small!
• Bespoke decoder
  • Error rate controlled
• Practical evaluation

PQCrypto 2016 (split input)

• Cheap masking
  • Not provable secure, but increases DPA resistance
• Easy to implement: no new building blocks used, shares handled separately.
• Easy to upgrade from unmasked to masked
• Manageable overhead
error rates
error rates

\[ \frac{p_g}{p_{\text{baseline}}} \]

Iterations \( N \)