HAMPERING FAULT ATTACKS AGAINST LATTICE-BASED SIGNATURE SCHEMES

— COUNTERMEASURES AND THEIR EFFICIENCY

TECHNISCHE
UNIVERSITÄT
DARMSTADT

CROSSING

EMBEDDED
SYSTEMS
WEEK

CODES/ISSS 2017
10/15/2017

Nina Bindel
Juliane Krämer
Johannes Schreiber
TU Darmstadt

PQ CRYPTO & IMPLEMENTATION ATTACKS

- NIST‘s call for PQ submissions, November 2017:

> “submissions that are secured against side channel attacks are considered to be more desirable“

- more attention to implementation attacks during 2nd phase of NIST‘s standardization process

FAULT ANALYSIS OF LATTICE-BASED CRYPTO IN THE LITERATURE

2015

- “Implementation attacks on PQ cryptographic schemes“ by Taha and Eisenbarth:
  - Kamal and Youssef [KY12] → NTRUSign
  - Kamal and Youssef [KY11], [KY13] → NTRUEncrypt

2016

- FA of **signature** schemes
  - Espitau, Fouque, Gérard, and Tibouchi [EFGT16] → GLP, BLISS, ring-TESLA, GPV-NTRU, PassSign
  - Bindel, Buchmann, and Krämer [BBK16] → GLP, BLISS, ring-TESLA

2017

- FA of **encryption** schemes
  - Oder, Schneider, Pöppelmann, and Güneysu [OSPG17] → Ring-LWE

VULNERABILITIES OF LBSS

Fault Attack	Changed Value or Op.	Algorithm	GLP	BLISS	ring-TESLA	Pass-Sign	GPV-NTRU
Randomization	Secret	Sign	●	●	○	?	?
Skipping	Addition	Key Gen	●	●	●	?	?
	Addition	Sign	●	○	○	?	?
	Correctness check	Verify	●	●	●	?	?
	Size check	Verify	●	●	○	?	?
Zeroing	Secret	Key Gen	●	-	○	?	?
	Randomness	Sign	●	●	●	?	?
	Hash polynomial	Sign	●	●	●	?	?
Loop-abort	Loop counter	Key Gen & Sign	●	●	●	●	●

MITITGATION OF ZEROING RANDOMNESS

- introduce new variable
- add secret to random value
- parity bits
- loop counter
- zero counting

EFFECTIVENESS OF ZERO COUNTING

Fault attack	Changed value or op.	Algorithm	GLP	BLISS	ring-TESLA	PassSign	GPV-NTRU
Randomization	Secret	Sign	●	●	○	?	?
Skipping	Addition	Key Gen	●	●	●	?	?
	Addition	Sign	●	○	○	?	?
	Correctness check	Verify	●	●	●	?	?
	Size check	Verify	●	●	○	?	?
Zeroing	✓ Secret	Key Gen	●	-	○	?	?
	✓ Randomness	Sign	●	●	●	?	?
	✓ Hash polynomial	Sign	●	●	●	?	?
Loop-abort	✓ Loop counter	Key Gen & Sign	●	●	●	●	●

CONTRUTION

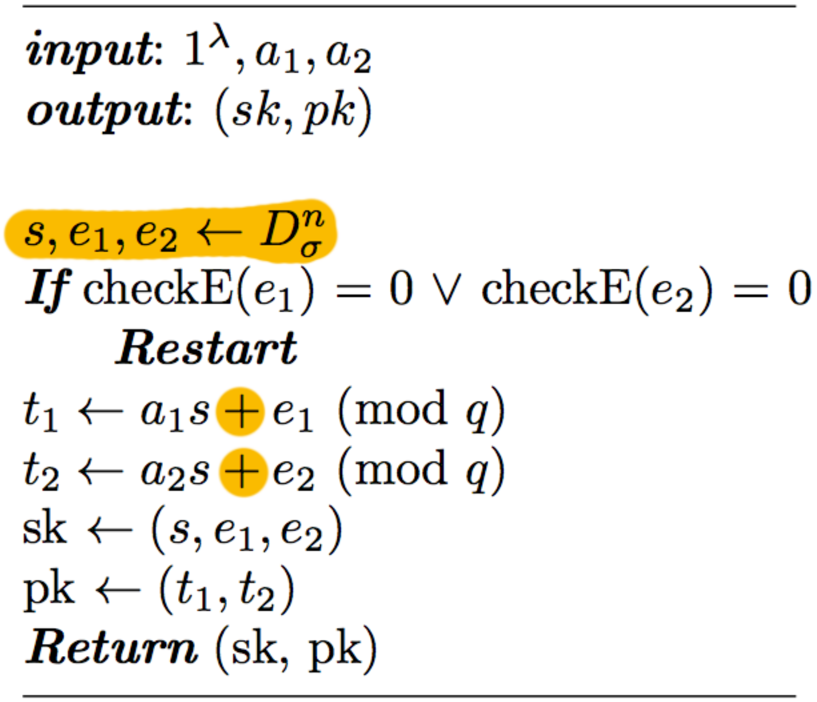
- Investigation of tranfer of different countermeasures
- Implementation of countermeasures at the example of ring-TESLA

OUTLINE

- Zeroing attack on randomness
- Countermeasure against zeroing attack
- Implementation and efficiency

POSSIBLE POINTS OF ATTACK

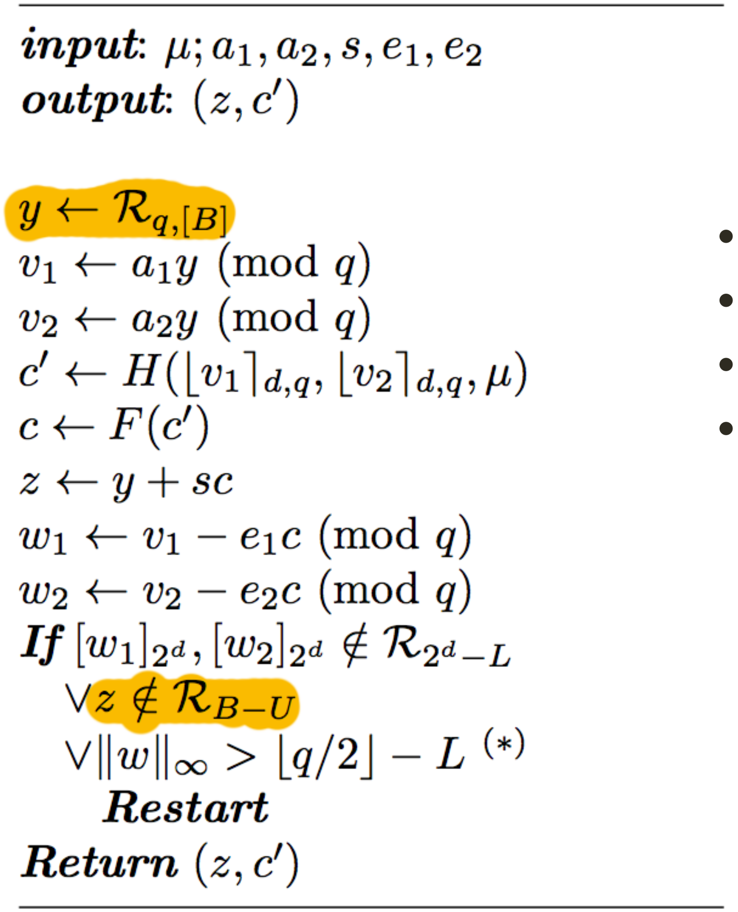
Key generation



```
input: $1^\lambda, a_1, a_2$
output: $(sk, pk)$

$s, e_1, e_2 \leftarrow D_\sigma^n$
If $\mathrm{checkE}(e_1) = 0 \vee \mathrm{checkE}(e_2) = 0$
    Restart
$t_1 \leftarrow a_1 s + e_1 \pmod q$
$t_2 \leftarrow a_2 s + e_2 \pmod q$
$\mathrm{sk} \leftarrow (s, e_1, e_2)$
$\mathrm{pk} \leftarrow (t_1, t_2)$
Return $(\mathrm{sk}, \mathrm{pk})$
```

Signature generation



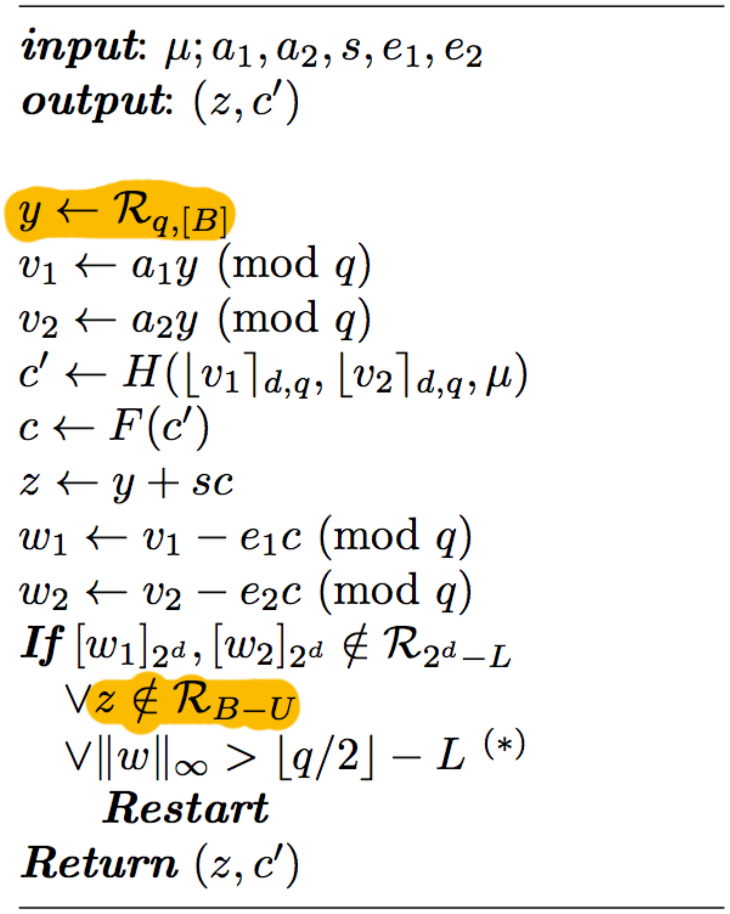
```
input: $\mu; a_1, a_2, s, e_1, e_2$
output: $(z, c')$

$y \leftarrow \mathcal{R}_{q,[B]}$
$v_1 \leftarrow a_1 y \pmod q$
$v_2 \leftarrow a_2 y \pmod q$
$c' \leftarrow H(\lfloor v_1 \rceil_{d,q}, \lfloor v_2 \rceil_{d,q}, \mu)$
$c \leftarrow F(c')$
$z \leftarrow y + sc$
$w_1 \leftarrow v_1 - e_1 c \pmod q$
$w_2 \leftarrow v_2 - e_2 c \pmod q$
If $[w_1]_{2^d}, [w_2]_{2^d} \notin \mathcal{R}_{2^d - L}$
  $\vee z \notin \mathcal{R}_{B-U}$
  $\vee \|w\|_\infty > \lfloor q/2 \rfloor - L$ (*)
    Restart
Return $(z, c')$
```

- Zeroing sk
- Skip addition of e_1, e_2
- Zeroing y
- Skip rejection sampling

ZEROING RANDOMNESS

Signature generation



```
input: $\mu; a_1, a_2, s, e_1, e_2$
output: $(z, c')$

$y \leftarrow \mathcal{R}_{q,[B]}$
$v_1 \leftarrow a_1 y \pmod q$
$v_2 \leftarrow a_2 y \pmod q$
$c' \leftarrow H(\lfloor v_1 \rceil_{d,q}, \lfloor v_2 \rceil_{d,q}, \mu)$
$c \leftarrow F(c')$
$z \leftarrow y + sc$
$w_1 \leftarrow v_1 - e_1 c \pmod q$
$w_2 \leftarrow v_2 - e_2 c \pmod q$
If $[w_1]_{2^d}, [w_2]_{2^d} \notin \mathcal{R}_{2^d - L}$
    $\vee z \notin \mathcal{R}_{B-U}$
    $\vee \|w\|_\infty > \lfloor q/2 \rceil - L$ (*)
        Restart
Return $(z, c')$
```

$y = 0$

$v_1, v_2 = 0$

$c', c \neq 0$

$z = sc$ Compute secret!

Possible Countermeasure ?

```
poly vec_y;
sample_y(vec_y);
```

```
If(vec_y == 0)
{ // restart: sample new
vec_y }
```

Not enough!

Attacks works also if not all coefficients are zero

COUNTERMEASURE AGAINST ZEROING Y

```
poly vec_y;
sample_y(vec_y);
[…]

if (count_zeroes(vec_y) > 8) {
    // restart sign

    continue;
z = y + sc
}
```

Why 8?

```
int count_zeroes(poly p) {
    int zeroes = 0;
    for (int i = 0; i < PARAM_N; i++) {
        if (p[i] == 0.0) {
            zeroes++;
        }
    }
    return zeroes;
}
```

COMPUTING NUMBER OF ZEROS

$\Pr[\, a = 0 \mid a \leftarrow_{\$} [-B, B]\,] \approx \frac{1}{4{,}200{,}000}$ ← very small

Why not define $\mathbf{y_i \neq 0}$?

→ change in distribution might invalidate (parts of) the security reduction

→ find number such that change of distribution is $\leq \frac{1}{2^{128}}$

→ forbid polys with more than 8 zero coefficients

Attention! Depends on instance and distribution!

→ might be necessary to choose value different from 8

2ND ORDER FAULTS

Disadvantage: Skip check of if-condition for more powerful 2nd order faul attacks

```
poly vec_y;
sample_y(vec_y);
[…]

if (count_zeroes(vec_y) > 8) {
    // restart sign

    continue;
z = y + sc
}
```

Possible solution:

```
poly vec_y;
sample_y(vec_y);
[…]
long lambda = check_zeros(vec_y);
poly_mul_constant(lambdaSc,Sc,lambda);
poly_add(result,vec_y,lambdaSc)
[…]
```

ROUTINE CHECK_ZEROS

```
long check_zeros(poly p){
    int zeroes = 0;
    for (int i = 0; i < PARAM_N; i++) {
        if (p[i] == 0.0) {
            zeroes++;
        }
    }
    if (zeroes > 8) {
        return 0;
    } else {
        return 1;
    }
}
```

If randomness y was faulty

→ $\lambda = 0$
→ $z = y + \lambda sc = y$ returned
→ Attacker learns nothing about s

CRITICAL PARTS IN ASSEMBLY

```
Long check_zeros(poly p){
    int nonzeroes = 0;
    for (int i = 0; i < PARAM_N; i++) {
        if (p[i] == 0.0) {
            nonzeroes++;
        }
    }
    if (zeroes > 8) {
        return 0;
    } else {
        return 1;
    }

    return nonzeros;
}
```

```
  asm volatile (
      "cmpl $504, %0;"
      "cmpl $504, %0;"
      "setg %%bl;"
      "setg %%bl;"
      "cmpl $513, %0;"
      "cmpl $513, %0;"
      "setl %%bh;"
      "setl %%bh;"
      "andb %%bh, %%bl;"
      "andb %%bh, %%bl;"
      "movzbl %%bl, %1;"
      "movzbl %%bl, %1;"
      :"=r"(nonzeroes)
      :"r"(nonzeroes)
      :"%ebx"
      );
```

SUMMARY COUNTERMEASURE

Combination of different countermeasures:

- check of number of zero elements
- dummy variable λ to “automatically“ delete secret information in case of fault
- avoiding if-conditions
- limit compiler optimization if plausible
- redundant computation

EFFICIENCY OF THE IMPLEMENTATION

Total signature generation: ~ 330,000 cycles

Countermeasure	Algorithm	Additonal Time [cycles]	Addtional code length [instructions]
Count_zeros	Signature gen.	1,900	407
Count_zeros	Key gen.	3,000	286
Introduce new variable	Key gen.	~10	~10
Rewrite branchless	Verify	~10	~10
Additional rejection	Signature gen.	1100	241
Sample twice	Key gen.	9,000,000	~10

CONCLUSION

- First approaches with respect to fault attacks, but rather simple attacks
  - → need for more sophisticated attacks, comparison with RSA or ECDSA
  - → analysis of encryption schemes and key exchange
- Implementation complicated: might introduce new ones
  - → test effectiveness with software
  - → need careful implementations

More research needed!
Active participation for 2nd
NIST standardization challenge!

TECHNISCHE
UNIVERSITÄT
DARMSTADT

CROSSING

EMBEDDED
SYSTEMS
WEEK

THANKS