Special Session: Hampering fault attacks against lattice-based signature schemes— countermeasures and their efficiency

Nina Bindel
Technische Universität Darmstadt
Germany
nbindel@cdc.tu-darmstadt.de

Juliane Krämer
Technische Universität Darmstadt
Germany
jkraemer@cdc.tu-darmstadt.de

Johannes Schreiber
Technische Universität Darmstadt
Germany

ABSTRACT

Research on physical attacks on lattice-based cryptography has seen some progress in recent years and first attacks and countermeasures have been described. In this work, we perform an exhaustive literature review on fault attacks on lattice-based encryption and signature schemes. Based on this, we provide a complete overview of suggested countermeasures and analyze which of the proposed attacks can prevented by respective countermeasures. Moreover, we show for selected countermeasures how they affect the runtime of the protected operations.

KEYWORDS

lattice-based cryptography, fault attacks, side channel analysis

ACM Reference format:
Nina Bindel, Juliane Krämer, and Johannes Schreiber. 2017. Special Session: Hampering fault attacks against lattice-based signature schemes— countermeasures and their efficiency . In *Proceedings of CODES/ISSS '17 Companion, Seoul, Republic of Korea, October 15–20, 2017,* 3 pages.
https://doi.org/10.1145/3125502.3125546

1 INTRODUCTION

Due to their high efficiency and their strong security properties, lattice-based primitives are a very promising post-quantum candidate to replace currently used public key cryptography. While the security of lattice-based schemes has been deeply analyzed mathematically, only little effort has been spent on the analysis with respect to implementation attacks. First results about powerful attacks exist, but there is no comprehensive work on countermeasures that helps software developers of lattice-based primitives to secure the implementations against physical attacks.

In this work, we provide such an overview for active physical attacks, i.e., fault attacks. Based on all publications on fault attacks on lattice-based signature [2, 5, 9] and encryption schemes [8, 10, 11], we provide a complete overview of suggested countermeasures[1] (cm) for first-order fault attacks. Thus, we assume that an attacker can only induce a single fault, and he cannot circumvent a cm with an additional fault. We analyze which fault attacks can be prevented by the found countermeasures and which signature scheme can be secured with them. A summary of the proposed countermeasures and the attacks they can prevent is given in Table 1. Moreover, we describe for selected countermeasures the impact that they have on the runtime of the protected operations. To this end we implement four countermeasures of the lattice-based signature scheme ring-TESLA [1].

[1]To the best of our knowledge, no attacks on lattice-based key exchange protocols have been published so far.

CODES/ISSS '17 Companion, October 15–20, 2017, Seoul, Republic of Korea

ACM ISBN 978-1-4503-5185-0/17/10…$15.00
https://doi.org/10.1145/3125502.3125546

2 PRELIMINARIES

For $q \in \mathbb{N}$ a prime, we denote by $\mathbb{Z}_q$ the finite field $\mathbb{Z}/q\mathbb{Z}$. Furthermore, we define the rings $\mathcal{R} = \mathbb{Z}[x]/(x^n + 1)$ and $R_q = \mathbb{Z}_q[x]/(x^n + 1)$. For a finite set S, we write $s \leftarrow_\$ S$ to indicate that an element s is sampled uniformly at random from S.

For the description of the lattice-based signature schemes GLP [6], BLISS [3], PassSign [7], and the NTRU-like instantiation of GPV [4], we refer to the original works or to [2, 5]. We briefly describe the scheme ring-TESLA [1] since we explain countermeasures for this scheme in Section 4. The secret key sk is a tuple of polynomials s, e_1, and e_2 with small coefficients; the public key vk consists of $a_1, a_2 \leftarrow_\$ R_q$, $b_1 = a_1 s + e_1 \pmod q$, and $b_2 = a_2 s + e_2 \pmod q$. The signature of a message μ is (z, c') with $z = y + sc$, y a random vector, c an encoding polynomial of the hash value c' that depends on the most significant bits of $a_1 y$, $a_2 y$, and μ. Before returning the signature, rejection sampling is applied. For verification of the signature (c', z), the size of z and the equality of c' and the hash of μ and the most significant bits of $a_1 z - b_1 c$ is checked.

3 OVERVIEW OF COUNTERMEASURES

We provide an exhaustive overview of all countermeasures (cm) from the literature about fault attacks on lattice-based signature and encryption schemes and summarize our findings in Table 1. In the table we consider the following attacks during key generation (KG), sign (S), and verify (V): (1) Randomization of the secret (KG), [2, Sec. III-A]; (2) Zeroing of secret (KG), [2, Sec. V-A]; (3) Skip of addition (KG), [2, Sec. IV-A2]; (4) Skip of addition (S), [2, Sec. IV-B2]; (5) Zeroing of randomness (S), [2, Sec. V-B]; (6) Zeroing of hash polynomial (V), [2, Sec. V-D]; (7) Skip of addition (KG), [2, Sec. IV-A2]; (8) Skip of rejection sampling, [2, Sec. IV-B1]; (9) Skip of correctness check (V), [2, Sec. IV-C1]; (10) Skip of size check (V), [2, Sec. IV-C2]; (11) Zeroing of hash value (S), [2, Sec. V-C]; (12) Loop-abort against Fiat-Shamir signatures(S), [5, Sec. 3]; (13) Loop-abort against Hash-and-Sign signatures (S), [5, Sec. 4].

We emphasize that all countermeasures have to be implemented carefully to ensure that they work as expected. While this is true

in general, it is especially important for those cms where a *constraint on the effectiveness* is shown in Table 1. We believe that these constraints are necessary for the respective cm to be effective. For instance, the cm *loop-counter*, which we think can be transferred to several attacks, requires that the vulnerable operation is implemented with a for loop. Regarding the attack *skip of addition*, this means that the addition of polynomials has to be processed coefficient by coefficient, with a for loop indexing the coefficients. We use the following constraints in the table: (15) attacks prevented in at least 50% of the times; (16) if addition is implemented as a for loop; (17) if check is implemented as a for loop; (18) if zeroing fault is an early abort of a for loop.

sk-correctness: To evaluate the correctness of the secret key during key generation is to use correctness checks or comparisons [2].

Countermeasure	Prevented attacks	Schemes vulnerable to attacks GLP	BLISS	ring-TESLA	Pass-Sign	GPV-NTRU	Constraints
sk-correctness	(1)	●	●				
	(2)	●					
pk-correctness	(3)	●	●	●			
dummy-value	(1)	●	●				
new-variable	(3)	●	●	●			
	(4)	●					
	(2)	●					
	(5)	●	●	●			
	(6)	●	●	●			
add-secret-to-random	(7)	●	●	●			
	(4)	●					
	★ (5)	●	●	●			
	★ (6)	●	●	●			
if-true	(8)	(●)	(●)	(●)			(15)
	(9)	●	●	●			(15)
	(10)	●	●				(15)
loop-counter	★ (3)	●	●	●			(16)
	★ (4)	●					(16)
	★ (9)	●	●	●			(17)
	★ (10)	●	●				(17)
	★ (2)	●					(18)
	★ (5)	●	●	●			(18)
	★ (6)	●	●	●			(18)
	★ (11)						(18)
	(12)	●	●	●	●		
loop-rand.	(12)	●	●	●	●		
zero-counting	★ (2)	●					
	★ (5)	●	●	●			
	★ (6)	●	●	●			
	(12)	●	●	●	●		
redund.-comp.	★ (9)	●	●	●			
	★ (10)	●	●				
	★ (6)	●	●	●			
verify-in-sign	★ (4)	●					
	(13)					●	
parity-bits	★ (1)	●	●				(15)
	★ (2)	●					(15)
	★ (5)	●	●	●			(15)
	★ (6)	●	●	●			(15)
error-det.-codes							
add.-rej.-sampling	★ (8)	(●)	(●)	(●)			
return-comparison	★ (9)	●	●	●			

Table 1: Overview on countermeasures (cm); in column "prevented attacks" ★ denotes that we find that the respective cm also prevents the corresponding attack, otherwise cm was originally proposed; in column "schemes vulnerable to attacks", ● denotes that the attack was originally proposed against the respective scheme

pk-correctness: To prevent returning a leaking public key, advanced correctness checks should be implemented. dummy-value: To ensure that the correct secret key is used during sign, a "dummy" value is computed and multiplied with the values that should be returned. If no fault occurs, the dummy value is equal to one, otherwise it randomizes the signature. In [2] this is described for the GLP scheme A similar idea for a different scheme is also mentioned in [9]. new-variable: A way to circumvent skipping faults addressing the addition or multiplication is to define a new variable to save the resulting sum [2]. add-secret-to-random: A similar method consists in adding secret information to a random value instead of the other way around [2]. We use the name *add-secret-to-random* also to denote similar cms for multiplication . if-true: To prevent skipping attacks such as skipping the rejection sampling condition, an if-condition should be implemented such that a value is returned if the if-condition holds true.This cm prevents skipping attacks of this kind only in roughly 50% of the cases [2]. loop-counter: In [5] it is proposed to introduce a second loop counter and compare its value with the number of intended loops in order to prevent early abortion of a for loop. loop-randomization: To protect the sampling/computation of polynomials against early abortion faults to receive a low-degree polynomial, randomization of the order of the coefficients can be implemented [5]. zero-counting: To check the degree of the computed/sampled polynomial, it can be checked that not too many consecutive coefficients are equal to zero without changing the distribution notably [2, 5]. redundant-computations: In [8], a redundant, rather inefficient computation is sketched. Furthermore, they describe a smart redundant computation, similarly proposed in [9]: Due to the cyclic properties of NTRU-lattices, a second signature is computed on cyclic shifts of values and shifted back before compared with the first signature. We find that this cm can not be applied for ideal-lattice-based Fiat-Shamir signatures. verify-in-sign: Espitau et al. [5] propose to run the verification algorithm before returning the signature as a cm in GPV-NTRU, since by construction of the NTRU-GPV scheme and the fault attack, faulted signatures are not valid with very high probability. parity-bits: To detect faults during decryption, parity bits can be used [8] . error-det.-codes: In [10], error-detecting codes which target algebraic properties of the decryption process are suggested as cm. By construction of the scheme, for for several polynomials that are used during decrypt, the sum of their coefficients is equal modulo a system parameter. This equality is invalidated with high probability by a fault attack and can therefore be checked as a cm. This cm might hamper attacks on signature schemes as well, but to the best of our knowledge, it can not prevent the attacks that we already know. We present to more countermeasures in this work that we describe in Section 4: additional-rejection-sampling and return-comparison.

4 THE EFFICIENCY OF COUNTERMEASURES

We implemented the countermeasure (cm) new-variable to prevent skipping the addition of the error during the key generation . In Listing 1, we show the original, vulnerable key generation algorithm of ring-TESLA next to our implemented cm. To protect against zeroing attacks against the randomness `poly vec_y` in the signature generation of ring-TESLA, we implement the cm

```
# orignal                              #with CM
poly_mul_fixed(poly_T1,poly_S,poly_a1); poly_mul_fixed(poly_A1S,poly_S,poly_a1);
poly_add(poly_T1,poly_T1,poly_E1);      poly_add(poly_T1,poly_A1S,poly_E1);
poly_mul_fixed(poly_T2,poly_S,poly_a2); poly_mul_fixed(poly_A2S,poly_S,poly_a2);
poly_add(poly_T2,poly_T2,poly_E2);      poly_add(poly_T2,poly_A2S,poly_E2);
[...]                                   [...]
compress_pk(pk, poly_T1,poly_T2);       compress_pk(pk,poly_T1,poly_T2);
```

Listing 1: Parts of the key generation algorithm of ring-TESLA with and without countermeasures against skipping the addition of the error polynomial

```
[...]                              int count_zeroes(poly p) {
poly vec_y;                          int zeroes = 0;
sample_y(vec_y);                     for (int i = 0; i < PARAM_N; i++) {
if (count_zeroes(vec_y) > 8){        if (p[i] == 0.0) {
  // restart in Line 1                 zeroes++; }
}                                    }
[...]                                return zeroes;
                                   }
```

Listing 2: Sampling during signature generation with check if zeroing fault was introduced and implementation of `count_zeros()`

```
static void compress_sig(unsigned char *sm,  static double fmodb_u(double x)
unsigned char *c, double vec_z[PARAM_N])
{  int i,k;                                  { int modulus = PARAM_B - PARAM_U;
   int ptr =0;                                 if(x < -modulus)
   int32_t t=0;                                { return x + modulus; }
                                               else if (x > modulus)
   //store the hash value                      { return x - modulus; }
   for (i=0; i<32; i++)                        else
   { sm[ptr++] =c[i]; }                        { return x; }
   for(i=0; i < PARAM_N; i++)                }
   { t = (int32_t) fmodb_u(vec_z[i]);
     for(k=0;k<4;k++)
       sm[ptr++] = ((t>>(8*(k))) & 0xff);
   }
}
```

Listing 3: Changed subroutines `compress_sig()` and implementation of `fmodb_u()`

zero-counting, see Listing 2. Because `vec_y` has 512 integer coefficients, sampled uniformly random over $[-2097151, 2097151]$, the probability of a polynomial having more than eight zero coefficients is less than 2^{-128}. Hence, we consider the change in the probability distribution to be negligible. We implement the cm additional-rejection-sampling within the compression of the signature. The function `fmodb_u()` was added to the original code. We show the modification of `compress_sig()` and implementation of `fmodb_u()` in Listing 3. During the verification algorithm of ring-TESLA, the input value `c_sig` and the computed hash polynomial `c` are compared. If they are equal, `memcmp()` returns 0, otherwise it returns a positive integer (which results in rejected the signature). An attacker who wants his victim to accept an invalid signature, must force the verification algorithm to return 0, which signifies a valid signature. This can be achieved by a skipping attack. Considering only first-order attacks, we can prevent the skipping attack by implementing the cm return-comparison as in Listing 4. The function still returns 0 if the signature is valid and, by definition of `memcmp()`, a nonzero value otherwise (albeit not -1).

```
 [...]
*mlen = smlen-CRYPTO_BYTES;
memcpy(m, sm, *mlen);
return memcmp(c,c_sig,32);
```

Listing 4: Implementation of return-comparison

Algorithm	KG	S		V
CM	new-var.	zero-count.	add.-rej.-sampl.	return-comp.
Without compiler optimization				
w/ CMs	58345806	1768674	1743670	467762
w/o CMs	55308534	1735398		467321
With compiler optimization				
w/ CM	35247385	71354	71075	73739
w/o CMs	33429812	69843		71299

Table 2: Comparison of benchmarks of the ring-TESLA implementation with and without implemented countermeasures and with and without compiler optimization

We benchmarked the implementations of ring-TESLA with and without countermeasures. The benchmarks were recorded on a 4.0 GHz Intel Core i7-6700k. For compilation we used GCC 7.1.0. The benchmarks are given in clock cycles and are averaged over 2,000 runs for the key generation and 20,000 runs for signature generation and verification. Since compiler optimizations might remove some of the countermeasures, we report two benchmarks – with and without the optimization `-Ofast` – in Table 2.

ACKNOWLEDGMENTS

This work has been supported by the the German Research Foundation (DFG) as part of project P1 within the CRC 1119 CROSSING.

REFERENCES

[1] Sedat Akleylek, Nina Bindel, Johannes Buchmann, Juliane Krämer, and Giorgia Azzurra Marson. 2016. An Efficient Lattice-Based Signature Scheme with Provably Secure Instantiation. In AFRICACRYPT 2016. Springer.
[2] Nina Bindel, Johannes A. Buchmann, and Juliane Krämer. 2016. Lattice-Based Signature Schemes and Their Sensitivity to Fault Attacks. In FDTC 2016. IEEE Computer Society.
[3] Léo Ducas, Alain Durmus, Tancrède Lepoint, and Vadim Lyubashevsky. 2013. Lattice Signatures and Bimodal Gaussians. In CRYPTO 2013, Springer (Ed.).
[4] Léo Ducas, Vadim Lyubashevsky, and Thomas Prest. 2014. Efficient Identity-Based Encryption over NTRU Lattices. In ASIACRYPT 2014. Springer.
[5] Thomas Espitau, Pierre-Alain Fouque, Benoît Gérard, and Mehdi Tibouchi. 2016. Loop abort Faults on Lattice-Based Fiat-Shamir & Hash'n Sign signatures. IACR Cryptology ePrint Archive (2016).
[6] Tim Güneysu, Vadim Lyubashevsky, and Thomas Pöppelmann. 2012. Practical Lattice-Based Cryptography: A Signature Scheme for Embedded Systems. In CHES 2012, Springer (Ed.).
[7] Jeffrey Hoffstein, Jill Pipher, John M. Schanck, Joseph H. Silverman, and William Whyte. 2014. Practical Signatures from the Partial Fourier Recovery Problem. In ACNS 2014. Springer.
[8] Abdel Alim Kamal and Amr M. Youssef. 2011. Fault Analysis of the NTRUEncrypt Cryptosystem. IEICE Transactions (2011).
[9] Abdel Alim Kamal and Amr M. Youssef. 2012. Fault analysis of the NTRUSign digital signature scheme. Cryptography and Communications (2012).
[10] Abdel Alim Kamal and Amr M. Youssef. 2013. Strengthening hardware implementations of NTRUEncrypt against fault analysis attacks. Journal of Cryptographic Engineering 3, 4 (01 Nov 2013), 227–240.
[11] Tobias Oder, Tobias Schneider, Thomas Pöppelmann, and Tim Güneysu. 2016. Practical CCA2-Secure and Masked Ring-LWE Implementation. IACR Cryptology ePrint Archive (2016).