# Loop-Abort Faults on Lattice-Based Signature Schemes and Key Exchange Protocols

Thomas Espitau, Pierre-Alain Fouque, Benoît Gérard, and Mehdi Tibouchi

**Abstract**—Although postquantum cryptography is of growing practical concern, not many works have been devoted to implementation security issues related to postquantum schemes.

In this paper, we look in particular at fault attacks against implementations of lattice-based signatures and key exchange protocols. For signature schemes, we are interested both in Fiat–Shamir type constructions (particularly BLISS, but also GLP, PASSSign, and Ring-TESLA) and in hash-and-sign schemes (particularly the GPV-based scheme of Ducas–Prest–Lyubashevsky). For key exchange protocols, we study the implementations of NewHope, Frodo, and Kyber. These schemes form a representative sample of modern, practical latticebased signatures and key exchange protocols, and achieve a high level of efficiency in both software and hardware. We present several fault attacks against those schemes that recover the entire key recovery with only a few faulty executions (sometimes only one), show that those attacks can be mounted in practice based on concrete experiments in hardware, and discuss possible countermeasures against them.

Keywords: Fault Attacks, Digital Signatures, Postquantum Cryptography, Lattices.

## **1** INTRODUCTION

#### 1.1 Lattice-based cryptography

Recent progress in quantum computation [17], the NSA advisory memorandum recommending the transition away from Suite B and to postquantum cryptography [51], as well as the announcement of the NIST standardization process for postquantum cryptography [14] all suggest that research on postquantum schemes, which is already plentiful but mostly focused on theoretical constructions and asymptotic security, should increasingly take into account real-world implementation issues.

Among all flavors of postquantum cryptography, latticebased schemes occupy a position of particular interest, as they rely on well-studied problems and come with uniquely strong security guarantees, such as worst-case to averagecase reductions [58]. A number of works have also focused

- T. Espitau is a PhD student at Sorbonne Universités, UPMC, LIP6. E-mail: thomas.espitau@lip6.fr
- P.-A. Fouque is a Full Professor at Univ Rennes and at Institut Universitaire de France & IRISA.
- E-mail: pierre-alain.fouque@univ-rennes1.fr
- B. Gérard is with DGA.MI and is an Associate Researcher at IRISA. E-mail: benoit.gerard@irisa.fr
- M. Tibouchi is a Distinguished Researcher at NTT Secure Platform Laboratories.
   E-mail: tibouchi.mehdi@lab.ntt.co.jp

Manuscript received May 15, 2017.

on improving the performance of lattice-based cryptography, and actual implementation results suggest that properly optimized schemes may be competitive with, or even outperform, classical factoring and discrete logarithm-based constructions<sup>1</sup>.

The literature on the underlying number-theoretic problems of lattice-based cryptography is extensive (even though concrete bit security is not nearly as well understood as for factoring and discrete logarithms). On the other hand, there is currently a distinct lack of cryptanalytic results on the *physical* security of implementations of lattice-based schemes (or in fact, postquantum schemes in general! [66]). It is well-known that physical attacks, particularly against public-key schemes, are often simpler, easier to mount and more devastating than attacks targeting underlying hardness assumptions: it is often the case that a few bits of leakage or a few fault injections can reveal an entire secret key (the well-known attacks from [7], [9] are typical examples). We therefore deem it important to investigate how fault attacks may be leveraged to recover secret keys in the lattice-based setting, particularly against signature schemes and key exchange protocols, as those primitives are probably the most likely to be deployed in a setting where fault attacks are relevant, and they have also received the most attention in terms of efficient implementations both in hardware and software.

#### 1.2 Implementations of lattice-based signatures

Efficient signature schemes have been typically proved secure in the random oracle model, and can be roughly divided in two families: the hash-and-sign family (which includes schemes like FDH and PSS), as well as signatures based on identification schemes, using the Fiat–Shamir heuristic or a variant thereof. Efficient lattice-based signatures can also be divided along those lines, as observed for example in the survey of practical lattice-based digital signature schemes presented by O'Neill and Güneysu at the NIST workshop on postquantum cryptography [40].

The Fiat–Shamir family is the most developed, with a number of schemes coming with concrete implementations in software, and occasionally in hardware as well. Most

<sup>1.</sup> Usually, those efficient instantiations do not have parameters achieving high security under known reductions to worst-case problems, but the existence of those reductions is usually seen as a good sign that the security assumptions themselves are sound.

schemes in that family follow Lyubashevsky's "Fiat-Shamir with aborts" paradigm [42], which uses rejection sampling to ensure that the underlying identification scheme achieves honest-verifier zero-knowledge. Among latticebased schemes, the exemplar in that family is Lyubashevsky's scheme from EUROCRYPT 2012 [43]. It is, however, of limited efficiency, and had to be optimized to yield practical implementations. This was first carried out by Güneysu et al., who described an optimized hardware implementation of it at CHES 2012 [37], and then to a larger extent by Ducas et al. in their scheme BLISS [20], which includes a number of theoretical improvements and is the topperforming lattice-based signature. It was also implemented in hardware by Pöppelmann et al. [62]. Other schemes in that family include Hoffstein et al.'s PASSSign [39], which incorporates ideas from NTRU, and Akleylek et al.'s Ring-TESLA [1], which boasts a tight security reduction.

On the hash-and-sign side, there were a number of early proposals with heuristic security (and no actual security proofs), particularly GGH [35] and NTRUSign [38], but despite several attempts to patch them<sup>2</sup> they turned out to be insecure. A principled, provable approach to designing lattice-based hash-and-sign signatures was first described by Gentry, Peikert, and Vaikuntanathan in [33], based on discrete Gaussian sampling over lattices. The resulting scheme, GPV, is rather inefficient, even when using faster techniques for lattice Gaussian sampling [49]. However, Ducas, Lyubashevsky and Prest [23] later showed how it could be optimized and instantiated over NTRU lattices to achieve a relatively efficient scheme with particularly short signature size. The DLP scheme is somewhat slower than BLISS in software, but still a good contender for practical latticebased signatures, and seemingly the only one in the hashand-sign family.

#### 1.3 Implementations of lattice-based key exchange

In the last few years, very efficient lattice-based key exchange protocols have been proposed at several security conferences [3], [10], [11], [57] and some of them have been field tested by Microsoft and Google as alternatives to the prequantum key agreements in the TLS handshake protocol. This has shown that lattice-based key exchange protocols can be practical in many contexts and offer credible alternatives to schemes like ECDH, incurring only a 50% performance penalty or so compared to elliptic curves.

The various lattice-based key exchange protocols have a similar structure, relying on Peikert's reconciliation mechanism [57] that allows the two parties to recover the *exact same* secret even if they both have a *noisy* version of the common secret. They mostly differ on the underlying lattice assumptions they are based on. Similarly to the signature setting, one can in particular distinguish between ring-based constructions, like NewHope [3], and constructions using standard lattices, like Frodo [10]. The recent scheme Kyber [12], relying on so-called module lattices, is in some sense a middle ground between those two approaches.

#### 1.4 Our contributions

In this paper, we initiate the study of fault attacks against lattice-based signatures and key exchange protocols, and obtain attacks against all the practical schemes mentioned above.

As noted previously, early lattice-based signature schemes with heuristic security have been broken using standard attacks [32], [34], [52] but recent constructions including [20], [23], [33], [42], [43] are provably secure, and cryptanalysis therefore requires a more powerful attack model. In this work we consider fault attacks.

We present two attacks on signatures, both using a similar type of faults which allows the attacker to cause a loop inside the signature generation algorithm to abort early. Successful loop-abort faults have been described many times in the literature, including against DSA [50] and pairing computations [55], and in our attacks they can be used to recover information about the private signing key. The underlying mathematical techniques used to actually recover the key, however, are quite different in the two attacks.

Our first attack applies to the schemes in the Fiat-Shamir family: we describe it against BLISS [20], [62], and show how it extends to GLP [37], PASSSign [39] and Ring-TESLA [1]. In that attack, we inject a fault in the loop that generates the random "commitment value" y of the sigma protocol associated with the Fiat-Shamir signature scheme. That commitment value is a random polynomial generated coefficient by coefficient and an early loop abort causes it to have abnormally low degree, so that the protocol is no longer zero-knowledge. In fact, this will usually leak enough information that a single faulty signature is enough to recover the entire signing key. More specifically, we show that the faulty signature can be used to construct a point that is very close to a vector in a suitable integer lattice of moderate dimension, and such that the difference is essentially (a subset of) the signing key, which can thus be recovered using lattice reduction.

Our second attack targets the GPV-based hash-and-sign signature scheme of Ducas et al. [23]. In that case, we consider early loop abort faults against the discrete Gaussian sampling in the secret trapdoor lattice used in signature generation. The early loop abort causes the signature to be a linear combination of the last few rows of the secret lattice. A few faulty signatures can then be used to recover the span of those rows and using the special structure of the lattice, we can then use lattice reduction to find one of the rows up to sign, which is enough to completely reconstruct the secret key. In practice, if we can cause loop aborts after up to *m* iterations, we find that m + 2 faulty signatures are enough for full key recovery with high probability.

In addition, we also describe loop-abort fault attacks on three protocols that represent the state of the art for latticebased key exchange, namely NewHope [3], Frodo [10] and Kyber [12]. Although those schemes have a completely different overall structure than the signature schemes mentioned above, they also involve the sampling of random Gaussian secrets coefficient by coefficient during each execution of the protocol. Injecting a fault that causes this random sampling loop to abort early causes abnormally "low-dimensional" secrets to be generated, and this yields

<sup>2.</sup> There is a provably secure scheme due to Aguilar et al. [48] that claims to "seal the leak on NTRUSign", but it actually turns the construction into a Fiat–Shamir type scheme, using rejection sampling à la Lyubashevsky.

a key recovery attack very similar to the one we mount on BLISS and other Fiat–Shamir signatures.

All of our attacks are supported by extensive mathematical simulations in Sage [65]. We also take a close look at the concrete software and hardware implementations of the schemes above, and discuss the concrete feasibility of injecting the required loop-abort faults in practice. We find the attacks to be highly realistic. Moreover, we demonstrate the practicality of those attacks (taking NewHope as an example) in two ways against two types of platforms:

- first, we successfully carry out a simulation of our fault attack against the emulated execution of actual compiled code for the 32-bit SPARC processor LEON3, using a readily available fault simulation tool for that architecture;
- second, we concretely carry out those faults with clock glitches against an 8-bit AVR XMEGA microcontroller, using the power analysis and fault attack testing board ChipWhisperer-Lite [53].

Finally, we discuss several possible countermeasures to protect against our attacks.

#### 1.5 Related work

To the best of our knowledge, the first previous work on fault attacks against lattice-based signatures, and in particular the only one mentioned in the survey of Taha and Eisenbarth [66], is the fault analysis work of Kamal and Youssef on NTRUSign [41]. It is, however, of limited interest since NTRUSign is known to be broken [24], [52]; it also suffers from a very low probability of success.

In 2016, concurrently with our work, Bindel, Buchmann and Krämer [8] described various fault attacks against Fiat– Shamir type signature schemes. Most of the attacks, however, are either in a relatively contrived model (targeting key generation), or require unrealistically many faults and are arguably straightforward (bypassing rejection sampling in signature generation or size/correctness checks in signature verification). One attack described in the paper can be seen as posing a serious threat, namely the one in [8, §IV-B], but it amounts to a weaker variant of our Fiat–Shamir attack, using simple linear algebra rather than lattice reduction. As a result, it requires several hundred faulty signatures, whereas our attack needs only one.

Another interesting concurrent work is the recent cache attack against BLISS of Groot Bruinderink et al. [36]. It uses cache side-channels to extract information about the coefficients of the commitment polynomial **y**, and then lattice reduction to recover the signing key based on that sidechannel information. In that sense, it is similar to our Fiat-Shamir attack. However, since the nature of the information is quite different than in our setting, the mathematical techniques are also quite different. In particular, again, in contrast with our fault attack, that cache attack requires many signatures for a successful key recovery. Further sidechannel attacks on BLISS, including the works of Pessl et al. [60] and Espitau et al. [26], have similar limitations.

A preliminary version of this paper has been published at SAC 2016 in [25]. We extend this version by attacking a larger number of signature schemes (GLP, PASSSign, Ring-TESLA), and generalizing our original attacks to the key exchange settings (against NewHope, Frodo and Kyber). Our discussion of the practical implementations of the faults we consider has also been revised and improved; in particular, the fault simulation on LEON3 compiled code is a novel contribution, as is the practical attack on the AVR microcontroller.

## 1.6 Applicability to NIST-submitted variants of the target schemes

Many of the schemes we consider in this paper have been submitted to the NIST postquantum standardization effort [14] in a slightly modified form.

In particular, the NIST-submitted versions of NewHope, Frodo and Kyber are presented as key encapsulation mechanisms (KEMs) instead of key-exchange protocols in the usual sense. Key exchange then corresponds to key generation for one party and key encapsulation for the other. That modification of the overall structure of the schemes does not significantly affect the actual operations carried out as part of those schemes, and in particular our attacks apply to the KEM versions with essentially no change.

Regarding signatures, none of the schemes discussed in this paper (BLISS, Lyubashevsky's scheme, Ring-TESLA, PASSSign and the DLP scheme of Ducas, Lyubashevsky and Prest) have been submitted to the NIST effort. However, NIST candidate qTESLA [2] is a slight variant of Ring-TESLA, and our attack extends in a straightforward way. One should simply pay attention to the fact that the randomness y is generated as a deterministic function of the message; since that deterministic function involves sampling each coefficient of y one by one, loop-abort faults apply similarly. On a related note, NIST candidate Dilithium [22] can be seen as a modified version of BLISS, using module lattices instead of ideal lattices (and, again, a deterministic generation for y). As a result, just as what happens for Frodo and Kyber, our attack may extend or not depending on the direction in which the matrix of coefficients is filled as part of the generation of y.

PASSSign does not appear to have a counterpart submitted to NIST, and Lyubashevsky's scheme is conceptually related to most submissions, but the low-level behavior is usual different. On the hash-and-sign side, the closest NIST candidate to DLP is Falcon [30]. However, although the main idea of constructing a hash-and-sign signature using Gaussian sampling on an NTRU lattice is the same, Falcon uses a tree-shaped recursive algorithm for lattice sampling instead of a linear loop. Due to this modified structure, our attack does not apply directly to that scheme.

# **2 DESCRIPTION OF THE LATTICE-BASED SIGNA-TURE AND KEY EXCHANGE SCHEMES**

#### 2.1 Notation

For any integer q, we identify the elements of the ring  $\mathbb{Z}_q$  with the integers in [-q/2, q/2). Vectors are considered as column vectors and are written in bold lower case letters; matrices are denoted by upper case letters. We use both the  $\ell_2$  Euclidean norm of vectors,  $\|\mathbf{v}\|_2 = (\sum_i v_i^2)^{1/2}$ , and their  $\ell_{\infty}$ -norm,  $\|\mathbf{v}\|_{\infty} = \max_i |v_i|$ . In ring-based schemes, ring elements (which can be seen as polynomials) are identified

with their vectors of coefficients, and hence also denoted by bold lower case letters.

The Gaussian distribution with standard deviation  $\sigma \in \mathbb{R}$  and center  $c \in \mathbb{R}$  at  $x \in \mathbb{R}$ , is defined by  $\rho_{c,\sigma}(x) = \exp\left(\frac{-(x-c)^2}{2\sigma^2}\right)$  and more generally in higher dimension by  $\rho_{\mathbf{c},\sigma}(\mathbf{x}) = \exp\left(\frac{-(\mathbf{x}-\mathbf{c})^2}{2\sigma^2}\right)$  and when  $\mathbf{c} = \mathbf{0}$ , by  $\rho_{\sigma}(\mathbf{x})$ . The discrete Gaussian distribution over  $\mathbb{Z}$  centered at  $\mathbf{0}$  is defined by  $D_{\sigma}(x) = \rho_{\sigma}(x)/\rho_{\sigma}(\mathbb{Z})$  (or  $D_{\mathbb{Z},\sigma}$ ) and more generally over  $\mathbb{Z}^m$  by  $D_{\sigma}^m(\mathbf{x}) = \rho_{\sigma}(\mathbf{x})/\rho_{\sigma}(\mathbb{Z}^m)$ , where  $\rho_{\sigma}(\mathbb{Z}^m) = \sum_{\mathbf{x} \in \mathbb{Z}^m} \rho_{\sigma}(\mathbf{x})$ .

#### 2.2 Description of BLISS

The BLISS signature scheme [20] is possibly the most efficient lattice-based signature scheme so far. It has been implemented in both software [21] and hardware [62], and boasts performance numbers comparable to classical factoring and discrete-logarithm based schemes. BLISS can be seen as a ring-based optimization of the earlier latticebased scheme of Lyubashevsky [43], sharing the same "Fiat-Shamir with aborts" structure [42]. One can give a simplified description of the scheme as follows: the public key is an NTRU-like ratio of the form  $\mathbf{a}_q = \mathbf{s}_2/\mathbf{s}_1 \mod q$ , where the signing key polynomials  $\mathbf{s}_1, \mathbf{s}_2 \in \mathcal{R} = \mathbb{Z}[\mathbf{x}]/(\mathbf{x}^n + 1)$  are small and sparse. To sign a message  $\mu$ , one first generates commitment values  $\mathbf{y}_1, \mathbf{y}_2 \in \mathcal{R}$  with normally distributed coefficients, and then computes a hash  $\ensuremath{\mathbf{c}}$  of the message  $\mu$  together with  $\mathbf{u} = -\mathbf{a}_q \mathbf{y}_1 + \mathbf{y}_2 \mod q$ . The signature is then the triple  $(\mathbf{c}, \mathbf{z}_1, \mathbf{z}_2)$ , with  $\mathbf{z}_i = \mathbf{y}_i + \mathbf{s}_i \mathbf{c}_i$ , and there is rejection sampling to ensure that the distribution of  $z_i$ is independent of the secret key. Verification is possible because  $\mathbf{u} = -\mathbf{a}_{q}\mathbf{z}_{1} + \mathbf{z}_{2} \mod q$ . The real BLISS scheme, described in full in Figure 1, includes several optimizations on top of the above description. In particular, to improve the repetition rate, it targets a bimodal Gaussian distribution for the  $\mathbf{z}_i$ 's, so there is a random sign flip in their definition. In addition, to reduce key size, the signature element  $\mathbf{z}_2$ is actually transmitted in compressed form  $\mathbf{z}_2^{\dagger}$ , and accordingly, the hash input includes only a compressed version of **u**. These various optimizations are essentially irrelevant for our purposes.

#### 2.3 Description of the GPV-based scheme of [23]

The second signature scheme we consider is the one proposed by Ducas, Lyubashevsky and Prest at ASIACRYPT 2014 [23]. It is an optimization using NTRU lattices of the GPV hash-and-sign signature scheme of Gentry, Peikert and Vaikuntanathan [33], and has been implemented in software by Prest [63]. As in GPV, the signing key is a "good" basis of a certain lattice L (with short, almost orthogonal vectors), and the public key is a "bad" basis of the same lattice (with longer vectors and a large orthogonality defect). To sign a message  $\mu$ , one simply hashes it to obtain a vector **c** in the ambient space of L, and uses the good, secret basis to sample  $\mathbf{v} \in L$  according to a discrete Gaussian distribution of small variance supported on L and centered at  $\mathbf{c}$ . That vector  $\mathbf{v}$ is the signature; it is, in particular, a lattice point very close to c. That property can be checked using the bad, public basis, but that basis is too large to sample such close vectors (this, combined with the fact that the discrete Gaussian

leaks no information about the secret basis is what makes it possible to prove security). The actual scheme of Ducas– Lyubashevsky–Prest, described in Figure 2, uses a lattice of the same form as NTRU:  $L = \{(\mathbf{y}, \mathbf{z}) \in \mathcal{R}^2 \mid \mathbf{y} + \mathbf{z} \cdot \mathbf{h} = 0\}$ , where the public key  $\mathbf{h}$  is again a ratio  $\mathbf{g}/\mathbf{f} \mod q$  of small, sparse polynomials in  $\mathcal{R} = \mathbb{Z}[\mathbf{x}]/(\mathbf{x}^n + 1)$ . The use of such a lattice yields a very compact representation of the keys and makes it possible to compress the signature as well by publishing only the second component of the sampled vector  $\mathbf{v}$ . As a result, this hash-and-sign scheme is very space efficient (even more than BLISS). However, the use of lattice Gaussian sampling makes signature generation significantly slower than BLISS at similar security levels.

#### 2.4 Description of NewHope

The NewHope [3] key exchange protocol is one of the highest-profile key exchange protocols from lattices, and has been awarded the 2016 Internet Defense Prize at the USENIX Security conference. It is based on the so-called Peikert tweak and can be seen as a variant of the scheme of Ding et al. [18]. NewHope improves upon earlier schemes in various ways. On the one hand the authors proposed a refined analysis of the failure probability of the key exchange and its resistance towards quantum adversaries. On the other hand, various tweaks were introduced in the design of the scheme: like Peikert, the authors use the KEM framework, defined by the algorithms (Setup, Gen, Encaps, Decaps); after a successful protocol run both parties share an ephemeral secret key that can be used to protect further communication: the reconciliation allows both parties to derive the session key from an approximately matching pseudorandom ring element. On Alice's side, this element can be written as  $\mathbf{u} \cdot \mathbf{s} = \mathbf{a} \cdot \mathbf{s}' \cdot \mathbf{s} + \mathbf{e}' \cdot \mathbf{s}$  and on Bob's side:  $\mathbf{v} = \mathbf{b} \cdot \mathbf{s}' + \mathbf{e}'' = \mathbf{a} \cdot \mathbf{s} \cdot \mathbf{s}' + \mathbf{e} \cdot \mathbf{s}' + \mathbf{e}''$ , where  $\mathbf{s}, \mathbf{s}'$  are the respective secrets of Alice and Bob, taken as element in the convolution ring  $\mathcal{R}$ . The full outline of the NewHope protocol is given in Figure 3.

#### 2.5 Description of Frodo

The second key exchange scheme we consider has been introduced by Bos et al. at CCS 2016 [10]. This scheme is a practical demonstration of an efficient lattice scheme based on the hardness of LWE (in contrast with NewHope, which relies on the hardness of the *Ring*-LWE problem). Its performance has been evaluated in a "real-world" setting by benchmarking its implementation within the TLS protocol, when coupled with ECDSA certificates. As in NewHope key exchange, the reconciliation elements can be written as  $\mathbf{B}' \cdot \mathbf{S} = \mathbf{S}' \cdot \mathbf{A} \cdot \mathbf{S} + \mathbf{E}' \cdot \mathbf{S}$  and  $\mathbf{V} = \mathbf{S}' \cdot \mathbf{B} + \mathbf{E}'' =$  $\mathbf{S}' \cdot \mathbf{A} \cdot \mathbf{S} + \mathbf{S}' \cdot \mathbf{E} + \mathbf{E}''$  for Alice and Bob, but in contrast with NewHope, the secrets are no longer elements of a ring, but vectors of integers. The full outline of the Frodo protocol is given in Figure 4.

# **3** ATTACK ON FIAT-SHAMIR TYPE LATTICE-BASED SIGNATURES

The first fault attack that we consider targets the latticebased signature schemes of Fiat–Shamir type, and specifically the generation of the random "commitment" element

1: function KEYGEN() 2: sample $\mathbf{f}, \mathbf{g} \in \mathcal{R} = \mathbb{Z}[\mathbf{x}]/(\mathbf{x}^n + 1)$ , uniformly with $\lceil \delta_1 n \rceil$ coefficients in $\{\pm 1\}$ , $\lceil \delta_2 n \rceil$ coefficients in $\{\pm 2\}$ and other equal to zero 3: $\mathbf{S} = (\mathbf{s}_1, \mathbf{s}_2)^T \leftarrow (\mathbf{f}, 2\mathbf{g} + 1)^T$ 4: if $N_{\kappa}(\mathbf{S}) \ge C^2 \cdot 5 \cdot (\lceil \delta_1 n \rceil + 4\lceil \delta_2 n \rceil) \cdot \kappa$ then restart 5: $\mathbf{a}_q = (2\mathbf{g} + 1)/\mathbf{f} \mod q$ (restart if $\mathbf{f}$ is not invertible) 6: return $(pk = \mathbf{a}_1, sk = \mathbf{S})$ where $\mathbf{a}_1 = 2\mathbf{a}_q \mod 2q$ 7: end function 1: function VERIFY( $\mu, pk = \mathbf{a}_1, (\mathbf{z}_1, \mathbf{z}_2^{\dagger}, \mathbf{c})$ ) 2: if $\ (\mathbf{z}_1, 2^d \cdot \mathbf{z}_2^{\dagger})\ _2 > B_2$ then reject	1: function SIGN( $\mu, pk = \mathbf{a}_1, sk = \mathbf{S}$ ) 2: $\mathbf{y}_1 \leftarrow D_{\mathbb{Z},\sigma'}^n \mathbf{y}_2 \leftarrow D_{\mathbb{Z},\sigma}^n$ 3: $\mathbf{u} = \zeta \cdot \mathbf{a}_1 \cdot \mathbf{y}_1 + \mathbf{y}_2 \mod 2q$ 4: $\mathbf{c} \leftarrow H(\lfloor \mathbf{u} \rfloor_d \mod p, \mu)$ 5: choose a random bit $b$ 6: $\mathbf{z}_1 \leftarrow \mathbf{y}_1 + (-1)^b \mathbf{s}_1 \mathbf{c}$ 7: $\mathbf{z}_2 \leftarrow \mathbf{y}_2 + (-1)^b \mathbf{s}_2 \mathbf{c}$ 8: rejection sampling: restart to step 2 except with probability $1/(M \exp(-  \mathbf{Sc}  /(2\sigma^2)) \cosh(\langle \mathbf{z}, \mathbf{Sc} \rangle / \sigma^2))$ 9: $\mathbf{z}_2^{\dagger} \leftarrow (\lfloor \mathbf{u} \rfloor_d - \lfloor \mathbf{u} - \mathbf{z}_2 \rceil_d) \mod p$ 10: return $(\mathbf{z}_1, \mathbf{z}_2^{\dagger}, \mathbf{c})$
1: function VERIFY $(\mu, pk = \mathbf{a}_1, (\mathbf{z}_1, \mathbf{z}_2^{\dagger}, \mathbf{c}))$	9: $\mathbf{z}_2^{\dagger} \leftarrow ( \mathbf{u} _d -  \mathbf{u} - \mathbf{z}_2 _d) \mod p$
2: if $\ (\mathbf{z}_1, 2^d \cdot \mathbf{z}_2^{\dagger})\ _2 > B_2$ then reject	10: return $(\mathbf{z}_1, \mathbf{z}_2^{\dagger}, \mathbf{c})$
3: if $\ (\mathbf{z}_1, 2^d \cdot \mathbf{z}_2^{\dagger})\ _{\infty} > B_{\infty}$ then reject	11: end function
4: <b>accept iff</b> $\mathbf{c} = H(\lfloor \zeta \cdot \mathbf{a}_1 \cdot \mathbf{z}_1 + \zeta \cdot q \cdot \mathbf{c} \rceil_d + \mathbf{z}_2^{\dagger} \mod p, \mu)$	
5: end function	

Fig. 1. Description of the BLISS signature scheme. The random oracle H takes its values in the set of polynomials in  $\mathcal{R}$  with 0/1 coefficients and Hamming weight exactly  $\kappa$ , for some small constant  $\kappa$ . The value  $\zeta$  is defined as  $\zeta \cdot (q-2) = 1 \mod 2q$ . The authors of [20] propose four different sets of parameters with security levels at least 128 bits. The interesting parameters for us are: n = 512, q = 12289,  $\sigma \in \{215, 107, 250, 271\}$ ,  $(\delta_1, \delta_2) \in \{(0.3, 0), (0.42, 0.03), (0.45, 0.06)\}$  and  $\kappa \in \{23, 30, 39\}$ . We refer to the original paper for other parameters and for the definition of notation like  $N_{\kappa}$  and  $|\cdot|_d$ , as they are not relevant for our attack. The underlined instruction (Step 2 in SIGN) is where we introduce our faults.

1: **function** KEYGEN(n, q)1: **function** GAUSSIANSAMPLER( $\mathbf{B}, \sigma, \mathbf{c}$ )  $\triangleright \mathbf{b}_i$  (resp.  $\mathbf{b}_i$ ) denote  $\mathbf{f} \leftarrow D_{\sigma_0}^n, \mathbf{g} \leftarrow D_{\sigma_0}^n$ if  $\|(\mathbf{g}, -\mathbf{f})\|_2 > \sigma$  then restart  $\triangleright \, \sigma_0 = 1.17 \sqrt{q/2n} \\ \triangleright \, \sigma = 1.17 \sqrt{q}$ 2: the rows of **B** (resp. of its Gram–Schmidt matrix  $\mathbf{B}$ ) 3:  $\mathbf{v} \leftarrow \mathbf{0}$ 2: if  $\|(\overline{qf}, \overline{qg}, \overline{qg})\|_2 > \sigma$  then restart using the extended Euclidean algorithm, compute 4: for i = 2n down to 1 do 3: 5:  $c' \leftarrow \langle \mathbf{c}, \widetilde{\mathbf{b}}_{\mathbf{i}} \rangle / \| \widetilde{\mathbf{b}}_i \|_2^2$ 4:  $\rho_f, \rho_g \in \mathcal{R}$  and  $R_f, R_g \in \mathbb{Z}$  s.t.  $\rho_f \cdot \mathbf{f} = R_f \mod \mathbf{x}^n + 1$ and  $\rho_g \cdot \mathbf{g} = R_g \mod \mathbf{x}^n + 1$ 5:  $\sigma' \leftarrow \sigma / \| \mathbf{b}_i \|_2$ 6:  $r \leftarrow D_{\mathbb{Z},\sigma',c'}$ if  $gcd(R_f, R_g) \neq 1$  or  $gcd(R_f, q) \neq 1$  then restart 6: 7:  $\mathbf{c} \leftarrow \mathbf{c} - r\mathbf{b}_i$  and  $\mathbf{v} \leftarrow \mathbf{v} + r\mathbf{b}_i$ using the extended Euclidean algorithm, compute  $u, v \in$ 7: 8: end for  $\mathbb{Z}$  s.t.  $u \cdot R_f + v \cdot R_g = 1$ **return**  $\mathbf{v} > \mathbf{v}$  sampled according to the lattice Gaussian  $\mathbf{F} \leftarrow qv\rho_g, \mathbf{G} \leftarrow -qu\rho_f$ 8: distribution  $D_{L,\sigma,\mathbf{c}}$ 9: 10: end function 
$$\begin{split} \mathbf{k} \leftarrow \left\lfloor \frac{\mathbf{F} \cdot \mathbf{\bar{f}} + \mathbf{G} \cdot \mathbf{\bar{f}}}{\mathbf{f} \mathbf{\bar{f}} + \mathbf{g} \mathbf{\bar{g}}} \right\rceil \in \mathcal{R} \\ \mathbf{F} \leftarrow \mathbf{\bar{F}} - \mathbf{k} \cdot \mathbf{f}, \mathbf{G} \leftarrow \mathbf{G} - \mathbf{k} \cdot \mathbf{g} \end{split}$$
10: 1: function SIGN( $\mu$ ,  $sk = \mathbf{B}$ ) 11: 2:  $\mathbf{c} \leftarrow H(\mu) \in \mathbb{Z}_q^n$ until k=0 12:  $\begin{aligned} \mathbf{h} &\leftarrow \mathbf{g} \cdot \mathbf{f}^{-1} \bmod q \\ \mathbf{B} &\leftarrow \begin{pmatrix} \mathbf{M}_{\mathbf{g}} & -\mathbf{M}_{\mathbf{f}} \\ \mathbf{M}_{\mathbf{G}} & -\mathbf{M}_{\mathbf{F}} \end{pmatrix} \in \mathbb{Z}^{2n \times 2n} \end{aligned}$  $(\mathbf{y}, \mathbf{z}) \leftarrow (\mathbf{c}, \mathbf{0}) - \text{GaussianSampler}(\mathbf{B}, \sigma, (\mathbf{c}, \mathbf{0}))$ 3: 13: 4:  $\triangleright$  **y**, **z** are short and satisfy **y** + **z** · **h** = **c** mod q 14:  $\triangleright$  short lattice basis 5: return z 6: end function return  $sk = \mathbf{B}, pk =$ 15: 16: end function 1: function VERIFY( $\mu, pk = \mathbf{h}, \mathbf{z}$ ) accept iff  $\|\mathbf{z}\|_2 + \|H(\mu) - \mathbf{z} \cdot \mathbf{h}\|_2 \le \sigma \sqrt{2n}$ 2: 3: end function

Fig. 2. Description of the GPV-based signature scheme of Ducas–Lyubashevsky–Prest. The random oracle H takes its values in  $\mathbb{Z}_q^n$ . We denote by  $\mathbf{f} \mapsto \overline{\mathbf{f}}$  the conjugation involution of  $\mathcal{R} = \mathbb{Z}[\mathbf{x}]/(\mathbf{x}^n + 1)$ , i.e. for  $\mathbf{f} = \sum_{i=0}^{n-1} f_i x^i$ ,  $\overline{\mathbf{f}} = f_0 - \sum_{i=1}^{n-1} f_{n-i} x^i$ .  $\mathbf{M}_{\mathbf{a}}$  represents the matrix of the multiplication by  $\mathbf{a}$  in the polynomial basis of  $\mathcal{R}$ , which is skew-circulant of dimension n. For 128 bits of security, the authors of [23] recommend the parameters n = 256 and  $q \approx 2^{10}$ . The constant 1.17 is an approximation of  $\sqrt{e/2}$ . The underlined step (the main loop in GAUSSIANSAMPLER) is where we introduce our faults.

in the underlying sigma protocols, which is denoted by **y** in our descriptions. That element consists of one or several polynomials generated coefficient by coefficient, and the idea of the attack is to introduce a fault in that random sampling to obtain a polynomial of abnormally small degree, in which case signatures will leak information about the private signing key. For simplicity's sake, we introduce

the attack against BLISS in particular, but it works against the other Fiat–Shamir type schemes (GLP, PASSSign and Ring-TESLA) with almost no changes: see the Appendix for details.

In BLISS, the commitment element actually consists of two polynomials  $(y_1, y_2)$ , and it suffices to attack  $y_1$ . Intuitively,  $y_1$  should mask the secret key element  $s_1$  in the

Alice		Bob
$seed \leftarrow \$ \ \{0,1\}^{256}$		
$\mathbf{a} \leftarrow SHAKE-128(seed)$		
$\mathbf{s}, \mathbf{e} \leftarrow \psi_{16}^n$	(7	$\mathbf{s}', \mathbf{e}', \mathbf{e}'' \leftarrow \psi_{16}^n$
$\mathbf{b} \leftarrow \mathbf{a} \cdot \mathbf{s} + \mathbf{e}$	$\xrightarrow{(\mathbf{b}, seed)}$	$\mathbf{a} \leftarrow SHAKE-128(\mathit{seed})$
		$\mathbf{u} \gets \mathbf{a} \cdot \mathbf{s}' + \mathbf{e}'$
		$\mathbf{v} \leftarrow \mathbf{b} \cdot \mathbf{s}' + \mathbf{e}''$
$\mathbf{v}' \gets \mathbf{u} \cdot \mathbf{s}$	$\overleftarrow{(\mathbf{u},\mathbf{r})}$	$\mathbf{r} \gets ^{\$} \mathrm{HelpRec}(\mathbf{v})$
$\nu \gets \operatorname{Rec}(\mathbf{v}', \mathbf{r})$		$\nu \gets \operatorname{Rec}(\mathbf{v}, \mathbf{r})$
$\mu \gets \text{SHA3-256}(\nu)$		$\mu \gets \text{SHA3-256}(\nu)$

Fig. 3. Description of the NewHope scheme. The security parameters given are  $q = 12289 < 2^{14}$ , n = 1024. The REC and HELPREC subprocedures are encoding and decoding functions between bits and coordinates in a small dimension lattice, as fully analyzed in [3]. Note that computations are taken modq

Alice		Bob
$seed \leftarrow^{\$} \mathcal{U}(\{0,1\}^s)$		
$\mathbf{A} \leftarrow Gen(seed)$		
$\mathbf{S}, \mathbf{E} \leftarrow \chi(\mathbb{Z}_q^{n  imes ar{n}})$		$\mathbf{S}', \mathbf{E}', \mathbf{E}'' \leftarrow^{\$} \chi(\mathbb{Z}_q^{\bar{m} \times n})$
$\mathbf{B} \leftarrow \mathbf{A} \cdot \mathbf{S} + \mathbf{E}$	$\xrightarrow{(\mathbf{B}, seed)}$	$\mathbf{A} \leftarrow Gen(seed)$
		$\mathbf{B}' \gets \mathbf{S}' \cdot \mathbf{A} + \mathbf{E}'$
		$\mathbf{V} \leftarrow \mathbf{S}' \cdot \mathbf{B} + \mathbf{E}''$
	$\overleftarrow{(\mathbf{B'},\mathbf{C})}$	$\mathbf{C} \leftarrow \langle \mathbf{V}  angle_{2^B}$
$K \gets \operatorname{Rec}(\mathbf{B}' \cdot \mathbf{S}, \mathbf{C})$		$K \leftarrow \lfloor \mathbf{V} \rceil_{2^B}$

Fig. 4. Description of the Frodo scheme with parameters  $(n, q, \chi)$ , and protocol specific parameters  $\bar{n}, \bar{m}, \bar{B} \in \mathbb{Z}$ . The matrix  $\mathbf{A} \in \mathbb{Z}^{n \times n}$  is generated from seed via a pseudo-random function GEN. Recommended parameters are  $n = 752, q = 2^{15}, \chi$  an approximate discrete Gaussian distribution of variance 1.75. Note that computations are taken  $\operatorname{mod} q$ .

relation  $\mathbf{z}_1 = \pm \mathbf{s}_1 \mathbf{c} + \mathbf{y}_1$ , and therefore modifying the distribution of  $\mathbf{y}_1$  should cause some information about  $\mathbf{s}$  to leak in signatures. The actual picture in the Fiat–Shamir with aborts paradigm is in fact slightly different (namely, rejection sampling ensures that the distribution of  $\mathbf{z}_1$  is independent of  $\mathbf{s}_1$ , but only does so under the assumption that  $\mathbf{y}_1$  follows the correct distribution), but the end result is the same: perturbing the generation of  $\mathbf{y}_1$  should lead to secret key leakage.

Concretely speaking, in BLISS,  $\mathbf{y}_1 \in \mathcal{R}_q$  is a ring element generated according to a discrete Gaussian distribution<sup>3</sup>, and that generation is typically carried out coefficient by coefficient in the polynomial representation. Therefore, if we can use faults to cause an early termination of that generation process, we should obtain signatures in which the element  $y_1$  is actually a low-degree polynomial. If the degree is low enough, we will see that this reveals the whole secret key right away, from a single faulty signature!

Indeed, suppose that we can obtain a faulty signature obtained by forcing a termination of the loop for sampling  $\mathbf{y}_1$  after the *m*-th iteration, with  $m \ll n$ . Then, the resulting polynomial  $\mathbf{y}_1$  is of degree at most m - 1. As part of the faulty signature, we get the pair  $(\mathbf{c}, \mathbf{z}_1)$  with  $\mathbf{z}_1 = (-1)^b \mathbf{s}_1 \mathbf{c} + \mathbf{y}_1$ . Without loss of generality, we may assume that b = 0 (we will recover the whole secret key only up to sign, but in BLISS,  $(\mathbf{s}_1, \mathbf{s}_2)$  and  $(-\mathbf{s}_1, -\mathbf{s}_2)$  are clearly equivalent secret keys). Moreover, with high probability,  $\mathbf{c}$  is invertible: if we heuristically assume that  $\mathbf{c}$  behaves like a random element of the ring in terms of invertibile with probability about  $(1 - 1/q)^n$ , which is over 95% for all proposed BLISS parameters. We thus get an equation of the form:

$$\mathbf{c}^{-1}\mathbf{z}_1 - \mathbf{s}_1 \equiv \mathbf{c}^{-1}\mathbf{y}_1 \equiv \sum_{i=0}^{m-1} y_{1,i}\mathbf{c}^{-1}\mathbf{x}^i \pmod{q} \quad (1)$$

Thus, the vector  $\mathbf{v} = \mathbf{c}^{-1}\mathbf{z}_1$  is very close to the sublattice of  $\mathbb{Z}^n$  generated by  $\mathbf{w}_i = \mathbf{c}^{-1}\mathbf{x}^i \mod q$  for i = 0, ..., m - 1 and  $q\mathbb{Z}^n$ , and the difference should be  $\mathbf{s}_1$ .

The previous lattice is of full rank in  $\mathbb{Z}^n$ , so the dimension is too large to apply lattice reduction directly. However, the relation given by equation (1) also holds for all subsets of indices. More precisely, let J be a subset of  $\{0, \ldots, n-1\}$  of cardinality  $\ell$ , and  $\varphi_J \colon \mathbb{Z}^n \to \mathbb{Z}^J$  be the projection  $(u_j)_{0 \leq j < n} \mapsto (u_j)_{j \in J}$ . Then we also have that  $\varphi_J(\mathbf{z}_1)$  is a close vector to the sublattice  $L_J$  of  $\mathbb{Z}^J$  generated by  $q\mathbb{Z}^J$  and the images under  $\varphi_J$  of the  $\mathbf{w}_i$ 's; and the difference should be  $\varphi_J(\mathbf{s}_1)$ .

Equivalently, using Babai's nearest plane approach to the closest vector problem [4], we hope to show that  $(\varphi_J(\mathbf{s}_1), B)$ , for a suitably chosen positive constant B, is the shortest vector in the sublattice  $L'_J$  of  $\mathbb{Z}^J \times \mathbb{Z}$  generated by  $(\varphi_J(\mathbf{v}), B)$  as well as the vectors  $(\varphi_J(\mathbf{w}_i), 0)$  and  $q\mathbb{Z}^J \times \{0\}$ .

The volume of  $L'_J$  is given by:

$$\operatorname{vol}(L'_J) = B \cdot \operatorname{vol}(L_J) = B \cdot \frac{\operatorname{vol}(q\mathbb{Z}^J)}{[L_J : q\mathbb{Z}^J]} = Bq^{\ell-r}$$

where *r* is the rank of the family  $(\varphi_J(\mathbf{w}_0), \dots, \varphi_J(\mathbf{w}_{m-1}))$ in  $\mathbb{Z}_q^J$ , which is at most *m*. Hence  $\operatorname{vol}(L'_J) \ge Bq^{\ell-m}$ , and the Gaussian heuristic predicts that the shortest vector should be of norm:

$$\lambda_J \approx \sqrt{\frac{\ell+1}{2\pi e}} \cdot \operatorname{vol}(L'_J)^{1/(\ell+1)}$$
  
$$\gtrsim \sqrt{\frac{\ell+1}{2\pi e}} \cdot B^{1/(\ell+1)} q^{1-(m+1)/(\ell+1)}$$

Thus, we expect that  $(\varphi_J(\mathbf{s}_1), B)$  will actually be the shortest vector of  $L'_J$  provided that its norm is significantly smaller than this bound  $\lambda_J$ . Now  $\varphi_J(\mathbf{s}_1)$  has roughly  $\delta_1 \ell$ entries equal to  $\pm 1$ ,  $\delta_2 \ell$  entries equal to  $\pm 2$  and the rest are zeroes; therefore, the norm of  $(\varphi_J(\mathbf{s}_1), B)$  is around  $\sqrt{(\delta_1 + 4\delta_2)\ell + B^2}$ . Let us choose  $B = \lceil \sqrt{\delta_1 + 4\delta_2} \rceil$ . The

<sup>3.</sup> In the other Fiat–Shamir schemes such as [37], the distribution of each coefficient is uniform in some interval rather than Gaussian, but this doesn't affect our attack strategy at all.

condition for  $s_1$  to be the shortest vector  $L_J$  can thus be written as:

$$\sqrt{(\delta_1 + 4\delta_2) \cdot (\ell + 1)} \ll \sqrt{\frac{\ell + 1}{2\pi e}} \cdot B^{1/(\ell + 1)} q^{1 - (m+1)/(\ell + 1)}$$

or equivalently:

$$\ell + 1 \gtrsim \frac{m + 1 + \frac{\log \sqrt{\delta_1 + 4\delta_2}}{\log q}}{1 - \frac{\log \sqrt{2\pi e(\delta_1 + 4\delta_2)}}{\log q}}.$$
(2)

The denominator of the right-hand side of (2) ranges from about 0.91 for the BLISS–I and BLISS–II parameter sets down to about 0.87 for BLISS–IV. In all cases, we thus expect to recover  $\varphi_J(\mathbf{s}_1)$  if we can solve the shortest vector problem in a lattice of dimension slightly larger than m. This is quite feasible with the LLL algorithm for m up to about 50, and with BKZ for m up to 100 or so.

To complete the attack, it suffices to apply the above to a family of subsets J of  $\{0, \ldots, n-1\}$  covering the whole set of indices, which reveals the entire vector  $\mathbf{s}_1$ . The second component of the secret key is then obtained as  $\mathbf{s}_2 = \mathbf{a}_1 \mathbf{s}_1 / 2 \mod q$ .

Simulations using our Sage implementation confirm the theoretical estimates, and show that full key recovery can be achieved in practice in a time ranging from a few seconds to a few hours depending on m. Detailed experimental results are reported in Table 1.

**Remark 1.** A variant of that attack which is possibly slightly simpler consists in observing that  $\varphi_J(\mathbf{s}_1)$  should be the shortest vector in the lattice generated by  $L_J$  and  $\varphi_J(\mathbf{v})$ . The bound on the lattice dimension becomes essentially the same as (2). The drawback of that approach, however, is that we obtain each  $\varphi_J(\mathbf{s}_1)$  up to sign, and so one needs to use overlapping subsets J to ensure the consistency of those signs.

**Remark 2.** Note that a single *faulty* signature is enough to recover the entire secret key with this attack, a successful key recovery may require several *fault injections*. This is due to rejection sampling: after a faulty  $y_1$  is generated, the whole signature may be thrown away in the rejection step. On average, the fault attacker may thus need to inject the same number of faults as the repetition rate of the scheme, which is a small constant ranging from 1.6 to 7.4 depending on chosen parameters [20], and even smaller with the improved analysis of BLISS–B [19].

**Remark 3.** Finally, we note that in certain hardware settings, fault injection may yield a faulty value of  $y_1$  in which all coefficients upwards of a certain degree bound are non zero but equal to a common constant (see the discussion in §7.3). Our attack adapts to that setting in a straightforward way: that simply means that  $y_1$  is a linear combination of the  $x^i$  for small *i* and of the all-one vector (1, ..., 1), so it suffices to add that vector to the set of lattice generators.

# 4 ATTACK ON HASH-AND-SIGN TYPE LATTICE-BASED SIGNATURES

Our second attack targets the practical hash-and-sign signature scheme of Ducas, Lyubashevsky and Prest [23], which is based on GPV-style lattice trapdoors. More precisely, the faults we consider are again early loop aborts, this time in the lattice-point Gaussian sampling routine used in signature generation.

#### 4.1 Description of the attack

The attack can be described as follows. A correctly generated signature element is of the form  $\mathbf{z} = \mathbf{R} \cdot \mathbf{f} + \mathbf{r} \cdot \mathbf{F} \in \mathbb{Z}[\mathbf{x}]/(\mathbf{x}^n + 1)$ , where the short polynomials  $\mathbf{f}$  and  $\mathbf{F}$  are components of the secret key, and  $\mathbf{r}, \mathbf{R}$  are short random polynomials sampled in such a way that  $\mathbf{z}$  follows a suitable Gaussian distribution. In fact,  $\mathbf{r}, \mathbf{R}$  are generated coefficient by coefficient, in a single loop with 2n iterations, going from the top-degree coefficient of  $\mathbf{r}$  down to the constant coefficient of  $\mathbf{R}$ .

Therefore, if we inject a fault aborting the loop after  $m \le n$  iterations (in the first half of the loop), the resulting signature simply has the form:

$$\mathbf{z} = r_0 \mathbf{x}^{n-1} \mathbf{F} + r_1 \mathbf{x}^{n-2} \mathbf{F} + \dots + r_{m-1} \mathbf{x}^{n-m} \mathbf{F}.$$

Any such faulty signature is, in particular, in the lattice *L* of rank *m* generated by the vectors  $\mathbf{x}^{n-i}\mathbf{F}$ , i = 1, ..., m, in  $\mathbb{Z}[\mathbf{x}]/(\mathbf{x}^n + 1)$ .

Suppose then that we obtain several signatures  $\mathbf{z}^{(1)}, \ldots, \mathbf{z}^{(\ell)}$  of the previous form. If  $\ell$  is large enough (slightly more than m is sufficient; see §4.2 below for an analysis of success probability depending on  $\ell$ ), the corresponding vectors will then generate the lattice L. Assuming the lattice dimension is not too large, we should then be able to use lattice reduction to recover a shortest vector in L, which is expected to be one of the signed shifts  $\pm \mathbf{x}^{n-i}\mathbf{F}$ ,  $i = 1, \ldots, m$ , since the polynomial  $\mathbf{F}$  is constructed in a such a way as to make it quite short relative to the Gram–Schmidt norm of the ideal lattice it generates. Hence, we can recover  $\mathbf{F}$  among a small set of at most 2m candidates.

And recovering **F** is actually sufficient to reconstruct the entire secret key  $(\mathbf{f}, \mathbf{g}, \mathbf{F}, \mathbf{G})$ , and hence completely break the scheme. This is due to the particular structure of the NTRU lattice. On the one hand, **G** is linked to **F** via the public key polynomial **h**:  $\mathbf{G} = \mathbf{F} \cdot \mathbf{h} \mod q$ , so we obtain it directly. On the other hand, the basis completion algorithm of Hoffstein et al. [38] allows to recover the pair  $(\mathbf{f}, \mathbf{g})$  from  $(\mathbf{F}, \mathbf{G})$  via the defining relation  $\mathbf{f} \cdot \mathbf{G} - \mathbf{g} \cdot \mathbf{F} = q$ . This is actually used in the opposite direction in the key generation algorithm of the scheme of Ducas et al. (i.e. they construct  $(\mathbf{F}, \mathbf{G})$  from  $(\mathbf{f}, \mathbf{g})$ : see steps 5–12 of KEYGEN in Figure 2), but applying [38, Theorem 1], the technique is easily seen to work in both ways.

Moreover, if we start from a polynomial of the form  $\zeta \mathbf{F}$ where  $\zeta$  is of the form  $\pm \mathbf{x}^{\alpha}$ , then applying the previous steps yields the quadruple ( $\zeta \mathbf{f}, \zeta \mathbf{g}, \zeta \mathbf{F}, \zeta \mathbf{G}$ ), which is also a valid secret key equivalent to ( $\mathbf{f}, \mathbf{g}, \mathbf{F}, \mathbf{G}$ ), in the sense that signing with either keys produces signatures with exactly the same distributions. Thus, we do not even need to carry out an exhaustive search on several possible values of  $\mathbf{F}$  after the lattice reduction step: it suffices to use the first vector of the reduced basis directly.

#### 4.2 How many faults do we need?

Let us analyze the probability of success of the attack depending on the iteration m at which the iteration is inserted

#### TABLE 1

Experimental success rate of the attack and average CPU time for key recovery for several values of m, the iteration after which the loop-abort fault is injected. We attack the BLISS–II parameter set  $(n, q, \sigma, \delta_1, \delta_2, \kappa) = (512, 12289, 10, 0.3, 0, 23)$  from [20]. Since the choice of  $\ell$  has no effect on the concrete fault injection (e.g. it does not affect the required number of faulty signatures, which is always 1), we did not attempt to optimize it very closely. The simulation was carried out using our Sage implementation on a single core of an Intel Xeon E5-2697v3 workstation, using 100 trial runs for each value of m.

Fault after iteration number $m =$	2	5	10 11	20 22	40 44	60 66	80 88	100 110
	5	0	11	22	11	00	00	110
Dimension $\ell$ in our experiment	3	6	12	24	50	80	110	150
Lattice reduction algorithm	LLL	LLL	LLL	LLL	BKZ-20	BKZ-25	BKZ-25	BKZ-25
Success probability (%)	100	99	100	100	100	100	100	98
Avg. CPU time to recover $\ell$ coeffs. (s)	0.002	0.005	0.022	0.23	7.3	119	941	33655
Avg. CPU time for full key recovery	0.5 s	0.5 s	1 s	5 s	80 s	14 min	80 min	38 h

#### TABLE 2

Experimental success probability of the attack and average CPU time for key recovery for several values of m, the iteration after which the loop-abort fault is injected. We consider the attack with  $\ell = m + 1$  and  $\ell = m + 2$  faulty signatures. The attacked parameters are (n,q) = (256, 1021) as suggested in [23] for signatures. The simulation was carried out using our Sage implementation (see the Appendix ) on a single core of an Intel Xeon E5-2697v3 workstation, using 100 trial runs for each pair  $(\ell, m)$ .

Fault after iteration number $m =$ Lattice reduction algorithm	2	5	10	20	40	60	80	100
	LLL	LLL	LLL	LLL	LLL	LLL	BKZ-20	BKZ–20
Success probability for $\ell = m + 1$ (%)	75	77	90	93	94	94	95	95
Avg. CPU time for $\ell = m + 1$ (s)	0.001	0.003	0.016	0.19	2.1	8.1	21.7	104
Success probability for $\ell = m + 2$ (%)	89	95	100	100	99	99	100	100
Avg. CPU time for $\ell = m + 2$ (s)	0.001	0.003	0.017	0.19	2.1	8.2	21.6	146

and the number  $\ell > m$  of faulty signatures  $\mathbf{z}^{(i)}$  available. As we have seen, a sufficient condition for the attack to succeed (provided that our lattice reduction algorithm actually finds a shortest vector) is that the  $\ell$  faulty signatures generate the rank-m lattice L defined above. This is not actually necessary (the attack works as soon as *one* of the shifts of  $\mathbf{F}$  is in sub-lattice generated by the signatures, rather than all of them), but we will be content with a lower bound on the probability of success.

Now, that condition is equivalent to saying that the vectors  $(r_0^{(i)}, \ldots, r_{m-1}^{(i)}) \in \mathbb{Z}^m$  (sampled according to the distribution given by the GPV algorithm) that define the faulty signatures:

$$\mathbf{z}^{(i)} = r_0^{(i)} \mathbf{x}^{n-1} \mathbf{F} + \dots + r_{m-1}^{(i)} \mathbf{x}^{n-m} \mathbf{F}$$

generate the whole integer lattice  $\mathbb{Z}^m$ . But the probability that  $\ell > m$  random vectors generate  $\mathbb{Z}^m$  has been computed by Maze, Rosenthal and Wagner [47] (see also [28]), and is asymptotically equal to  $\prod_{k=\ell-m+1}^{\ell} \zeta(k)^{-1}$ . In particular, if  $\ell = m + d$  for some integer d, it is bounded below by:

$$p_d = \prod_{k=d+1}^{+\infty} \frac{1}{\zeta(k)}.$$

Thus, if we take  $\ell = m + 1$  (resp.  $\ell = m + 2$ ,  $\ell = m + 3$ ), we expect the attack to succeed with probability at least  $p_1 \approx 43\%$  (resp.  $p_2 \approx 71\%$ ,  $p_3 \approx 86\%$ ).

As shown in Table 2, this is well verified in practice (and the lower bound is in fact quite pessimistic: see the Appendix for an extended discussion). Moreover, the attack is quite fast even for relatively large values of m: only a couple of minutes for full key recovery for m = 100.

#### 5 ATTACK ON NEWHOPE KEY EXCHANGE

We present an attack against the NewHope key exchange protocol and more specifically the generation of the "commitment" elements e, e'. By the inherent symmetry of the protocol, we describe the attack when mounted on Alice's side. The adaptation to Bob's side is done *mutatis mutandis*.

In this scheme, the commitment element consists of a polynomial **e**, which intuitively plays the role of an additive mask to the secret key element **s** in the relation:

$$\mathbf{b} = \mathbf{a} \cdot \mathbf{s} + \mathbf{e}.$$

As a consequence, tampering with the distribution of e should cause some information leakage when sending the element b to Bob.

More formally,  $\mathbf{e} \in \mathcal{R}_q$  is a ring element drawn from the centered binomial distribution  $\psi_{16}$ . Its generation is typically carried out coefficient by coefficient in a polynomial representation. Thus, if one can use faults to cause an early termination of that generation process, we should obtain elements in which the element  $\mathbf{e}$  is actually a low-degree polynomial. If the degree is low enough, we will see that this reveals the whole secret key right away, from a single faulty element: this is quite similar to the attack on BLISS.

Indeed, suppose that we can obtain a faulty element **b** obtained by forcing a termination of the loop for sampling **e** after the *m*-th iteration, with  $m \ll n$ , that is, the resulting polynomial **e** is of degree at most m - 1. As part of the commitment message, we get the pair (*seed*, **b**), and since the generation of **a** is both deterministic and public, the pair (**a**, **b**) with

$$\mathbf{b} = \mathbf{s} \cdot \mathbf{a} + \mathbf{e}.$$

With high probability, **a** is invertible: if we heuristically assume that **a** behaves like a random element of the ring from that standpoint, we expect it to be the case with probability about  $(1 - 1/q)^n$ , which is slightly over 92% for the proposed NewHope parameters. We thus get an equation of the form:

$$\mathbf{a}^{-1}\mathbf{b} \equiv \mathbf{s} + \mathbf{a}^{-1}\mathbf{e} \equiv \mathbf{s} + \underbrace{\sum_{i=0}^{m-1} e_i \mathbf{a}^{-1} \mathbf{x}^i}_{\in \operatorname{Span}_{\mathbb{Z}}((\mathbf{a}^{-1}\mathbf{x}^i)_{0 \le i \le m-1})} \pmod{q} \quad (3)$$

for integers  $e_0, \ldots, e_{m-1}$ . Thus, the vector  $\mathbf{v} = \mathbf{a}^{-1}\mathbf{b}$  is close to the sublattice  $\mathcal{L}$  of  $\mathbb{Z}^n$  generated by the elements  $\mathbf{w}_i = \mathbf{a}^{-1}\mathbf{x}^i \mod q$  for  $i = 0, \ldots, m-1$  and  $q\mathbb{Z}^n$ . Then the difference between  $\mathcal{L}$  and  $\mathbf{v}$  should be precisely s. The rest of the analysis is similar to the attack against BLISS. The main difference lies in the fact that s is sampled according to the *n*-dimensional centered binomial distribution of parameter k = 16 instead of a discrete Gaussian distribution. All in all the computations yield the following condition on  $\ell$ :

$$\ell + 1 \gtrsim \frac{m + 1 + \frac{\log\sqrt{k/2}}{\log q}}{1 - \frac{\log\sqrt{k/2}\sqrt{2\pi e}}{\log q}}.$$
(4)

The denominator of the right-hand side of (4) is roughly about 0.70 for the parameters sets. We thus expect to recover s if we can solve the shortest vector problem in a lattice of dimension slightly larger than  $m/0.7 \approx 1.4 \cdot m$ . This is quite feasible with the LLL algorithm for m up to about 40, and with BKZ for m up to 80 or so.

## 6 ATTACK ON FRODO KEY EXCHANGE

Our second attack targets the Frodo key exchange protocol. See also the Appendix for a discussion of an attack on Kyber [12], which has some similarity to the one below. Like in the previous attack we tamper the generation of the "commitment" matrices  $\mathbf{E}, \mathbf{E}'$ . By the inherent symmetry of the protocol, we describe the attack on Bob's side. The adaptation to Alice's side is straightforward, once one takes into account the simple observation the multiplication by the secret is performed on the other side of the public matrix *A* (left-multiplication for Bob and right-multiplication for Alice).

In this scheme, the commitment matrix consists on a vector  $\mathbf{E}'$ , which acts as an additive mask to the secret key matrix  $\mathbf{S}'$  in the relation:

$$\mathbf{B}' = \mathbf{S}' \cdot \mathbf{A} + \mathbf{E}'.$$

As a consequence, tampering the distribution of  $\mathbf{E}'$  leaks some information when sending the matrix  $\mathbf{B}'$  to Bob.

More formally,  $\mathbf{E}' \in \mathbb{Z}_q^{n \times \bar{n}}$  is a matrix of integers, whose coefficients are independently drawn under a discrete Gaussian distribution. Hence its generation is then once again typically carried out coefficient by coefficient, allowing us to use faults to cause an early abort of this generation process. However, since we are dealing with matrices instead of vectors as in the previous attacks, there is a subtlety in the way the generation can be carried out. The entries of the matrix can be sampled column-by-column or row-by-row, and these two generation algorithms are not equivalent from

the standpoint of our attack. In the former case an early abort on the outermost loop of the generation process leads to a matrix with a certain number of column correctly set and then a family of zero column vectors:

$$\mathbf{E}' = \left[ \begin{array}{c} E_1' & \mathbf{0} \end{array} \right]. \tag{5}$$

In the latter case an early abort makes arise a matrix with full first rows, followed with a certain number of zero rows vectors:

$$\mathbf{E}' = \left\lfloor \frac{E_1'}{\mathbf{0}} \right\rfloor. \tag{6}$$

#### 6.1 Column-wise generation

We first focus on the case where the generation is carried out *column-by-column*, so that the fault attack yields a matrix  $\mathbf{E}'$  of the form (5).

Precisely, let us consider a matrix  $\mathbf{B}'$  obtained by forcing a termination of the loop for sampling  $\mathbf{E}'$  after the *m*-th iteration, with  $m \ll n$  (that is, the resulting matrix  $\mathbf{E}'$ has at most m - 1 non-zero columns, collected in a plain matrix denoted by  $E'_1$ ). Without loss of generality, we can assume the non-zero columns are the first ones. As part of the commitment message, we get the pair (*seed*,  $\mathbf{B}'$ ), and since the generation of  $\mathbf{A}$  is both deterministic and public, we have access to the pair ( $\mathbf{A}, \mathbf{B}'$ ) of integer matrices, satisfying the following relation taken modulo q (notice that the block decomposition is taken to be consistent with the block decomposition of E in  $[E'_1|\mathbf{0}]$ ):

$$\begin{bmatrix} B'_1 \mid B'_2 \end{bmatrix} = \mathbf{B}' = \mathbf{S}' \cdot \mathbf{A} + \mathbf{E}'$$
$$= \begin{bmatrix} S'_1 \mid S'_2 \end{bmatrix} \begin{bmatrix} A_1 \mid A_2 \\ \hline A_3 \mid A_4 \end{bmatrix} + \begin{bmatrix} E'_1 \mid \mathbf{0} \end{bmatrix},$$

yielding the system:

$$\begin{cases} S'_1 \cdot A_1 + S'_2 \cdot A_3 &= B'_1 - E'_1 \\ S'_1 \cdot A_2 + S'_2 \cdot A_4 &= B'_2 \end{cases} \mod q.$$

Let us suppose at that point that  $A_4$  is invertible<sup>4</sup> mod q.

$$\begin{cases} S'_1 \cdot A_1 + S'_2 \cdot A_3 &= B'_1 - E'_1 \\ S'_2 &= (B'_2 - S'_1 \cdot A_2) \cdot A_4^{-1} & \text{mod } q, \end{cases}$$
(7)

yielding by replacement in the first equation:

~ .

$$S'_{1} \cdot (A_{1} - A_{2} \cdot A_{4}^{-1} \cdot A_{3}) + E'_{1} = B'_{1} - B'_{2} \cdot A_{4}^{-1} \cdot A_{3} \mod q$$

This equation is then of shape  $S'_1 \cdot A + E'_1 = B$ . Suppose now that  $\tilde{A} = (A_1 - A_2 \cdot A_4^{-1} \cdot A_3)$  is also invertible. Then performing the inversion trick described in the attack against NewHope yields:

$$B \cdot A^{-1} = S'_1 + E'_1 \cdot A^{-1} \mod q$$

Considering separately each of the n rows yields n equations on vectors:

$$\forall i, \left[\tilde{B}\tilde{A}^{-1}\right]_i = [S_1']_i + \underbrace{\left[E_1' \cdot \tilde{A}^{-1}\right]_i}_{\in \operatorname{Span}_{\mathbb{Z}}((\mathbf{e}_{i,j}\tilde{A}^{-1})_{0 \le j \le m-1})} \mod q$$

4. Seeing this matrix as a random uniform  $m \times m$  matrix over  $\mathbb{F}_q$ , which is invertible with probability  $\prod_{i=1}^{m} (1 - q^{-i})$ .

where  $\mathbf{e}_{i,j}$  is the (i,j) - th elementary matrix. The attack is then mounted to recover each rows, exactly in the same fashion as before, this time considering the *n* lattices of rank m,  $(\mathcal{L}_i)_{0 \le i < m}$ , generated by the  $\mathbf{w}_j^{(i)} = \tilde{A}^{-1}(\mathbf{e}_{i,j})_j \mod q$ for  $j = 0, \ldots, m-1$  and  $q\mathbb{Z}^m$ . Notice that here the recovery can be performed directly by solving the SVP problem in these lattices since their common rank, *m* is supposed to be small. Once  $S'_1$  is recovered, Equation (7), allows to recover directly  $S'_2$  by linear algebra.

#### 6.2 Row-wise generation

We now turn to the case where the generation is conducted *row-by-row*, so that  $\mathbf{E}'$  is of the form (6).

More precisely, consider a matrix  $\mathbf{B}'$  obtained by forcing a loop abort during the sampling  $\mathbf{E}'$  after the *m*-h iteration, with  $m \ll n$  (that is, the resulting matrix  $\mathbf{E}'$  has at most m - 1 non-zero rows, collected in a full matrix denoted by  $E'_1$ ). Without loss of generality, we can assume the nonzero columns are the first ones. Once again, we get access to the pair  $(\mathbf{A}, \mathbf{B}')$  of integer matrices, satisfying the following relation taken modulo *q*:

$$\begin{bmatrix} \underline{B'_1} \\ \overline{B'_2} \end{bmatrix} = \mathbf{B'} = \mathbf{S'} \cdot \mathbf{A} + \mathbf{E'}$$

$$= \begin{bmatrix} \underline{S'_1} & \underline{S'_2} \\ \overline{S'_3} & \overline{S'_4} \end{bmatrix} \begin{bmatrix} \underline{A_1} \\ \overline{A_2} \end{bmatrix} + \begin{bmatrix} \underline{E'_1} \\ \mathbf{0} \end{bmatrix}.$$

In this case, no factorization similar to the one appearing in the previous section allows to reduce the problem to a smaller instance. As a result, in this setting, it does not seem possible to mount the loop-abort fault attack on Bob's side!

However, note that in this case, the scheme is vulnerable on *Alice's side*. Indeed, taking the same notations one have, by tampering the generation of  $\mathbf{E}$  to get the following equation once ( $\mathbf{A}$ ,  $\mathbf{B}$ ) are obtained:

$$\begin{bmatrix} B_1 \\ B_2 \end{bmatrix} = \mathbf{B} = \mathbf{A} \cdot \mathbf{S} + \mathbf{E}$$
$$= \begin{bmatrix} A_1 & A_2 \\ A_3 & A_4 \end{bmatrix} \begin{bmatrix} S_1 \\ S_2 \end{bmatrix} + \begin{bmatrix} E_1 \\ \mathbf{0} \end{bmatrix}$$

yielding the system:

$$\begin{cases} A_1 \cdot S_1 + A_2 \cdot S_2 &= B_1 - E_1 \\ A_3 \cdot S_1 + A_4 \cdot S_2 &= B_2 \end{cases} \mod q,$$

which is of the same shape of Equation (7). The discussion of Paragraph 6.1 can then be directly adapted to recover  $S_1$  and  $S_2$ .

# 7 FEASIBILITY OF THE FAULTS

In this section, we investigate how an attacker may obtain helpful faulty values for the proposed attacks. We base our discussion on two available implementations of BLISS signature, namely the software implementation of Ducas and Lepoint [21] and the FPGA implementation by Pöppelmann *et al.* [62], and on Prest's software implementation of the GPV-based scheme of Ducas et al. [63]. The discussion extends to the other signature schemes and to the key exchange protocols directly, since the type of faulty behavior we want to cause (loop aborts) is the same in all cases.

We emphasize that those three implementations were not supposed to have any resilience with respect to fault attacks and were only developed as proofs of concept to illustrate the efficiency properties of the schemes. The point here is to show that the fault attacks presented in this paper are relevant based on the analysis of freely available and published implementations to put forward the need for dedicated protections against faults attacks (when attackers have such abilities).

#### 7.1 Classical fault models

Faults during a computation may be induced by different means as a laser beam shot, electromagnetic injection, under-powering, glitches, etc. These faults are mainly characterized by their

- range: impacting a single bit or many bits (e.g. register or memory word);
- effect: typically target chunk is set to a chosen value, random value or all-zero/all-one value;
- persistence: a fault may only modify the target for a short period or it may be definitive.

Obviously, some fault models are close to being purely theoretical: it is very unlikely to be able to set a 32-bit register to 0xbad00dad during precisely 2 cycles. Nevertheless, many recent works have been published showing that some faults models that seemed overly ambitious are actually achievable during lab experiments with good reproducibility. That is true for single-bit faults both using EM [54] or laser injections [13]. There are also many reported instruction-skipping faults in the literature. One example of such faults is the work of Rivière et al. on ARMv7 instruction cache [64].

In the next subsections, we discuss which fault models<sup>5</sup> may lead to faulty signatures that are relevant with respect to the attacks presented in this paper.

#### 7.2 Fault attacks on software implementations

In BLISS, the polynomial  $y_1$  can be generated using a loop over the *n* coefficients. This is indeed how the implementation in [21] does it: a loop constructs the polynomials  $y_1$ and  $y_2$  coefficient by coefficient using a Gaussian sampler (function Sign::signMessage). The condition to perform the attack successfully is not very demanding since we only try to ensure that three quarters or so of the coefficients of  $y_1$  are fixed to zero. Such a result can easily be obtained by exiting the loop after a few iterations. A random fault on the loop counter or skipping the jump operation will lead to such a result.

Notice here that it is not trivial to decide whether a faulty signature will be helpful or not. Hopefully, the required timing precision is much less important in this setting since the attack will succeed even with 50 unknown coefficients out of 512. This means that the time window for the fault to occur is composed of dozens of loop iterations. Moreover, we may use side-channel analysis to detect the loop iteration

5. We only focus on single fault attacks here.

pattern to trigger the fault injection. That pattern is likely to be detected after much fewer than 50 iterations, and thus it seems that the synchronization here will be quite easy.

Similarly, the short random polynomials  $\mathbf{R}$  and  $\mathbf{r}$  used in the GPV scheme are generated in a single loop [63] ranging from leading coefficient of  $\mathbf{r}$  to the constant term in  $\mathbf{R}$  which allows to fault both polynomials using a single fault. Again, a random fault on the counter or skipping a jump makes it work and the time window is large since, according to the results described in Table 2, if one collects say 50 faulty signatures, one can tolerate close to 50 executed iterations each time.

To conclude, these attacks seem to be a real threat since synchronization (which is a major difficulty when performing fault attacks in general) is made easy by the loose condition on the number of known coefficients in the faulty polynomials.

#### 7.3 Fault attacks on hardware implementations

In BLISS, the generation of the polynomial  $y_1$  requires sampling n random coefficients, where n = 512 in the most common parameter settings. Even in hardware, it is highly unlikely that all these coefficients are obtained at the same time (n is simply too large); thus, the generation of  $y_1$  is most likely to be sequential. This is the case in the implementation we took as example where the super memory is linked to the sampler through a 14-bit port. We may fault a flag or a state register to fool the control logic (in this case, the BLISS coprocessor) and keep part of the BRAM cells to their initial state. If this initial state is known then we know all the corresponding coefficients and hopefully, the number of unknown ones will be small enough for the attack to work. Again, the large number of unknown coefficients supported by the attack helps the adversary by providing a large time window for the fault to occur. The feasibility of the attack will mostly depend on the precise flag/state implementation and the knowledge of the previous/initial value of memory cells.

There is a second way of performing the fault injection here. The value of  $y_1$  has to be stored somehow until the computation of  $z_1$  (close to the end of the signature generation). In the example implementation, a BRAM is used. We may fault BRAM accesses to fix some coefficients to a known value. A possible fault would be to set the rstram or rstreg signal to one (Xilinx's nomenclature). Indeed, when set to one, this will set the output latches (*resp.* register) of the RAM block to some fixed value SRVAL defined by the designer. We may notice two points to understand why this kind of fault enables the proposed attack.

- (i) The value y<sub>1</sub> used to compute u will not be the faulted one but this has no impact on the attack.
- (ii) If we do not know the default value for the output register, all coefficients are unknown but a big part of them are equal to the same unknown default value. In that case, the attack is still applicable by adding one generator to the constructed lattice: see Remark 3 in §3.

Again a large time window is given to the attacker due to sequential read induced by the size of  $y_1$ .

The BRAM storage of  $y_1$  helps the attacker since a single bit-set fault may have effects on many coefficients. The only difficulty seems to be able to perform a single-bit fault which is a real threat as stated in §7.1—as well as the localization of the rstram signal<sup>6</sup>.

## 8 IMPLEMENTATION OF THE FAULTS

In this section, we further validate the proposed attack model in software against two different platforms. We consider the specific case of NewHope, and compile a lightweight implementation of it for the SPARC-based 32-bit LEON3 processor, as well as for the 8-bit Atmel XMEGA128 microcontroller.

In the case of the LEON3 code, we carry out a simulation of our loop abort fault using a tool [56] based on the tsim LEON3 emulator. And in the case of the XMEGA code, we actually mount the attack in practice using the ChipWhisperer-Lite side-channel and fault analysis evaluation board [53]; we use clock glitches to inject the fault, and successfully manage to cause early loop aborts.

#### 8.1 Simulated faults on LEON3

To confirm that an instruction-skipping fault actually leads to a full secret recovery, we have first simulated fault attacks on NewHope (both on s and e) using the tool made available by authors of [56]. They propose a python script that makes calls to the Aeroflex Gaisler's LEON3 CPU simulator (namely tsim) to simulate the replacement of one or more instruction by a nop (or to modify some data but it is out of our interest here). The LEON3 is a 32-bit processor (SPARC architecture) with freely available source code. Thus, it can be used as a soft-core (i.e. a processor instantiated on an FPGA) or directly integrated as an IP for an ASIC. This target is the only one for which a fault simulator is available thus it is a perfect candidate to simulate the proposed attack.

For both polynomials e and s, the generation from the "small" (centered binomial) distribution in NewHope is performed by first generating some pseudorandom bits using Chacha20 then processing each resulting 32-bit word to convert it into a binomially distributed value (function poly\_get\_noise). Our fault injection simulation targets this loop, in particular, attempting to cause an early abort in its execution.

More precisely, sufficiently many random bytes are first obtained and stored in a 32-bit word buffer. They are then processed to derive polynomial coefficients which are stored in a 16-bit word array. We first directly targeted the second loop (where coefficients are derived from random words) and managed to obtain two relevant faults in a few hours of simulation. This was mainly due to the fact that the granularity of the simulator is a range of address to fault. Due to the binary code structure, a jumping instruction was present in this range leading to a huge amount of executed code between the first and the last potentially

<sup>6.</sup> Since  $\mathbf{y}_1$  is not output directly, checking if the attack actually worked can be a bit tricky. Again side-channel collision analysis may help here. We may also notice that if the faulty  $\mathbf{y}_1$  is sparse (that is known coefficients have been set to zero) then the number of non-zero coefficients in the corresponding  $\mathbf{z}_1$  should be significantly smaller than for a  $\mathbf{z}_1$  corresponding to a dense  $\mathbf{y}_1$ .

Fig. 5. The ChipWhisperer-Lite evaluation board, with XMEGA microcontroller target attached.



faulted instruction. This resulted in a large simulation time. Due to time limitations, we did not try to modify the script from [56] to take this particular situation into account. Indeed, two faulty outputs corresponding to loop aborts in the first and second iterations were observed and allowed key-recovery and this was enough to confirm that the attack was an actual threat when instruction-skip faults can be injected on the device.

Nevertheless, and for the sake of completeness, the code has been modified to first derive coefficients in place and then to copy coefficients in the polynomial structure. This latter copy has been moved into a dedicated function to narrow the step-by-step simulation region. Using this code modification faults have been obtained for all abort iteration indexes. These faulty outputs have then been used to feed a sage attack script recovering the key (when the number of iterations is small enough).

This confirms that skipping an instruction will lead to the expected behavior according to the widely deployed simulator for the LEON3 core. In particular, this shows that there is no side-effect of the fault nor of the targeted code that would render the proposed attack more complicated or inefficient than expected.

#### 8.2 Concrete faults on XMEGA with ChipWhisperer

Finally, we concretely mount our fault attack against the same software implementation of NewHope, this time compiled for the Atmel XMEGA128D4 8-bit microcontroller coming with the ChipWhisperer-Lite evaluation board [53]. The ChipWhisperer-Lite (Fig. 5) enables the injection of clock and voltage glitches during the execution of the compiled code on the microcontroller. We successfully use clock glitches to cause an early loop abort in the generation of the polynomial e, allowing a full secret key recovery with the attack of §5.

While the XMEGA128D4 has a substantial amount of flash memory to store the compiled code, it has only 8 KB of RAM, which makes it impossible to run the entire NewHope code on the microcontroller. As a result, we have to do some offline precomputations on the host computer and store the result in the compiled binary before the experiment. However, the relevant part of the code as far as our attack is concerned, namely the loop generating the polynomial e from pseudorandom bits generated in advance (or the copy moving the result to the target area in memory), can be executed in full, and we can carry out our clock glitches against it. In addition, we note that the XMEGA executes the more time-consuming operations in NewHope like the direct and inverse number-theoretic transforms without problem.

Another way in which we "cheat" slightly in our fault attack is by raising a hardware trigger in the code before the execution of the loop of interest. Using more advanced tooling than the ChipWhisperer-Lite, however, it would be easy to replace that manually inserted trigger by a trigger on the waveform of the power trace before entering the loop. Such a triggering mechanism is for example available in the ChipWhisperer-Pro (which can detect patterns in power traces using SAD) and most higher-end oscilloscopes.

Apart from those slight changes to accommodate for our very lightweight target, we are able to mount the glitch attack on an almost unmodified implementation of NewHope. It takes some time to select proper parameters for the clock glitches, so as to obtain reliably reproducible faults, but the "glitch explorer" included in the ChipWhisperer Capture software makes this a relatively painless exercise. In the end, we ended up selecting one-shot clock glitches of width 2.5% of a clock period and offset -5.5% of a period: in other words, at the clock cycle where the fault is injected, the square signal providing the clock input for the microcontroller is modified by XORing it with a short rectangular pulse of the same amplitude as the original signal, slightly before the normal raising edge. This reliably caused a perturbation in the execution of the code on the microcontroller, which can be observed on power traces: see Fig. 6 (the effect is especially visible when we repeat the glitch over multiple cycles, as in Fig. 6(c), but such a long glitch causes the device to reset, and hence is not useful after parameter selection is complete). Note that although it took us some time to choose those parameters, reliable faults can be obtained from a relatively large range of parameters: see the short discussion in the Appendix and its accompanying figure.

When inserting the glitch at a suitable offset from the trigger, we were able to cause an early loop abort after any chosen number of iterations. More precisely, on the compiled code used in our experiments, inserting the glitch at offset 12k + 9 clock cycles after the trigger is raised<sup>7</sup>, we obtained an exit of the loop after exactly k iterations, and hence a polynomial e of degree exactly k - 1 allowing to recover the entire secrets for all small values of k. This result is highly reproducible in our setup: among 100 injected faults at various offsets, we obtained 97 successful loop aborts after the expected number of iterations.

#### **9 POSSIBLE COUNTERMEASURES**

We have shown that unprotected implementations of latticebased signature and key exchange schemes are vulnerable to fault attacks, in fault models that our analysis suggests are quite realistic: the faulty signatures required by our attacks can be obtained on actual implementations. As a result,

<sup>7.</sup> Like the parameters for the glitches, this offset pattern was also found by trial and error using the glitch explorer in ChipWhisperer Capture.

Fig. 6. Power traces of the target loop in NewHope on the XMEGA microcontroller, with and without faults. Sampling rate is  $4\times$  the clock frequency.



(c) Trace with 20 glitched cycles from offset 33

countermeasures should be added in applications where such a physical attacker is relevant to the threat model.

Simple countermeasures exist to thwart the single fault attacks proposed. There are simple, non-cryptographic countermeasures that consist in validating that the full loop has been correctly performed. This can be achieved for instance by adding a second loop counter and doing a consistency check after exiting the loop. Such a countermeasure is very cheap and we therefore recommend introducing it in all deployed implementations.

Nevertheless, it will only detect early-abort faults while an attacker may succeed in getting the same kind of faulty signature using another technique. For instance, we mentioned the possibility of faulting BRAM blocks so that they output a fixed value. For software implementations, the compiler may decide to put the coefficient in some RAM location which address could be faulted to point to another part of the memory leading in many coefficients having the same value. A single fault may also alter instruction cache leading to a nop operation instead of a load from memory and thus not updating the coefficient. We propose now other countermeasures that may deal with this issue for both types of signature schemes we considered.

We have described our attack on the Fiat–Shamir schemes in a setting where the attacker can obtain a commitment polynomial **y** of low degree and it works more generally with a sparse **y**, provided that the attackers *knows* 

where the non zero coefficients are located. If the locations are unknown, however, the attack does not work, so one possible countermeasure is to randomize the order of the loop generating y. One should be careful that this may not protect against faults introduced after the very first few iterations, however: in the case of BLISS, for example, we have seen that we could easily attack polynomials y in which the non zero coefficients are located in the 20% lower degree coefficients, say; then, if a fault attacker can collect a few hundred faulty signatures with y of very low Hamming weight (say 3 or 4) at random positions, they have a good chance of finding one fault with all non zero coefficients in the lower 20%, and hence be able to attack. The same observation applies to the key exchange protocols.

Another possible approach for the Fiat–Shamir schemes is to check that the degree of the generated y is not too low. One cannot demand that all its coefficients are non zero, as this would skew the distribution and invalidate the security argument, but verifying that the top  $\varepsilon \cdot n$  coefficients of y are not all zero for some small constant  $\varepsilon > 0$ , say  $\varepsilon = 1/16$ , would be a practical countermeasure that does not affect the security proof. Indeed, in the case of BLISS, for example, the probability that all of these coefficients vanish is roughly  $(1/\sigma\sqrt{2\pi})^{\varepsilon n}$ , which is exponentially small. Thus, the resulting distribution of y after this check is statistically indistinguishable from the original distribution and security is therefore preserved. Moreover, the lattice dimension required to mount our fault attack is then greater than  $(1 - \varepsilon)n$ , so it will not work. An additional advantage of that countermeasure is that it also adapts easily to thwart faults that cause all the top coefficients of y to be equal to some constant non-zero value. Again, this method is also applicable to NewHope (where the larger size of the coefficient vectors ensures that a smaller  $\varepsilon \approx 0.3$  suffices to achieve a statistical distance below  $2^{-128}$ ). It can also be adapted relatively easily to Frodo and Kyber as well (in fact, for Frodo, one could even require all the rows and columns of the noise matrices to be non-zero for maximum safety, although the countermeasure then becomes somewhat costly).

Regarding the hash-and-sign signature of Ducas et al., one possible countermeasure is to simply check the validity of generated signatures. This will usually work due to the fact that a faulty signature generated from an early loop abort from the GAUSSIANSAMPLER algorithm is of significantly larger norm than a valid signature: a rough estimate of the norm after  $m \leq n$  iterations is  $\|\mathbf{F}\|_2 \sqrt{mq/12}$  (as q/12 is the variance of a uniform random variable in  $\{-(q-1)/2, \ldots, (q-1)/2\}$ ), which is too large for correct verification even for very small values of m. An added benefit of that countermeasure is that even the correct signature generation algorithm has a very small but non zero probability of generating an invalid signature, so this countermeasure doubles up as a safeguard against those rare accidental failures.

## **10** CONCLUSION AND OUTLOOK

The main takeaway of our results is that physical attacks, and particularly faults, can be a devastating threat against lattice-based schemes. Extensive research has been conducted on fault attacks against RSA and discrete logarithmbased schemes, so that both threats and countermeasures are well-understood. As lattice-based cryptography advances closer to real-world deployment, similar efforts need to be devoted to their fault analysis.

This paper considered one class of attacks, loop-abort faults, that has wide-ranging applicability to lattice-based signature schemes and key exchange protocols. It is also a relatively cheap attack to protect against, so introducing countermeasures in actual implementations is strongly recommended.

Future work could extend our results both by widening the scope of possible schemes vulnerable to that class of attacks (e.g. loop-abort faults have also recently been shown to be a threat to isogeny-based cryptography [31]), and by considering other, possibly more advanced fault attacks on the same schemes. Another possible avenue for further research could be the provable security of certain fault countermeasures on lattice-based (such as the ones discussed above) within a sufficiently restricted fault model. Results of this type have been obtained before on RSA [6], [15], for example, although it is usually not hard to bypass the limits of the corresponding fault model [29].

#### Acknowledgments

We acknowledge the support of the French Programme d'Investissement d'Avenir under national project RISQ. This work is also partially supported by the European Union projects PROMETHEUS (Horizon 2020 Research and Innovation Program, grant 780701) and HEAT (Horizon 2020 Research and Innovation Program, grant 644209).

## REFERENCES

- S. Akleylek, N. Bindel, J. A. Buchmann, J. Krämer, and G. A. Marson. An efficient lattice-based signature scheme with provably secure instantiation. In D. Pointcheval, A. Nitaj, and T. Rachidi, editors, *AFRICACRYPT*, volume 9646 of *LNCS*, pages 44–60. Springer, 2016.
- [2] E. Alkim, N. Bindel, J. A. Buchmann, Ö. Dagdelen, E. Eaton, G. Gutoski, J. Krämer, and F. Pawlega. Revisiting TESLA in the quantum random oracle model. In T. Lange and T. Takagi, editors, *PQCrypto*, volume 10346 of *LNCS*, pages 143–162. Springer, 2017.
- [3] E. Alkim, L. Ducas, T. Pöppelmann, and P. Schwabe. Postquantum key exchange - A new hope. In T. Holz and S. Savage, editors, USENIX Security Symposium, pages 327–343. USENIX Association, 2016.
- [4] L. Babai. On Lovász' lattice reduction and the nearest lattice point problem. *Combinatorica*, 6(1):1–13, 1986.
  [5] S. Bai and S. D. Galbraith. An improved compression technique
- [5] S. Bai and S. D. Galbraith. An improved compression technique for signatures based on learning with errors. In J. Benaloh, editor, *CT-RSA*, volume 8366 of *LNCS*, pages 28–47. Springer, 2014.
- [6] G. Barthe, F. Dupressoir, P. Fouque, B. Grégoire, M. Tibouchi, and J. Zapalowicz. Making RSA-PSS provably secure against nonrandom faults. In L. Batina and M. Robshaw, editors, *CHES*, volume 8731 of *LNCS*, pages 206–222. Springer, 2014.
- [7] I. Biehl, B. Meyer, and V. Müller. Differential fault attacks on elliptic curve cryptosystems. In M. Bellare, editor, CRYPTO, volume 1880 of LNCS, pages 131–146. Springer, 2000.
- [8] N. Bindel, J. A. Buchmann, and J. Krämer. Lattice-based signature schemes and their sensitivity to fault attacks. In P. Maurine and M. Tunstall, editors, *FDTC*, pages 63–77. IEEE Computer Society, 2016.
- [9] D. Boneh, R. A. DeMillo, and R. J. Lipton. On the importance of eliminating errors in cryptographic computations. J. Cryptology, 14(2):101–119, 2001.

- [10] J. W. Bos, C. Costello, L. Ducas, I. Mironov, M. Naehrig, V. Nikolaenko, A. Raghunathan, and D. Stebila. Frodo: Take off the ring! Practical, quantum-secure key exchange from LWE. In E. R. Weippl, S. Katzenbeisser, C. Kruegel, A. C. Myers, and S. Halevi, editors, ACM CCS, pages 1006–1018. ACM, 2016.
- [11] J. W. Bos, C. Costello, M. Naehrig, and D. Stebila. Post-quantum key exchange for the TLS protocol from the ring learning with errors problem. In L. Bauer and V. Shmatikov, editors, *IEEE S&P*, pages 553–570. IEEE Computer Society, 2015.
- [12] J. W. Bos, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, J. M. Schanck, P. Schwabe, and D. Stehlé. CRYSTALS Kyber: a CCA-secure module-lattice-based KEM. In F. Piessens and M. Smith, editors, *EuroS&P*. IEEE Computer Society, 2018.
- [13] C. Champeix, N. Borrel, J. Dutertre, B. Robisson, M. Lisart, and A. Sarafianos. SEU sensitivity and modeling using pico-second pulsed laser stimulation of a D flip-flop in 40 nm CMOS technology. In O. Khan and M. K. Michael, editors, *DFT*, pages 177–182. IEEE Computer Society, 2015.
- [14] L. Chen, S. Jordan, Y.-K. Liu, D. Moody, R. Peralta, R. Perlner, and D. Smith-Tone. Report on post-quantum cryptography. Technical report, National Institute of Standards and Technology, Feb. 2016. Available at http://csrc.nist.gov/publications/drafts/nistir-8105/nistir\_8105\_draft.pdf.
- [15] J. Coron and A. Mandal. PSS is secure against random fault attacks. In M. Matsui, editor, ASIACRYPT, volume 5912 of LNCS, pages 653–666. Springer, 2009.
- [16] Ö. Dagdelen, R. E. Bansarkhani, F. Göpfert, T. Güneysu, T. Oder, T. Pöppelmann, A. H. Sánchez, and P. Schwabe. High-speed signatures from standard lattices. In D. F. Aranha and A. Menezes, editors, *LATINCRYPT*, volume 8895 of *LNCS*, pages 84–103. Springer, 2015.
- [17] V. S. Denchev, S. Boixo, S. V. Isakov, N. Ding, R. Babbush, V. Smelyanskiy, J. Martinis, and H. Neven. What is the Computational Value of Finite Range Tunneling? *ArXiv e-prints*, Dec. 2015.
- [18] J. Ding, X. Xie, and X. Lin. A simple provably secure key exchange scheme based on the learning with errors problem. *IACR Cryptology ePrint Archive*, 2012:688, 2012.
- [19] L. Ducas. Accelerating BLISS: the geometry of ternary polynomials. IACR Cryptology ePrint Archive, 2014:874, 2014.
- [20] L. Ducas, A. Durmus, T. Lepoint, and V. Lyubashevsky. Lattice signatures and bimodal gaussians. In R. Canetti and J. A. Garay, editors, *CRYPTO*, volume 8042 of *LNCS*, pages 40–56. Springer, 2013.
- [21] L. Ducas and T. Lepoint. A proof-of-concept implementation of BLISS. Available at http://bliss.di.ens.fr.
- [22] L. Ducas, T. Lepoint, V. Lyubashevsky, P. Schwabe, G. Seiler, and D. Stehlé. CRYSTALS - Dilithium: Digital signatures from module lattices. *IACR Cryptology ePrint Archive*, 2017:633, 2017.
- [23] L. Ducas, V. Lyubashevsky, and T. Prest. Efficient identity-based encryption over NTRU lattices. In P. Sarkar and T. Iwata, editors, *ASIACRYPT*, volume 8874 of *LNCS*, pages 22–41. Springer, 2014.
- [24] L. Ducas and P. Q. Nguyen. Learning a zonotope and more: Cryptanalysis of NTRUSign countermeasures. In X. Wang and K. Sako, editors, ASIACRYPT, volume 7658 of LNCS, pages 433– 450. Springer, 2012.
- [25] T. Espitau, P. Fouque, B. Gérard, and M. Tibouchi. Loop-abort faults on lattice-based Fiat–Shamir and hash-and-sign signatures. In R. Avanzi and H. M. Heys, editors, SAC, volume 10532 of LNCS, pages 140–158. Springer, 2016.
- [26] T. Espitau, P. Fouque, B. Gérard, and M. Tibouchi. Side-channel attacks on BLISS lattice-based signatures. In B. M. Thuraisingham, D. Evans, T. Malkin, and D. Xu, editors, ACM CCS, pages 1857– 1874. ACM, 2017.
- [27] A. Fiat and A. Shamir. How to prove yourself: Practical solutions to identification and signature problems. In A. M. Odlyzko, editor, *CRYPTO*, volume 263 of *LNCS*, pages 186–194. Springer, 1986.
- [28] F. Fontein and P. Wocjan. On the probability of generating a lattice. *Journal of Symbolic Computation*, 64:3–15, 2014.
- [29] P. Fouque, N. Guillermin, D. Leresteux, M. Tibouchi, and J. Zapalowicz. Attacking RSA-CRT signatures with faults on Montgomery multiplication. J. Cryptographic Engineering, 3(1):59–72, 2013.
- [30] P.-A. Fouque, J. Hoffstein, P. Kirchner, V. Lyubashevsky, T. Pornin, T. Prest, T. Ricosset, G. Seiler, W. Whyte, and Z. Zhang. Falcon: Fast-Fourier lattice-based compact signatures over NTRU. Speci-

fications v1.0, 2017. Available at https://falcon-sign.info/falcon.pdf.

- [31] A. Gélin and B. Wesolowski. Loop-abort faults on supersingular isogeny cryptosystems. In T. Lange and T. Takagi, editors, *PQCrypto*, volume 10346 of *LNCS*, pages 93–106. Springer, 2017.
- [32] C. Gentry, J. Jonsson, J. Stern, and M. Szydlo. Cryptanalysis of the NTRU signature scheme (NSS) from Eurocrypt 2001. In C. Boyd, editor, ASIACRYPT, volume 2248 of LNCS, pages 1–20. Springer, 2001.
- [33] C. Gentry, C. Peikert, and V. Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In C. Dwork, editor, *STOC*, pages 197–206. ACM, 2008.
- [34] C. Gentry and M. Szydlo. Cryptanalysis of the revised NTRU signature scheme. In L. R. Knudsen, editor, EUROCRYPT, volume 2332 of LNCS, pages 299–320. Springer, 2002.
- [35] O. Goldreich, S. Goldwasser, and S. Halevi. Public-key cryptosystems from lattice reduction problems. In B. S. Kaliski, Jr., editor, *CRYPTO*, volume 1294 of *LNCS*, pages 112–131. Springer, 1997.
- [36] L. Groot Bruinderink, A. Hülsing, T. Lange, and Y. Yarom. Flush, Gauss, and reload - A cache attack on the BLISS lattice-based signature scheme. In B. Gierlichs and A. Y. Poschmann, editors, *CHES*, volume 9813 of *LNCS*, pages 323–345. Springer, 2016.
- [37] T. Güneysu, V. Lyubashevsky, and T. Pöppelmann. Practical lattice-based cryptography: A signature scheme for embedded systems. In E. Prouff and P. Schaumont, editors, CHES, volume 7428 of LNCS, pages 530–547. Springer, 2012.
- [38] J. Hoffstein, N. Howgrave-Graham, J. Pipher, J. H. Silverman, and W. Whyte. NTRUSign: Digital signatures using the NTRU lattice. In M. Joye, editor, *CT-RSA*, volume 2612 of *LNCS*, pages 122–140. Springer, 2003.
- [39] J. Hoffstein, J. Pipher, J. M. Schanck, J. H. Silverman, and W. Whyte. Practical signatures from the partial Fourier recovery problem. In I. Boureanu, P. Owesarski, and S. Vaudenay, editors, ACNS, volume 8479 of LNCS, pages 476–493. Springer, 2014.
- [40] J. Howe, T. Pöppelmann, M. O'Neill, E. O'Sullivan, and T. Güneysu. Practical lattice-based digital signature schemes. ACM Trans. Embedded Comput. Syst., 14(3):41, 2015.
- [41] A. A. Kamal and A. M. Youssef. Fault analysis of the NTRUSign digital signature scheme. *Cryptography and Communications*, 4(2):131–144, 2012.
- [42] V. Lyubashevsky. Fiat–Shamir with aborts: Applications to lattice and factoring-based signatures. In M. Matsui, editor, ASIACRYPT, volume 5912 of LNCS, pages 598–616. Springer, 2009.
- [43] V. Lyubashevsky. Lattice signatures without trapdoors. In D. Pointcheval and T. Johansson, editors, *EUROCRYPT*, volume 7237 of *LNCS*, pages 738–755. Springer, 2012.
- [44] V. Lyubashevsky and D. Micciancio. Generalized compact knapsacks are collision resistant. In M. Bugliesi, B. Preneel, V. Sassone, and I. Wegener, editors, *ICALP*, volume 4052 of *LNCS*, pages 144– 155. Springer, 2006.
- [45] V. Lyubashevsky and D. Micciancio. Asymptotically efficient lattice-based digital signatures. In R. Canetti, editor, TCC, volume 4948 of LNCS, pages 37–54. Springer, 2008.
- [46] V. Lyubashevsky, C. Peikert, and O. Regev. On ideal lattices and learning with errors over rings. J. ACM, 60(6):43, 2013.
- [47] G. Maze, J. Rosenthal, and U. Wagner. Natural density of rectangular unimodular integer matrices. *Linear Algebra and its Applications*, 434(5):1319–1324, 2011.
- [48] C. A. Melchor, X. Boyen, J. Deneuville, and P. Gaborit. Sealing the leak on classical NTRU signatures. In M. Mosca, editor, *PQCrypto*, volume 8772 of *LNCS*, pages 1–21. Springer, 2014.
- [49] D. Micciancio and C. Peikert. Trapdoors for lattices: Simpler, tighter, faster, smaller. In D. Pointcheval and T. Johansson, editors, *EUROCRYPT*, volume 7237 of *LNCS*, pages 700–718. Springer, 2012.
- [50] D. Naccache, P. Q. Nguyen, M. Tunstall, and C. Whelan. Experimenting with faults, lattices and the DSA. In S. Vaudenay, editor, *PKC*, volume 3386 of *LNCS*, pages 16–28. Springer, 2005.
- [51] National Security Agency. Commercial national security algorithm suite and quantum computing FAQ. Technical report, Jan. 2016. Available at https://www.iad.gov/iad/library/iaguidance/ia-solutions-for-classified/algorithm-guidance/cnsasuite-and-quantum-computing-faq.cfm.
- [52] P. Q. Nguyen and O. Regev. Learning a parallelepiped: Cryptanalysis of GGH and NTRU signatures. J. Cryptology, 22(2):139–160, 2009.

- [53] C. O'Flynn and Z. D. Chen. ChipWhisperer: An open-source platform for hardware embedded security research. In E. Prouff, editor, COSADE, volume 8622 of LNCS, pages 243–260. Springer, 2014.
- [54] S. Ordas, L. Guillaume-Sage, K. Tobich, J. Dutertre, and P. Maurine. Evidence of a larger EM-induced fault model. In M. Joye and A. Moradi, editors, *CARDIS*, volume 8968 of *LNCS*, pages 245–259. Springer, 2014.
- [55] D. Page and F. Vercauteren. A fault attack on pairing-based cryptography. *IEEE Trans. Computers*, 55(9):1075–1080, 2006.
- [56] C. Patrick, B. Yuce, N. F. Ghalaty, and P. Schaumont. Lightweight fault attack resistance in software using intra-instruction redundancy. In R. Avanzi and H. M. Heys, editors, SAC, volume 10532 of LNCS, pages 231–244. Springer, 2016.
- [57] C. Peikert. Lattice cryptography for the internet. In M. Mosca, editor, PQCrypto 2014, volume 8772 of LNCS, pages 197–219. Springer, 2014.
- [58] C. Peikert. A decade of lattice cryptography. Cryptology ePrint Archive, Report 2015/939, 2015. http://eprint.iacr.org/.
- [59] C. Peikert and A. Rosen. Efficient collision-resistant hashing from worst-case assumptions on cyclic lattices. In S. Halevi and T. Rabin, editors, *TCC*, volume 3876 of *LNCS*, pages 145–166. Springer, 2006.
- [60] P. Pessl, L. Groot Bruinderink, and Y. Yarom. To BLISS-B or not to be: Attacking strongSwan's implementation of post-quantum signatures. In B. M. Thuraisingham, D. Evans, T. Malkin, and D. Xu, editors, ACM CCS, pages 1843–1855. ACM, 2017.
- [61] D. Pointcheval and J. Stern. Security proofs for signature schemes. In U. M. Maurer, editor, EUROCRYPT, volume 1070 of LNCS, pages 387–398. Springer, 1996.
- [62] T. Pöppelmann, L. Ducas, and T. Güneysu. Enhanced latticebased signatures on reconfigurable hardware. In L. Batina and M. Robshaw, editors, *CHES*, volume 8731 of *LNCS*, pages 353–370. Springer, 2014.
- [63] T. Prest. Implementation of the GPV-based scheme of Ducas et al. Available at https://github.com/tprest/Lattice-IBE.
- [64] L. Rivière, Z. Najm, P. Rauzy, J. Danger, J. Bringer, and L. Sauvage. High precision fault injections on the instruction cache of ARMv7-M architectures. In W. H. Robinson, M. Potkonjak, and S. Bhunia, editors, HOST, pages 62–67. IEEE Computer Society, 2015.
- [65] W. Stein et al. Sage Mathematics Software (Version 7.0), 2016. Available at http://www.sagemath.org.
- [66] M. Taha and T. Eisenbarth. Implementation attacks on postquantum cryptographic schemes. In E. A. Aleisa, editor, *ICACC*. IEEE Social Implications of Technology Society, 2015.



Thomas Espitau is an alumnus of École normale supérieure de Cachan (Cachan, France) and is currently a second-year Ph.D. student in computer science at UPMC under the joint supervision of Antoine Joux and Pierre-Alain Fouque. One of his main research topics is lattice-based cryptography and the analysis of lattice reduction algorithms.



**Pierre-Alain Fouque** obtained his Ph.D. in computer science from University Paris VII under Jacques Stern in 2001. He then served as Assistant Professor in the computer science department of École normale supérieure (Paris, France) from 2003 before moving to the University of Rennes I (Rennes, France) as in 2012. He is now Full Professor of computer science at Rennes I and the Institut universitaire de France. His research interests cover all aspects of cryptanalysis and security in general.



Benoît Gérard obtained his Ph.D. in computer science from INRIA Rocquencourt under the supervision of Jean-Pierre Tillich in 2010. He then joined the UCL CryptoGroup (Louvain, Belgium) as a postdoctoral researcher before taking his current research position at DGA.MI (Rennes, France), part of the French defense ministry. He is an expert of side-channel analysis and physical attacks.



**Mehdi Tibouchi** obtained his Ph.D. in computer science from University Paris VII and the University of Luxembourg in 2011 after doctoral studies at École normale supérieure (Paris, France) under the supervision of David Naccache and Jean-Sébastien Coron. He then joined NTT Corporation (Tokyo, Japan), where he has been carrying out research in cryptology ever since. His research interests cover several mathematical aspects of public-key cryptography and cryptanalysis.

## APPENDIX

In [43], Lyubashevsky describes a signature scheme proved secure in the random-oracle model which is an alternative to hash-and-sign methodology of Gentry et al. in [33]. Gentry, Peikert, and Vaikuntanathan were the first to propose a signature scheme whose security is based on the hardness of worst-case lattice problems, while Lyubashevsky and Micciancio present a one-time signature scheme based on the hardness of worst-case ideal lattice problems [45]. Lyubashevsky propose a Fiat–Shamir framework [27] using rejection sampling technique in [42]. Both signature schemes are inefficient in practice: [33] requires megabytes long signature and [42] needs 60,000 bits for reasonable parameters.

Many previous lattice-based signature schemes have been broken since information about the secret key leaks in every signature [24], [32], [34], [52]. Consequently, the basic idea of the Lyubashevsky and BLISS signature schemes is to use the rejection sampling so that the distribution output is independent of the secret key. This signature scheme is proved secure on the hardness of the ring version of  $\ell_2 - \text{SIS}_{q,n,m,\beta}$ .

In the Figure 7, we describe the version of Güneysu et al. in [37] which is a particular instantiation of the ring version of Lyubashevsky signature as presented in [43, §7]. We denote by  $\mathcal{R}_{q,k}$  the subset of  $\mathcal{R}_q$  that consists of all polynomials with integer coefficients in the interval [-k; k]. The hardness assumption of [37] is that  $(\mathbf{a}, \mathbf{t}) \in \mathcal{R}_q \times \mathcal{R}_q$ where **a** is chosen uniformly in  $\mathcal{R}_q$  and  $\mathbf{t} = \mathbf{as}_1 + \mathbf{s}_2$  with  $\mathbf{s}_1$ and  $s_2$  uniformly chosen in  $\mathcal{R}_{q,k}$  is indistinguishable from  $(\mathbf{a},\mathbf{t})$  uniformly chosen in  $\mathcal{R}_q \times \mathcal{R}_q$ . When  $\sqrt{q} < k$ , the solution  $(s_1, s_2)$  is not unique and finding one of them is as hard as worst-case lattice problems in ideal lattices [44], [59]. In [46], it was shown that if  $s_i$  are chosen according to a Gaussian distribution instead of a uniform one, then recovering the  $s_i$  given (a, t) is as hard as solving worstcase lattice problems using a quantum computer. In the following our attacks do not take into account the way the secret key is generated and work in all cases.

#### .1 Description of the PASSSign signature scheme

PASSSign is a signature scheme introduced by Hoffstein et al. in [39]. This scheme is a variant of the PASS and PASS-2 scheme from the same authors, adding the *rejection sampling* technique of Lyubashevsky from 2009. Its hardness is based on the problem of recovering a ring element with small norm from an incomplete description of its Chinese Remainder representation.

We follow in its description the original presentation and notation of [39]. Computations are made in the ring  $\mathbb{Z}_q[x]/(x^N - 1)$ . On that ring, we define  $\mathcal{B}_q^{\infty}$  the subset of polynomials whose coefficients lie in [-k, k]. Given g a primitive N-th root of unity in  $\mathbb{Z}_q$ ,  $\Omega$  a subset of  $\{g^i|1 \leq i \leq N-1\}$ , we define the mapping  $\mathcal{F}_{\Omega} : \mathbb{Z}_q[x]/(x^N-1) \to \mathbb{Z}_q^{|\Omega|}$ consisting in the multi-evaluation of a polynomial on the elements of  $\Omega$ . The image of a polynomial  $\mathbf{f}$  by  $\mathcal{F}_{\Omega}$  will be simply denoted by  $\mathbf{f}|_{\Omega}$ . The function FormatC maps the set of bit strings output by the Hash function H into a set of sparse polynomials. Once again, since its details are not mandatory when mounting the attack, we let the interested reader refer to the original paper for an in-depth description. Its full description is given in Figure 8.

#### .2 Description of the TESLA signature scheme

The TESLA scheme is a variant of the BG scheme presented in [5], initially modified by Dagdelen et al. in [16] to get rid of the forking lemma in the security analysis.

Contrary to the two previous presented schemes, the TESLA signatures works directly on vectors, and no longer on the additional algebraic structure provided by the use of polynomials. The matrix **A** used in the scheme is publicly known and can be seen as a global constant shared by arbitrary many users. The CheckE function is fully described in the original paper from Dagdelen et al. [16] and ensures mandatory properties to preserves that the signature remains short. Once again, we do not fully describe it here since its details are irrelevant for our attacks. We conclude this presentation by noting that the security proof uses the hardness of the LWE problem. Its specificity is to avoid the use of the forking lemma [61] used in the proof of most Fiat–Shamir type signature schemes.

More precisely, we are interested in its variant Ring-TESLA, presented in [1], which offers a provably secure instantiation. Its full description is given in Figure 9.

In this section, we precise a bit more why the attack described on BLISS apply almost straightly to the other members of the Fiat–Shamir family described above: GLP, PASSSign, and Ring-TESLA.

#### .3 On Lyubashevsky Scheme

The difference with BLISS lies in the rejection sampling used and in the generation of the  $y_1, y_2$  commitment coefficients. Thus there is no difference in the way of mounting the attack: here again, only a single fault is only needed to earlyabort the generation loop of the element  $y_1$  and force its degree to be low.

1: function SIGN $(\mu, \mathbf{a}, \mathbf{s}_1, \mathbf{s}_2)$	
2: $\mathbf{y}_1, \mathbf{y}_2 \leftarrow \mathcal{R}_{q,k}$	1: function VERIFY( $\mu$ , $\mathbf{z}_1$ , $\mathbf{z}_2$ , $\mathbf{c}$ , $\mathbf{a}$ , $\mathbf{t}$ )
3: $\mathbf{c} = H(\mathbf{a}\mathbf{y}_1 + \mathbf{y}_2, \mu)$	2: Accept iff $\mathbf{z}_1$ and $\mathbf{z}_2 \in \mathcal{R}_{a,k-32}$ and $\mathbf{c} = H(\mathbf{a}\mathbf{z}_1 + \mathbf{z}_2 - \mathbf{t}\mathbf{c}, \mu)$
4: $\mathbf{z}_1 = \mathbf{s}_1 \mathbf{c} + \mathbf{y}_1,  \mathbf{z}_2 = \mathbf{s}_2 \mathbf{c} + \mathbf{y}_2$	3: end function $(1 + 2 + q)$
5: If $\mathbf{z}_1$ or $\mathbf{z}_2 \notin \mathcal{R}_{q,k-32}$ , goto 1	
6: return $(\mathbf{z}_1, \mathbf{z}_2, \mathbf{c})$	
7: end function	

Fig. 7. Lyubashevsky or [37] signature scheme based on Ring  $\ell_2 - SIS_{q,n,m,\beta}$ . The signing key are  $\mathbf{s}_1, \mathbf{s}_2 \in \mathcal{R}_{q,1}$  where each coefficient of every  $\mathbf{s}_i$  is chosen uniformly and independently from  $\{-1, 0, 1\}$  The verification key is  $(\mathbf{a}, \mathbf{t})$  where  $\mathbf{a} \leftarrow \mathcal{R}_q$  and  $\mathbf{t} = \mathbf{a}\mathbf{s}_1 + \mathbf{s}_2$  The random oracle is modeled by  $H : \{0, 1\}^* \rightarrow \{\mathbf{v} : \mathbf{v} \in \{-1, 0, 1\}^n, \|\mathbf{v}\|_1 \le \kappa\}$  with  $\kappa = 32$  Two sets of parameters for (n, q, k) are given for estimated security of 100 and 256 bits: Set I (512, 8383489, 2^{14}) for a 8,950-bit signature, 1620-bit secret key and 11800-bit public key and Set II (1024, 16760833, 2^{15}) for a 18800-bit signature, 3250-bit secret key and 25000-bit public key.

1: function SIGN( $\mu, f$ )	
1: <b>runction</b> SIGN $(\mu, f)$ 2: $\mathbf{y} \leftarrow \mathcal{B}_{k}^{\infty}$ 3: $\mathbf{h} = H(\mathbf{y} _{\Omega}, \mu)$ 4: $\mathbf{c} = \text{FormatC}(\mathbf{h})$ 5: $\mathbf{z} = \mathbf{y} + \mathbf{f} \cdot \mathbf{c}$ 6: If $\mathbf{z} \notin \mathcal{B}_{k-b}^{\infty}$ , goto 1 7: return $(\mathbf{c}, \mathbf{z}, \mu)$ 8: end function	1: function VERIFY( $\mu$ , c, z, c, f _{\Omega}) 2: Accept iff $\mathbf{z}_2 \in \mathcal{B}_{k-b}^{\infty}$ and $\mathbf{c} = \text{FormatC}(H(\mathbf{z} _{\Omega} - \mathbf{f} \cdot \mathbf{c} _{\Omega}, \mu))$ 3: end function

Fig. 8. Description of the PASSSign signature. The public parameters are: g a primitive N - th root of unity in  $\mathbb{Z}_q$ ,  $\Omega$  a subset of  $\{g^i | 1 \le i \le N-1\}$ , t its cardinal, k the infinity norm of noise polynomials, and b the 1-norm of challenge polynomials. The signing key is the secret  $\mathbf{f} \in \mathbb{Z}_q[X]/(X^n - 1)$  of small norm, that is of  $L_{\infty}$  norm equal to 1. Authors recommend the simple strategy of choosing each coefficient independently and uniformly from 1, 0, 1. The vector  $\mathbf{t}$  is defined as  $\mathbf{as}_1 + \mathbf{s}_2$ . The random oracle is modeled by  $H : \mathbb{Z}_q^t \times \{0, 1\}^* \to \{0, 1\}^l$ . Two sets of parameters for (n, q, k) are given for estimated security of 100 and 128 bits: Set I (769, 1047379, 2^{15} - 1) for a 12624-bit signature, 1600-bit secret key and 7720-bit public key and Set II (1152, 968521, 2^{15} - 1) for a 18800-bit signature, 2000-bit secret key and 12000-bit public key.

Fig. 9. Description of the Ring-TESLA Signature Scheme. The public parameters are  $\mathbf{a}_1, \mathbf{a}_2 \in \mathbb{Z}_q^n, n \in \mathbb{Z}$ . The scheme uses an encoding function:  $F: \{0,1\}^{\kappa} \to \mathcal{B}_{n,\omega}$ , the space of vectors length n and weight  $\omega$ . The random oracle is modeled by  $H: \{0,1\}^{\kappa} \to \{0,1\}^{\kappa}$ . A set of parameters are proposed with security level at least 128 bits. The interesting parameters for us are:  $\kappa = 128, n = 512, q = 39960577, \sigma = 52, U = 3173, d = 23, \omega = 19, L = 2766$  and  $B = 2^{22} - 1$ . The resulting signature size around 1,488B, secret key size around 1,920B and public key size of 3,328B. From the point of view of the attack mounted, we are not interested in the CheckE function and we will not detail it here.

## .4 On Ring-TESLA

In Ring-TESLA, the situation is slightly different since only one element **y** is generated, whose coefficients are drawn uniformly in [-B; B]. Yet, the same early-abort in its generation can be performed to force its degree to be low. Let us suppose that its degree is m - 1; that is, the generation loop has been stopped after m iterations. Then, once again with high probability  $\approx (1 - 1/q)^n$  the element **c** output in signature is invertible and the following equality holds:

$$\mathbf{c}^{-1}\mathbf{z} - \mathbf{s} \equiv \mathbf{c}^{-1}\mathbf{y} \equiv \sum_{i=0}^{m-1} y_i \mathbf{c}^{-1}\mathbf{x}^i \pmod{q} \tag{8}$$

where  $\mathbf{y} = \sum_{i=0}^{m-1} y_i \mathbf{x}^i$ . We can now perform the same trick as in §3.

The analysis yields this time that:

$$\ell + 1 \gtrsim \frac{m + 1 + \frac{\log \sigma}{\log q}}{1 - \frac{\log(\sigma\sqrt{2\pi e})}{\log q}}$$

Then, as in §3, to complete the attack, it suffices to apply the above to a family of subsets J of  $\{0, \ldots, n-1\}$  covering the whole set of indices, which reveals the entire vector s. Recovering the remaining components of the secret key is now a straightforward modular inversion using the public parameters  $\mathbf{a}_1, \mathbf{a}_2$ .

#### .5 On PASSSign

Like in the Ring-TESLA scheme only one **y** is generated when signing and the same attack can be mounted against the generation of this last vector. With regards to the methodology used, the only difference which appears when following the analysis lies in the norm of the secret key f: in PASSSign, the secret key is a polynomial of coefficients independently drawn from  $\{-1, 0, 1\}$ . As such, if using the same notations as before, we get a vector  $\varphi_I(\mathbf{s})$  of norm roughly equals to  $\sqrt{\frac{2\ell}{3} + B^2}$ . We then choose B = 1, which leads to the following inequality on  $\ell$ :

$$\ell + 1 \gtrsim \frac{m+1}{1 - \frac{\log 2 \cdot \sqrt{\left(\frac{\pi e}{3}\right)}}{\log q}}$$

Then, to complete the attack, it suffices to apply the same method to a family of subsets J of  $\{0, ..., n-1\}$  covering the whole set of indices, which reveals the entire secret **f**.

#### .6 Description of the Kyber scheme

Kyber is a key-encapsulation mechanism (KEM), introduced in [12], based on hardness assumptions over module lattices and constructed as a variant of the Fujisaki–Okamoto transform to create a CCA-secure KEM from a CCA-secure encryption scheme. The security of this primitive is based on the hardness of Module-LWE. By consistency with the presentation of our other results, we expose an attack on the key exchange protocol deriving from the KEM. Like in Frodo and NewHope scheme, the shared commitment vectors are of the shape  $\mathbf{A} \cdot \mathbf{s} + \mathbf{e}$ , but with the difference that  $\mathbf{A}$  (resp.  $\mathbf{e}, \mathbf{s}$ ) is no longer an integral matrix nor a ring element but rather a matrix of ring elements (resp. vectors of ring elements). The full outline of the Kyber<sup>8</sup> protocol is given in Figure 10.

## .7 Attacking the key exchange

The problem of recovering the secret vector s for the commitment As + e with a tampered e can be seen as a multidimensional analog of the problem tackled when attacking the BLISS signature or the Frodo key exchange. Hence, one can naturally try to generalize the approach presented in §3 or in §5. Let then suppose that each of the coefficients  $e_i$  of the vector **e** generated by Alice has its degree bounded by m - 1. Once again, suppose that the matrix A is invertible in  $\mathcal{R}_q^{k \times k}$ ; this happens with very high probability: if we heuristically assume that A behaves like a matrix of random elements of the ring from that standpoint, we expect its invertibility to occur with probability about  $\prod_{i=1}^k 1 - q^{-in},$  which is almost greater than 99.9% for the proposed parameters of the scheme. Then (left-)multiplying by  $A^{-1}$  in the the relation  $\mathbf{b} = A\mathbf{s} + \mathbf{e}$  and taking its *i*-th line yields an equation of the form:

$$[A^{-1}]_i \mathbf{b} = [A^{-1}]_i \mathbf{e} + [\mathbf{s}]_i \mod q$$

in  $\mathcal{R}_q$ , where  $[X]_i$  denotes the *i*-th line of the matrix (resp. vector) *X*. Remark now that the element  $[A^{-1}]_i$ e is a  $\mathcal{R}_q$  linear combination of the *n* coefficients of e:

$$\begin{split} [A^{-1}]_{i}\mathbf{e} &= \sum_{j=1}^{k} [A^{-1}]_{i,j} e_{j} \\ &= \sum_{j=1}^{k} \left( \sum_{u=0}^{m-1} [A^{-1}]_{i,j} e_{j}^{(u)} X^{u} \right) \\ &= \sum_{u=0}^{m-1} \underbrace{\left( \sum_{j=1}^{k} [A^{-1}]_{i,j} e_{j}^{(u)} \right) X^{u}}_{\in \operatorname{Span}_{\mathbb{Z}} \left( [A^{-1}]_{i,1} X^{u}, \dots, [A^{-1}]_{i,k} X^{u} \right)} \\ &\in \operatorname{Span}_{\mathbb{Z}} \left( \left( [A^{-1}]_{i,j} X^{u} \right)_{u,j} \right) \end{split}$$

Therefore, the vector  $\mathbf{v} = \mathbf{a}^{-1}\mathbf{b}$  is close to the sublattice  $\mathcal{L}$  of  $\mathbb{Z}^n$  generated by the elements  $([A^{-1}]_{i,j}X^u)_{u,j} \mod q$  for  $j = 0, \ldots, m-1$  and  $u = 1, \ldots, k$ , and  $q\mathbb{Z}^n$ . Then the difference between  $\mathcal{L}$  and  $[A^{-1}]_i\mathbf{b}$  should be precisely the i - th line of s. The methodology of §3 and §5 applies here: one reduce the CVP problem in dimension n to multiple instances of SVP on lattices of rank  $\ell$ ; the analysis gives the following condition on the rank of the aformentioned sublattices:

$$\ell + 1 \gtrsim \frac{km + 1 + \frac{\log \sqrt{\eta/2}}{\log q}}{1 - \frac{\log \sqrt{\eta/2}\sqrt{2\pi e}}{\log q}}.$$
 (9)

The denominator of the right-hand side of (9) is roughly about 0.82 for the paranoid parameters and 0.85 for the light parameters. We thus expect to recover the projections of the lines of s if we can solve the shortest vector problem in a lattice of rank slightly larger than  $1.2 \cdot km$ . To satisfy the

<sup>8.</sup> As mentioned in the Kyber paper [12], the original cryptosystems did not include the "bit-dropping" function Compress<sub>q</sub> in key generation and encryption, but this idea is considered folklore. We present here the version without compression for the sake of simplicity, but it does not change the way the attack is mounted.

Alice		Bob
$\qquad \qquad $		
$\mathbf{A} \leftarrow Sam(seed_1)$		$m \leftarrow \mathcal{U}(\{0,1\}^s)$
$\mathbf{s}, \mathbf{e} \leftarrow \beta_{\eta}^k \times \beta_{\eta}^k = \mathrm{Sam}(seed)$		$\mathbf{A} \leftarrow Sam(seed_1)$
$\mathbf{b} \leftarrow \mathbf{A} \cdot \mathbf{s} + \mathbf{e}$	$\xrightarrow{(\mathbf{t}, seed_1)}$	$(\tilde{K}, r, d) \leftarrow Hash(t, seed_1, m)$
		$\mathbf{s}', \mathbf{e}', e'' \leftarrow \beta_q^k \times \beta_q^k \times \beta_q = \operatorname{Sam}(r)$
		$\mathbf{b}' \leftarrow \mathbf{A}^T \cdot \mathbf{s}' + \mathbf{e}'$
		$\mathbf{v}' \leftarrow \mathbf{t}^T \cdot \mathbf{s}' + e'' + \lceil q/2 \rfloor m$
$m' \leftarrow \lfloor v' - \mathbf{s}^T \mathbf{u} \rceil$	$\overleftarrow{(\mathbf{b}',v')}$	$K \gets \mathrm{Hash}(\tilde{K}, \mathbf{b}', v', d)$
$\mathbf{e}', e'' \leftarrow^{\$} \beta_q^k \times \beta_q = \operatorname{Sam}(r')$		
$(\tilde{K}', r, d) \leftarrow \text{Hash}(\mathbf{u}, seed_1, m)$		
$\mathbf{b} \leftarrow \mathbf{A}^T \cdot \mathbf{s}' + \mathbf{e}'$		
$\mathbf{v} \leftarrow \mathbf{t}^T \cdot \mathbf{s}' + e'' + \lceil q/2 \rfloor m'$		
if $(u, v, d) = (u, v, d)$ then $K \leftarrow \text{Hash}(\tilde{K}, c)$		
else $K \leftarrow \operatorname{Hash}(z, c)$		

Fig. 10. Description of the Kyber scheme with parameters  $(n, q, \chi)$ , and The matrix  $\mathbf{A} \in \mathcal{R}_q^{k \times k}$  is generated from seed via a pseudo-random function SAM. For the parameters described in 10, n = 256, q = 7681, k = 2 (light parameters) or k = 4 (paranoid parameters). The random ring elements are drawn such that each of their coefficients follows a centered binomial law  $\beta_{\eta}$  of parameter  $\eta = 3$  (paranoid) or  $\eta = 5$  (light). Sam is an extendable output function, that is a function on bit strings in which the output can be extended to any desired length.

hypothesis made on the error vector **e**, namely that each of its coefficients has their degree uniformly bounded by a constant  $m \ll n$ , two possibilities occur depending on the generation procedure of this vector. One the one hand, if **e** is drawn coefficient by coefficient, then one must perform a fault on each of the drawings, that is either 2 or 4 faults in a row. The constant *m* is, in this case, the maximum of the loop iterations performed before the faults. On the other hand, if the vector is generated degree by degree, a single fault is necessary: performing an early abort on the signature yields directly the desired property.

The estimation of the success probability of the attack on GPV was lower bounded by the probability that m + krandom vectors  $\mathcal{V} = v_1, \ldots, v_{m+k}$  of  $\mathbb{Z}^m$  generates the full lattice and not a strict full rank sublattice. However, a careful analysis shows that we only need one of the basis vectors of  $\mathbb{Z}^m$  (that is, an integral vector of norm 1) to belong to the sublattice Span<sub> $\mathbb{Z}$ </sub>( $\mathcal{V}$ ).

Although a complete and formal mathematical analysis of such a probability is beyond the scope of this article, an experimental approach is nonetheless possible and interesting. As such extensive experiments were conducted to obtain a numerical estimation of this probability. The sampled vectors  $\mathcal{V} = v_1, \ldots, v_{m+k}$  are drawn uniformly at random into a parallelepiped of norm B, for an increasing constant B. Then a basis of the corresponding sublattice is found by computing a Hermite Normal Form of the generating set  $\mathcal{V}$ . This form allows a polynomial-time membership test for the n vectors of the canonical basis of  $\mathbb{Z}^m$ . For m + 1 vectors, a vector of norm 1 is present in the lattice with probability of about 96%. For m + 2 this probability is greater than 99% and as soon as k gets greater than 3, every lattice sampled in such a way contained at least one of the desired vector.

These results corroborate the simulations conducted on the mounting of the attack presented in  $\S4.2$ .

As we mentioned in §8.2, our fault experiments against NewHope compiled on an XMEGA microcontroller was carried out with clock glitches of width 2.5% of a period, and offset -5.5%. We then used those parameters to pinpoint the clock cycles at which to insert those glitches.

Once the correct clock cycles to cause loop aborts are found, one can re-examine the choice of glitch parameters and check the range of parameters for which reliable faults can be obtained. It turns out that this range is quite large! On Fig. 11, we show the result of glitches of width varying from 0 to 10% of a period and offset varying from -8% to 8%. Roughly one third of this parameter range results in successful attacks.

```
from sage.stats.distributions.
      discrete_gaussian_integer \
          import
      DiscreteGaussianDistributionIntegerSampler
  #BLISS-II parameters
  q=12289
  n=512
  (delta1, delta2) = (0.3, 0)
  sigma=10
  kappa=23
  R.<xx>=QuotientRing(ZZ[x], ZZ[x].ideal(x^n+1))
 Rq.<xxx>=QuotientRing(GF(q)[x], GF(q)[x].ideal(x^{(1)})
      n+1))
  sampler =
      DiscreteGaussianDistributionIntegerSampler(
      sigma=sigma, algorithm='uniform+table')
17 def slgen():
      s1vec=[0]*n
```

Fig. 11. Results of faults with glitches of various parameters. Green bullets denote success, grey bullets denote failure (no early loop abort) and red bullets denote device malfunction (reset, freeze, memory dump, etc.).



```
def faultattack(d,e,bkz_size=25):
       sl=slgen()
59
       (c,z1)=faultyz1gen(s1,d)
       try:
           cinv=1/Rq(c.lift())
59
       except ZeroDivisionError:
           print "c not invertible"
61
           return s1,c,z1,matrix(ZZ,[])
63
       .....
       Try to recover the first e coefficients of s1
65
       (of course, if we succeed, we should succeed
       for *all* sets of
       e coefficients of s1, so we can recover the
67
       whole secret key).
       print "Starting attack"
69
       t=cputime(subprocesses=True)
       latvec=[(cinv*xxx^i).lift().list()[:e] for i
       in range(d)]
       latvec=[(cinv*Rq(z1.lift())).lift().list()[:e
       11 + latvec
       latvec=latvec+[[0]*i + [q] + [0]*(e-i-1) for
75
       i in range(e)]
       M=matrix(ZZ,latvec)
       M=M.augment(matrix(ZZ,e+d+1,1,[2*q]+[0]*(e+d)
       ))
       if bkz_size is None:
79
           M=M.LLL()
       else:
81
           M=M.BKZ(block_size=bkz_size)
       v=M[d+e]
83
       v = v \star (2 \star q / v [-1])
85
       print "Attack time:", cputime(subprocesses=
       True)-t
       print "Recovered vector:", v[:-1]
87
       print "Truncated key:", s1.lift().list()[:e]
89
       return s1, c, z1, M
91
   def faultattack_multiple(d,e,bkz_size=None,tries
       =100):
       succ=0
93
       secs=0.0
       for _ in range(tries):
95
           s1=s1gen()
97
           while True:
               (c, z1) = faultyz1gen(s1, d)
99
               try:
                    cinv=1/Rq(c.lift())
               except ZeroDivisionError:
                    print "*",
                    sys.stdout.flush()
                    continue
105
               break
107
           t=cputime(subprocesses=True)
109
           latvec=[(cinv*xxx^i).lift().list()[:e]
       for i in range(d)]
           latvec=[(cinv*Rg(z1.lift())).lift().list
       ()[:e]] + latvec
           latvec=latvec+[[0]*i + [q] + [0]*(e-i-1)
       for i in range(e)]
           M=matrix(ZZ,latvec)
           M=M.augment(matrix(ZZ,e+d+1,1,[2*q]+[0]*(
       e+d)))
           if bkz size is None:
```

M=M.LLL()

```
else:
               M=M.BKZ(block_size=bkz_size)
           v=M[d+e]
           v=v*(2*q/v[-1])
           t=cputime(subprocesses=True)-t
           secs+=float(t)
           if v[:-1].list() == s1.lift().list()[:e]:
               succ+=1
               print "+",
           else:
               print ".",
           sys.stdout.flush()
      print
      print "Success: %d/%d (%f%%)" % (succ,tries
       ,100*RR(succ/tries))
      print "Avg CPU time:", secs/tries
      print "Avg CPU time (total vec):", secs/tries
       *ceil(n/e*tries/succ)
  def theoretical_lattice_size(d):
       r=delta1+4*delta2
139
      u=0.5 \pm \log(r) / \log(q)
      v=0.5*log(2*pi*exp(1)*r)/log(q)
      return RR(((d+1+u)/(1-v))-1)
141
  # vim: ft=python
143
```

Listing 1. Attack on BLISS scheme

```
from sage.stats.distributions.
      discrete_gaussian_integer \
          import
      DiscreteGaussianDistributionIntegerSampler
  from sage.stats.distributions.
      discrete_gaussian_polynomial \
          import
      DiscreteGaussianDistributionPolynomialSampler
  q=1021
  n=256
  sigmaf=1.17*sqrt(q/(2*n))
  sigma=1.17*sqrt(q)
  x=polygen(ZZ)
  R.<xx>=QuotientRing(ZZ[x], ZZ[x].ideal(x^n+1))
  Rq.<xxx>=QuotientRing(Integers(q)[x], Integers(q)
      [x].ideal(x^n+1))
15 K=QuotientRing(QQ[x], QQ[x].ideal(x^n+1))
  def norm12(1):
      return sqrt(sum([RR(x)^2 for x in 1]))
19
  def anticirculant(f):
      return Matrix(ZZ, [ (xx^i*f).list() for i in
      range(n) ])
  def gpvkeygen():
      fsampler =
      DiscreteGaussianDistributionPolynomialSampler
      (R,n,sigmaf)
      while True:
          f,g = fsampler(), fsampler()
          fbar = K(f.lift().subs(-xx^{255}))
          gbar = K(g.lift().subs(-xx^255))
29
          f2 = q \star fbar / (f \star fbar + g \star gbar)
31
          q2 = q \star q bar / (f \star f bar + q \star q bar)
          norm = max(norml2(f.list() + g.list()), \
                     norml2(f2.list() + g2.list()))
37
          if norm > sigma:
              continue
39
```

```
Rf, rhof, \_ = xgcd(f.lift(), x<sup>n+1</sup>)
           Rg, rhog, _ = xgcd(g.lift(), x^n+1)
gg, u, v = xgcd(Rf, Rg)
           if gg == 1 and gcd(Rf,g) == 1:
               break
      F = q \star v \star rhoq
      G =-q*u*rhof
      while True:
           k = (F \star f bar + G \star g bar) / (f \star f bar + g \star g bar)
           kl= [ floor(c+0.5) for c in k.list() ]
           k = sum([ kl[i]*xx^i for i in range(len(
      kl)) ])
           if k.lift().degree() < 0:</pre>
               break
           F-= k★f
           G−= k*q
      h = Rq(q)/Rq(f)
      B = block_matrix( [[anticirculant(g),
      anticirculant(-f)],
                            [anticirculant(G).
      anticirculant(-F)]] )
      return h, f, q, F, G, B
67 def lowerkey(f,g):
      Rf, rhof, \_ = xgcd(f.lift(), x^n+1)
      Rg, rhog, \_ = xgcd(g.lift(), x<sup>n+1</sup>)
                 v = xgcd(Rf, Rg)
      gg, u,
      if gg != 1 or gcd(Rf,q) != 1:
           raise ValueError, "not coprime"
      F = q \star v \star rhoq
      G = -q \star u \star rhof
      fbar = K(f.lift().subs(-xx^(n-1)))
      gbar = K(g.lift().subs(-xx^(n-1)))
      while True:
           k = (F \star fbar + G \star gbar) / (f \star fbar + g \star gbar)
           kl= [ floor(c+0.5) for c in k.list() ]
           k = sum([ kl[i]*xx^i for i in range(len(
      kl)) ])
           if k.lift().degree() < 0:</pre>
               break
           F-= k★f
           G−= k★q
      return F.G
  def rndvec(v):
      return [x if 2*x<q else x-q for x in v]</pre>
  def gpvsign(B, fault=2*n, verbose=False, m=None):
      if m is None:
          t = Rq.random_element()
      else:
           t = m # assume m is a hash value in Rq
      Bgram, T = B.change_ring(RDF).gram_schmidt()
      Bgram = matrix(RDF, [T[i,i]*Bgram.row(i) for
      i in range(2*n)])
      v = vector(ZZ, 2*n)
      c = vector(ZZ,2*n, rndvec(t.lift().list()) +
      [0]*n)
      for i in range(2*n-1,2*n-1-fault,-1):
```

41

43

45

47

49

51

55

57

59

63

65

69

71

77

79

81

83

85

87

89

93

95

97

90

101

103

105

107

109

```
b = Bgram.row(i)
           s = sigma/b.norm()
           k = c.dot_product(b)/b.norm()^2
           z =
       DiscreteGaussianDistributionIntegerSampler(
       sigma=s,c=k,\
                    algorithm="uniform+online")()
           if verbose:
117
               print z, RR(k), RR(c.norm()), b.norm(),
       RR(c.norm())/b.norm()
           c = c - z \star B.row(i)
           v = v + z \star B.row(i)
121
       return t, vector(ZZ,n,rndvec(t.lift().list())
       )-v[:n], -v[n:] # (t,s1,s2)
  def test_gpvfault(B,m,d,bkz=None):
       .....
       Try to recover F from m faults with the
       iterations stopping at d.
       .....
129
       print "Computing the m=%d faulty signatures"
       % m
       sigs = [gpvsign(B,d)[2] for _ in range(m)]
       if bkz is None:
           print "Trying to reduce with LLL"
           M = Matrix(ZZ, sigs).LLL()
135
       else:
          print "Trying to reduce with BKZ-%d" %
       bkz
           M = Matrix(ZZ, sigs).BKZ(block_size=bkz)
139
       P = [sum([M[k,i]*xx^i for i in range(n)]) for
        k in range(m)]
      F = sum([B[-1,n+i]*xx<sup>i</sup> for i in range(n)])
print "P_i/F =", [K(Pi)/K(F) for Pi in P]
141
143
       zeta = filter(lambda x: x!=0, list(K(P[m-d])/
       K(F)))
       return zeta==[1] or zeta==[-1]
145
147
  def test_gpvfault_multiple(B,m,d,bkz=None,tries
       =100):
       succ=0
149
       secs=0.0
       for _ in range(tries):
           sigs = [gpvsign(B,d)[2] for _ in range(m)
       ]
           t=cputime(subprocesses=True)
           if bkz is None:
               M = Matrix(ZZ, sigs).LLL()
157
           else:
               M = Matrix(ZZ, sigs).BKZ(block_size=
       bkz)
           t=cputime(subprocesses=True)-t
159
           secs=secs+float(t)
161
          P = [sum([M[k,i]*xx^i for i in range(n)])
        for k in range(m)]
           F = sum([B[-1, n+i] * xx^i for i in range(n)]
163
       ])
           zeta = filter(lambda x: x!=0, list(K(P[m-
165
       d])/K(F)))
           if zeta==[1] or zeta==[-1]:
167
               succ+=1
               print "+",
169
           else:
              print ".",
           sys.stdout.flush()
```

```
173 print
print "Success: %d/%d (%f%%)" % (succ,tries
,100*RR(succ/tries))
print "Avg CPU time:", secs/tries
177
# vim: ft=python
```

```
Listing 2. Attack on GPV scheme
```