Loop abort Faults on Lattice-Based Fiat-Shamir & Hash'n Sign signatures

Thomas Espitau[4], Pierre-Alain Fouque[2],
Benoît Gérard[1], and Mehdi Tibouchi[3]

[1] DGA.MI & IRISA [‡]
[2] NTT Secure Platform Laboratories [§]
[3] Institut Universitaire de France & IRISA & Université de Rennes I [¶]
[4] Ecole Normale Supérieure de Cachan & Sorbonne Universités, UPMC Univ Paris 06, LIP6[‖]

Abstract. As the advent of general-purpose quantum computers appears to be drawing closer, agencies and advisory bodies have started recommending that we prepare the transition away from factoring and discrete logarithm-based cryptography, and towards postquantum secure constructions, such as lattice-based schemes.
Almost all primitives of classical cryptography (and more!) can be realized with lattices, and the efficiency of primitives like encryption and signatures has gradually improved to the point that key sizes are competitive with RSA at similar security levels, and fast performance can be achieved both in software and hardware. However, little research has been conducted on physical attacks targeting concrete implementations of postquantum cryptography in general and lattice-based schemes in particular, and such research is essential if lattices are going to replace RSA and elliptic curves in our devices and smart cards.
In this paper, we look in particular at fault attacks against some instances of the Fiat-Shamir family of signature scheme on lattices (BLISS, GLP, TESLA and PASSSign) and on the GPV scheme, member of the Hash'n Sign family. Some of these schemes have achieved record-setting efficiency in software and hardware. We present several possible fault attacks, one of which allows a full key recovery with as little as a single faulty signature, and discuss possible countermeasures to mitigate these attacks.

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

1 Introduction

Recent progress in quantum computation [DBI+15], the NSA advisory memorandum recommending the transition away from Suite B and to postquantum cryptography [NSA16], as well as the announcement of the NIST standardization process for postquantum cryptography [CJL+16] 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 postquantum directions, lattice-based cryptography occupies a position of particular interest, as it relies on well-studied problems and comes with uniquely strong security guarantees, such as worst-case to average-case reductions [Pei15]. A number of works have also focused on improving the performance of lattice-based schemes, and actual implementation results suggest that properly optimized schemes may be competitive with, or even outperform, classical factoring- and discrete logarithm-based cryptography.

[‡] benoit.gerard@irisa.fr
[§] tibouchi.mehdi@lab.ntt.co.jp
[¶] pierre-alain.fouque@univ-rennes1.fr
[‖] tespitau@ens-cachan.fr

Among the possible signatures schemes, one can devise schemes provable in the random oracle model into two categories. One the one hand the ones constructed using the *Full Domain Hash* (FDH) approach, and on the other hand schemes built with the *Fiat-Shamir* technique. This work focuses on the study of the effect of Loop abort faults type for both of the construction types, in the general framework of lattice-based cryptography. More precisely, we get interested in four instances of the Fiat-Shamir family: BLISS, PASSSign, GLP and TESLA, and one instance of the Hash'n Sign family: GPV. BLISS (Bimodal Lattice Signature Scheme) is a lattice-based signature scheme that incorporates many of the performance improvements geared towards reducing parameter sizes without security compromises, by careful analysis of the hardness of the underlying problems and of the statistical properties of the randomness distributions involved. It has been proposed by Ducas et al. in [DDLL13], building upon the previous signature scheme of Lyubashesky [Lyu12], and it has been implemented in hardware by Pöppelmann et al. in a paper presented at CHES 2014 [PDG14]. The Lyubashevsky signature scheme had also been implemented in hardware at CHES 2012 by Güneysu et al. [GLP12]. Both signature schemes are interesting targets for the cryptanalyst, since they are actually designed with real world practicality in mind, and have been the subject of concrete implementation efforts. PASSSign is signature scheme developed as a variant of the PASS and PASS-2 signatures, introduced by Hoffstein et al. in 2014 ACNS paper [HPS+14]. Its hardness is based on the problem of recovering a ring element with small norm from an incomplete description of its Chinese remainder representation. Eventually, TESLA was introduced by Alkim et al. in [ABBD15] as a tightly secure scheme based on the learning with errors (LWE) problem.

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; in addition, ring-based schemes have recently been subjected to new families of attacks that might eventually reduce their security, especially in the postquantum setting). 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! [TE15]). 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 [BDL01,BMM00] 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 some Lattice-Based signatures.

Our contributions. In this work, we initiate the study of fault attacks against lattice-based signature schemes. Early lattice-based signature schemes with heuristic security have been broken using standard attacks [GJSS01,GS02,NR09] but recent signature schemes [GPV08,Lyu09,Lyu12,DDLL13] are provably secure, and cryptanalysis probably requires a more powerful attack model. We therefore look at fault attacks. To the best of our knowledge, the only previous work in that direction is a fault attack against NTRUSign [HHP+03], due to Kamal and Youssef [KY12]. It is, however, of limited interest since NTRUSign is known to be broken [NR09,DN12]; it also suffers from a very low probability of success.

In this work, we consider faults against implementations of the Fiat-Shamir representents, both on the software [Lyu12,DDLL13] and hardware [GLP12,PDG14], and a single fault attack on the GPV scheme of the Hash'n Sign family. For both families, we present new fault attacks, the main idea of which is that some polynomials are generated coefficient by coefficient, and thus, we may use faults to abort the generation loop early. Similar fault attacks have been described on DSA [NNTW05] or pairing computations [PV06]. Using such faults, our first attack uses LLL to solve a closest vector problem in a lattice, and using a single successful faulty signature makes it possible to recover the entire secret key. The second attack on [GLP12] uses only a single fault and also relies on lattice reduction techniques to exploit this fault. We then take a closer look at the concrete software and hardware implementations, discuss how we can inject the required faults in practice, and propose countermeasures.

Organization of the paper. In Section 2, we give a basic description of the five schemes attacked (GLP, BLISS, PASSSign, TESLA on the one hand, and GPV on the other). In Section 3, we present all our fault

attacks on these latter schemes and in Section 4 we explain how we can inject the required faults in the concrete implementations. Finally, Section 5 is devoted to possible countermeasures.

2 Description of the considered Lattice-Based Signatures

2.1 Notation, Gaussian distributions and hardness assumptions

For any integer q, we represent the ring $\mathbb{Z}_q$ by $[-q/2, q/2) \cap \mathbb{Z}$. For any ring $\mathcal{R}$, we denote its quotient ring by $\mathcal{R}_q$ as the ring $\mathcal{R}/(q\mathcal{R})$. The BLISS signature scheme uses two quotient rings $\mathcal{R}_q = \mathbb{Z}_q[\mathbf{x}]/(\mathbf{x}^n + 1)$ and $\mathcal{R}_{2q} = \mathbb{Z}_{2q}[x]/(x^n + 1)$ where n is a power of two and q is a prime such that $q = 1 \pmod{2n}$. The elements of $\mathcal{R}_q$ can be represented by polynomials of degree $n-1$ with coefficients in the range $\mathbb{Z}_q$. We define $\mathbb{B} = \{0, 1\}$ and $\mathbb{B}_w^n$ the set of binary vectors of length n and Hamming weight w (i.e. vectors with exactly w out of n non-zero bits). Vectors are considered as column vectors and will be written in bold lower case letters and matrices with upper case letters. By default, we will use the ℓ_2 Euclidean norm, $\|\mathbf{v}\|_2 = (\sum_i v_i^2)^{1/2}$ and ℓ_∞-norm as $\|\mathbf{v}\|_\infty = \max_i |v_i|$.

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 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 $D_\sigma^m(\mathbf{x}) = \rho_\sigma(\mathbf{x})/\rho_\sigma(\mathbb{Z}^m)$.

All the constructions in this paper are based on the hardness of the generalized SIS (Short Integer Solution) problem, which is connected to hard lattices problems.

Definition 1. *($\mathcal{R} - \mathrm{SIS}_{q,n,m,\beta}^{\mathcal{K}}$ **problem**). Let $\mathcal{R}$ be some ring and $\mathcal{K}$ be some distribution over $\mathcal{R}_q^{n\times m}$, where $\mathcal{R}_q$ is the quotient ring $\mathcal{R}/(q\mathcal{R})$. Given a random $\mathbf{A} \in \mathcal{R}_q^{n\times m}$ drawn according to the distribution $\mathcal{K}$, find a non-zero $\mathbf{v} \in \mathcal{R}_q^m$ such that $\mathbf{A}\mathbf{v} = \mathbf{0}$ and $\|\mathbf{v}\|_2 \leq \beta$.*

If $\mathcal{R} = \mathbb{Z}$ and $\mathcal{K}$ be the uniform distribution, then the resulting problem is the classical SIS problem first defined by Ajtai in his seminal paper [Ajt96] showing connections between worst-case lattice problems and the average-case SIS problem. If $\beta \geq \sqrt{m}q^{n/m}$, the SIS instances are guaranteed to have a solution. In [LM08], Lyubashevsky and Micciancio show that if $\mathcal{R} = \mathbb{Z}_q[\mathbf{x}]/(\mathbf{x}^n + 1)$ with n a power of 2, then the $\mathcal{R} - \mathrm{SIS}_{q,n,m,\beta}^{\mathcal{K}}$ problem is as hard as the $\tilde{O}(\sqrt{n}\beta) - \mathrm{SVP}$ problem in all lattices that are ideal in $\mathcal{R}$ (where $\mathcal{K}$ is the uniform distribution over $\mathcal{R}_q^{1\times m}$).

Another hardness assumption used in the security reduction of some of the schemes presented in this paper is the LWE (learning with errors) problem. We first start by defining the LED which is the probability distribution used in the definition of the LWE problem.

Definition 2. *($\mathrm{LWE}_{q,n,m,\xi}$ **distribution**). Let $n, m, q > 0$ integers and ξ a distribution over $\mathbb{Z}$. We define by $\mathcal{D}_{s,\xi}$ LWE distribution which outputs $(a, \langle a, s\rangle + e) \in \mathbb{Z}_q^n \times \mathbb{Z}^q$, where a is drawn uniformly at random in $\mathbb{Z}_q^n$ and e under ξ.*

We are now able to define the LWE problem — in its decisional version since it will be the only one used in the security reduction —.

Definition 3. *($\mathrm{LWE}_{q,n,m,\xi}$ **problem**) Let $n, m, q > 0$ be integers and ξ be a distribution over $\mathbb{Z}$. Moreover, define $\mathcal{O}_\xi$ to be an oracle, which upon input vector $\mathbf{s} \in \mathbb{Z}_q^n$ returns samples from the distribution $\mathcal{D}_{s,\xi}$. The decisional learning with errors problem $LWE_{n,m,q,\xi}$ is (t, ϵ)-hard if for any algorithm $\mathcal{A}$, running in time t and making at most m queries to its oracle, we have*

$$\left|\Pr[\mathcal{A}^{\mathcal{O}_\xi(\mathbf{s})}(\cdot) = 1] - \Pr[\mathcal{A}^{\mathcal{U}(\mathbb{Z}_q^n \times \mathbb{Z}_q)}(\cdot) = 1]\right| \leq \epsilon$$

where the probabilities are taken over $s \leftarrow \mathcal{U}(\mathbb{Z}_q^n)$ and the random choice of the distribution $\mathcal{D}_{s,\xi}$, as well as the random coins of $\mathcal{A}$.

2.2 Description of the Lyubashevsky signature scheme

In [Lyu12], 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 [GPV08]. 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 [LM08]. Lyubashevsky propose a Fiat–Shamir framework [FS86] using rejection sampling technique in [Lyu09]. Both signature schemes are inefficient in practice: [GPV08] requires megabytes long signature and [Lyu09] 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 [GJSS01,GS02,NR09,DN12]. 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 $\ell_2 - \mathrm{SIS}_{q,n,m,\beta}$.

In the following, we describe the version of Güneysu et al. in [GLP12] which is a particular instantiation of the ring version of Lyubashevsky signature as presented in Section 7 in [Lyu12]. 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 [GLP12] is that $(\mathbf{a}, \mathbf{t}) \in \mathcal{R}_q \times \mathcal{R}_q$ where $\mathbf{a}$ is chosen uniformly in $\mathcal{R}_q$ and $\mathbf{t} = \mathbf{a}\mathbf{s}_1 + \mathbf{s}_2$ with $\mathbf{s}_1$ and $\mathbf{s}_2$ 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 $(\mathbf{s}_1, \mathbf{s}_2)$ is not unique and finding one of them is as hard as worst-case lattice problems in ideal lattices [LM06,PR06]. In [LPR13], it was shown that if $\mathbf{s}_i$ are chosen according a Gaussian distribution instead of a uniform one, then recovering the $\mathbf{s}_i$ given $(\mathbf{a}, \mathbf{t})$ is as hard as solving worst-case 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.

2.3 Description of BLISS

The BLISS signature scheme is a more efficient version of Lyubashevsky signature scheme. The rejection sampling has been optimized so that the number of times we need to reject is diminished using a bimodal distribution. The key generation has also been updated to generate the $\mathbf{s}_i$ using NTRU ideas. We do not define the parameter $N_\kappa(S)$ (the interested reader can read it in [DDLL13]), used only in the key generation algorithm, which is a technical parameter that is not important in our attacks. Finally, the $\mathbf{z}_2$ is compressed in $\mathbf{z}_2^\dagger$ in BLISS, but since our attack does not involve this value, we do not describe how the compression works. The value ζ is defined as $\zeta \cdot (q-2) = 1 \bmod 2q$. The only modification important for us is that H as now range $\{\mathbf{v} : \mathbf{v} \in \mathbb{B}^n, \|\mathbf{v}\|_1 \leq \kappa\}$.

2.4 Description of the PASSSign signature scheme

PASSSign is a signature scheme introduced by Hoffstein et al. in [HPS+14]. 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 [HPS+14]. 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 $\mathbb{Z}_q$, Ω 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 on the multi-evaluation of a polynomial on the elements of Ω. The image a a polynomial $\mathbf{f}$ by $\mathcal{F}_\Omega$ will be simply denoted by $\mathbf{f}|_\Omega$. The function FormatC maps the set of bitstrings 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 to refer to the original paper for an in-depth description.

Signing Key: $\mathbf{s}_1, \mathbf{s}_2 \in \mathcal{R}_{q,1}$ where each coefficient of every $\mathbf{s}_i$ is chosen uniformly and independently from $\{-1, 0, 1\}$
Verification Key: $(\mathbf{a}, \mathbf{t})$ where $\mathbf{a} \leftarrow \mathcal{R}_q$ and $\mathbf{t} = \mathbf{a}\mathbf{s}_1 + \mathbf{s}_2$
Random Oracle: $H : \{0,1\}^* \rightarrow \{\mathbf{v} : \mathbf{v} \in \{-1, 0, 1\}^n, \|\mathbf{v}\|_1 \leq \kappa\}$ with $\kappa = 32$

function SIGN($\mu, \mathbf{a}, \mathbf{s}_1, \mathbf{s}_2$)
  $\mathbf{y}_1, \mathbf{y}_2 \leftarrow \mathcal{R}_{q,k}$
  $\mathbf{c} = H(\mathbf{a}\mathbf{y}_1 + \mathbf{y}_2, \mu)$
  $\mathbf{z}_1 = \mathbf{s}_1\mathbf{c} + \mathbf{y}_1$, $\mathbf{z}_2 = \mathbf{s}_2\mathbf{c} + \mathbf{y}_2$
  If $\mathbf{z}_1$ or $\mathbf{z}_2 \notin \mathcal{R}_{q,k-32}$, goto 1
  return $(\mathbf{z}_1, \mathbf{z}_2, \mathbf{c})$
end function

function VERIFY($\mu, \mathbf{z}_1, \mathbf{z}_2, \mathbf{c}, \mathbf{a}, \mathbf{t}$)
  Accept iff $\mathbf{z}_1$ and $\mathbf{z}_2 \in \mathcal{R}_{q,k-32}$ and $\mathbf{c} = H(\mathbf{a}\mathbf{z}_1 + \mathbf{z}_2 - \mathbf{t}\mathbf{c}, \mu)$
end function

Fig. 1. Lyubashevsky or [GLP12] signature scheme based on Ring $\ell_2-\text{SIS}_{q,n,m,\beta}$.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.

function KEYGEN()
  Choose $\mathbf{f}, \mathbf{g}$ as uniform polynomials with exactly $d_1 = \lceil \delta_1 n \rceil$ entries in $\{\pm 1\}$ and $d_2 = \lceil \delta_2 n \rceil$ entries in $\{\pm 2\}$
  $\mathbf{S} = (\mathbf{s}_1, \mathbf{s}_2)^T \leftarrow (\mathbf{f}, 2\mathbf{g} + 1)^T$
  If $N_\kappa(\mathbf{S}) \geq C^2 \cdot 5 \cdot (\lceil \delta_1 n \rceil + 4\lceil \delta_2 n \rceil) \cdot \kappa$ **then restart**
  $\mathbf{a}_q = (2\mathbf{g} + 1)/\mathbf{f} \bmod q$ (**restart** if $\mathbf{f}$ is not invertible)
  return $(pk = \mathbf{A}, sk = \mathbf{S})$ where $\mathbf{A} = (\mathbf{a}_1 = 2\mathbf{a}_q, q - 2) \bmod 2q$
end function

function SIGN($\mu, pk = \mathbf{A}$, $sk = \mathbf{S}$)
  $\mathbf{y}_1, \mathbf{y}_2 \leftarrow D^n_{\mathbb{Z},\sigma}$
  $\mathbf{u} = \zeta \cdot \mathbf{a}_1 \cdot \mathbf{y}_1 + \mathbf{y}_2 \bmod 2q$
  $\mathbf{c} \leftarrow H(\lfloor \mathbf{u} \rceil_d \bmod p, \mu)$
  Choose a random bit b
  $\mathbf{z}_1 \leftarrow \mathbf{y}_1 + (-1)^b \mathbf{s}_1\mathbf{c}$
  $\mathbf{z}_2 \leftarrow \mathbf{y}_2 + (-1)^b \mathbf{s}_2\mathbf{c}$
  Continue with probability $1/\big(M \exp(-\|\mathbf{S}\mathbf{c}\|/(2\sigma^2)) \cosh(\langle \mathbf{z}, \mathbf{S}\mathbf{c} \rangle / \sigma^2)\big)$ otherwise **restart**
  $\mathbf{z}_2^\dagger \leftarrow (\lfloor \mathbf{u} \rceil_d - \lfloor \mathbf{u} - \mathbf{z}_2 \rceil_d) \bmod p$
  return $(\mathbf{z}_1, \mathbf{z}_2^\dagger, \mathbf{c})$
end function

function VERIFY($\mu, \mathbf{A}, (\mathbf{z}_1, \mathbf{z}_2^\dagger, \mathbf{c})$)
  If $\|(\mathbf{z}_1 | 2^d \cdot \mathbf{z}_2^\dagger)\|_2 > B_2$ **then** Reject
  If $\|(\mathbf{z}_1 | 2^d \cdot \mathbf{z}_2^\dagger)\|_\infty > B_\infty$ **then** Reject
  Accept iff $\mathbf{c} = H(\lfloor \zeta \cdot \mathbf{a}_1 \cdot \mathbf{z}_1 + \zeta \cdot q \cdot \mathbf{c} \rceil_d + \mathbf{z}_2^\dagger \bmod p, \mu)$
end function

Fig. 2. BLISS Signature Scheme. Four different sets of parameters are proposed 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 $\kappa \in \{23, 30, 39\}$, resulting in signature size around 5kb, secret key size around 2kb and 3kb and public key size of 7kb.

Public Parameters: g a primitive $N-th$ root of unity in $\mathbb{Z}_q$, Ω a subset of $\{g^i | 1 \leq i \leq N-1\}$, t its cardinal, k the infinity norm of noise polynomials, and b the 1-norm of challenge polynomials.
Signing Key: Secret $\mathbf{f} \in \mathbb{Z}_q[X]/(X^n-1)$ of small norm. $\mathbf{t} = \mathbf{a}\mathbf{s}_1 + \mathbf{s}_2$
Random Oracle: $H : \mathbb{Z}_q^t \times \{0,1\}^* \to \{0,1\}^l$

1: **function** SIGN(μ, f)
2: $\quad \mathbf{y} \leftarrow \mathcal{B}_k^\infty$
3: $\quad \mathbf{h} = H(\mathbf{y}|_\Omega, \mu)$
4: $\quad \mathbf{c} = \text{FormatC}(\mathbf{h})$
5: $\quad$ If $\mathbf{z} \notin \mathcal{B}_{k-b}^\infty$, goto 1
6: $\quad$ **return** $(\mathbf{c}, \mathbf{z}, \mu)$
7: **end function**

1: **function** VERIFY($\mu, \mathbf{c}, \mathbf{z}, \mathbf{c}, \mathbf{f}|_\Omega$)
2: $\quad$ 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. 3. PASSSign signature. 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.

2.5 Description of the TESLA signature scheme

The TESLA scheme is a variation of the BG scheme presented in [Ben14], initially modified by Dagdelen et al. in [DBG+14] at LATINCRYPT 14, allowing to get rid of the forking lemma in their security analysis.

On the contrary of the two previous presented schemes, the TESLA signatures works directly on vectors — and no more on the additional algebraic structure provided by the use of polynomials —. The matrix $\mathbf{A}$ 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 [DBG+14] 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* proposed by Pointcheval and Stern in [PS96].

2.6 GPV Signature Scheme

This signature is described in [DLP14] by Ducas, Lyubashevsky and Prest and derived from an IBE scheme. We denote here by $\overline{\mathbf{f}} = f_0 - \sum_{i=1}^{n-1} f_{n-i} x^i$ where $\mathbf{f} = \sum_{i=0}^{n-1} f_i x^i$ and by $\mathbf{B}$ the matrix $\mathbf{B} = \begin{pmatrix} \mathbf{M_g} & -\mathbf{M_f} \\ \mathbf{M_G} & -\mathbf{M_F} \end{pmatrix}$, where $\mathbf{F}$ and $\mathbf{G}$ are defined by $\mathbf{fG} - \mathbf{gF} = q$ and $\mathbf{M_a}$ represents the matrix of the multiplication in $\mathcal{R}$ by the element $\mathbf{a}$, which is anticirculant of dimension n.

Public parameters: $\mathbf{A} \in \mathbb{Z}_q^{m \times n}, m, n \in \mathbb{Z}$.
Encoding Function: $F : \{0,1\}^\kappa \to \mathcal{B}_{n,\omega}$, space of vectors length n and weight ω. Random Oracle: $H : \{0,1\}^* \to \{0,1\}^\kappa$.

function KeyGen()
  $\mathbf{S} \leftarrow D_\sigma^{n \times n}$
  $\mathbf{E} \leftarrow D_\sigma^{n \times n}$
  If not CheckE($\mathbf{E}$) **then Restart**
  return $(pk = \mathbf{T}, sk = (\mathbf{S}, \mathbf{E}))$ where $\mathbf{T} = (\mathbf{AS} + \mathbf{E} \bmod q$
end function

function Sign($\mu, pk = \mathbf{A}$, $sk = \mathbf{S}$)
  $\mathbf{y} \leftarrow^{\$} [-B; B]^n$
  $\mathbf{v} = \mathbf{Ay} \bmod q$
  $\mathbf{c} \leftarrow H(\lfloor \mathbf{v} \rceil_d, \mu)$
  $\mathbf{c} \leftarrow F(\mathbf{c})$
  $\mathbf{z} \leftarrow \mathbf{y} + \mathbf{sc}$
  $\mathbf{w} \leftarrow \mathbf{v} - \mathbf{Ec} \bmod q$
  If $|[\mathbf{w_i}]_{2^d}| > 2^{d-1} - L$ **or** $\|\mathbf{z}\|_\infty > B - U$ **then** Restart.
  return $(\mathbf{z}, \mathbf{c})$
end function

function Verify($\mu, \mathbf{A}, (\mathbf{z}_1, \mathbf{z}_2^\dagger, \mathbf{c})$)
  $\mathbf{c} \leftarrow F(\mathbf{c})$
  $\mathbf{w}' \leftarrow \mathbf{Az} - \mathbf{Tc} \bmod q$
  $\mathbf{c}' \leftarrow H(\lfloor \mathbf{w}' \rceil_d, \mu)$ **Accept iff** $\mathbf{c}' = \mathbf{c}$ **and** $\|\mathbf{z}\|_\infty \leq B - U$
end function

Fig. 4. TESLA Signature Scheme. A set of parameters are proposed with security level at least 128 bits. The interesting parameters for us are: $\kappa = 128$, $n = 416$, $m = 800$, $q = 2^{27} - 39$, $\sigma = 114$, $U = 7138$, $d = 24$, $\omega = 20$, $L = 6042$. The resulting signature size around 10.24kb, secret key size around 1Mb and public key size of 1.33Mb.

3 Description of our attacks

In this section we describe two different fault attacks against the previously described signature schemes. The first attack on all the schemes consists in reducing the polynomial $\mathbf{y}_1$ to only a few monomials (say 64 or 128 among 512) and allows to recover $\mathbf{s}_1$ with an efficient lattice attack. The second fault attack is aimed against the GPV scheme, targeting the vector $\mathbf{s}$ of the signature with an early abort of a generation loop.

Both of our attacks have been validated using a proof-of-concept implementation in Sage [Dev16]. The implementation is attached to this paper as supporting material for the reviewers' perusal.

4 Attack on $\mathbf{y_1}$

Due to the obvious similarities between the four instances of the Fiat-Shamir family that we choose to attack, we only give details of the attack on the BLISS scheme.

The first fault attack that we consider, which is also the more devastating one, targets the generation of the random "commitment" element $\mathbf{y}_1$, in both the original Lyubashevsky signature scheme and in BLISS. For simplicity's sake, we introduce the attack against BLISS in particular, but it works against other variants of Lyubashevsky signatures with almost no changes.

Intuitively, $\mathbf{y}_1$ should mask the secret key element $\mathbf{s}_1$ in the 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.

Basis: $\mathbf{B} \in \mathbb{Z}^{n \times n}$ of a n-dimensional lattice $\mathcal{L}$, standard deviation $\sigma > 0$, center $\mathbf{c} \in \mathbb{Z}^n$
$\mathbf{v}$ sampled in $\mathcal{D}_{\mathcal{L},\sigma,\mathbf{c}}$

function GaussianSampler($\mathbf{B}, \sigma, \mathbf{c}$)
  $\mathbf{v}_n \leftarrow \mathbf{0}$
  $\mathbf{c}_n \leftarrow \mathbf{c}$
  for $i \leftarrow \{n, \ldots, 1\}$ **do**
    $c'_i \leftarrow \langle c_i, \tilde{\mathbf{b}_i} \rangle / ||\tilde{\mathbf{b}_i}||^2$
    $\sigma'_i \leftarrow \sigma / ||\tilde{\mathbf{b}_i}||$
    $z_i \leftarrow$ SampleZ(σ'_i, c'_i)
    $\mathbf{c}_{i-1} \leftarrow z_i \mathbf{b}_i$ and $\mathbf{v}_{i-1} \leftarrow \mathbf{v}_i + z_i \mathbf{b}_i$
  end for
  return $\mathbf{v}_0$
end function

function KeyGen(n, q)
  $\sigma_{\mathbf{f}} = 1.17\sqrt{q/(2n)}$
  $\mathbf{f}, \mathbf{g} \leftarrow \mathcal{D}_{n,\sigma_{\mathbf{f}}}$
  Norm $\leftarrow \max \|(\mathbf{g}, -\mathbf{f})\|, \|(\frac{q\bar{\mathbf{f}}}{\mathbf{f}\bar{\mathbf{f}}+\mathbf{g}\bar{\mathbf{g}}}, \frac{q\bar{\mathbf{g}}}{\mathbf{f}\bar{\mathbf{f}}+\mathbf{g}\bar{\mathbf{g}}})\|)$
  if Norm$> 1.17\sqrt{q}$, go to step 3
  Using extended euclidean algo, compute $\rho_{\mathbf{f}}, \rho_{\mathbf{g}} \in \mathcal{R}$ and $R_{\mathbf{f}}, R_{\mathbf{g}} \in \mathbb{Z}$ s.t.
  – $\rho_{\mathbf{f}} \cdot \mathbf{f} = R_{\mathbf{f}} \bmod (x^n + 1)$
  – $\rho_{\mathbf{g}} \cdot \mathbf{g} = R_{\mathbf{g}} \bmod (x^n + 1)$
  if $\gcd(R_{\mathbf{f}}, R_{\mathbf{g}}) \neq 1$ or $\gcd(R_{\mathbf{f}}, q) \neq 1$, go to step 3
  Using extended euclidean algorithm, compute $u, v \in \mathbb{Z}$ s.t. $u \cdot R_{\mathbf{f}} + v \cdot R_{\mathbf{g}} = 1$
  $\mathbf{F} \leftarrow qv\rho_{\mathbf{g}}, \mathbf{G} \leftarrow -qu\rho_{\mathbf{f}}$
  while $k \neq 0$ **do**
    $k = \left\lfloor \frac{\mathbf{F}\cdot\bar{\mathbf{f}}+\mathbf{G}\cdot\bar{\mathbf{f}}}{\mathbf{f}\bar{\mathbf{f}}+\mathbf{g}\bar{\mathbf{g}}} \right\rceil \in \mathcal{R}$
    Reduce $\mathbf{F}$ and $\mathbf{G}$: $\mathbf{F} \leftarrow \mathbf{F} - k \cdot \mathbf{f}, \mathbf{G} \leftarrow \mathbf{G} - k \cdot \mathbf{g}$
  end while
  $\mathbf{h} = \mathbf{g} \cdot \mathbf{f}^{-1} \bmod q$
  return $sk = (\mathbf{f}, \mathbf{g}, \mathbf{F}, \mathbf{G})$, $pk = \mathbf{h}$ s.t. $\mathbf{f} \cdot \mathbf{h} = \mathbf{g} \bmod q$ and $\mathbf{fG} - \mathbf{gF} = q$
end function

Fig. 5. The function SampleZ(σ', c') samples a 1-dimensional Gaussian $\mathcal{D}_{\mathbb{Z},\sigma',c'}$.

Signing Key: $\mathbf{f}, \mathbf{g} \in \mathcal{R}_{q,1}$ where each coefficient is chosen uniformly and independently from {-1,0,1}
Verification Key: $\mathbf{h} \in \mathcal{R}_q$ where $\mathbf{g} = \mathbf{fh} \bmod q$
Random Oracle: $H : \{0,1\}^* \rightarrow \mathbb{Z}_q^n$

function Sign($\mu, \mathbf{B}$)
  $\mathbf{c} \leftarrow H(\mu) \in \mathbb{Z}_q^n$
  $(\mathbf{z}_1, \mathbf{z}_2) \leftarrow (\mathbf{c}, \mathbf{0}) -$ GaussianSampler($\mathbf{B}, \sigma, (\mathbf{c}, \mathbf{0})$) ▷ $\mathbf{z}_1 + \mathbf{z}_2 \cdot \mathbf{h} = \mathbf{t}$
  return $(\mathbf{z}_1, \mathbf{z}_2, \mathbf{c})$
end function

function Verify($\mu, \mathbf{z}_1, \mathbf{z}_2, \mathbf{c}, \mathbf{h}$)
  Accept iff $\mathbf{z}_1 + \mathbf{z}_2 \cdot \mathbf{h} = H(\mu)$ and $||(\mathbf{z}_1, \mathbf{z}_2)|| \leq 1.17\sqrt{q}$
end function

Fig. 6. IBE Sign [DLP14] signature scheme.

Concretely speaking, in BLISS, $\mathbf{y}_1 \in \mathcal{R}_q$ is a ring element generated according to a discrete Gaussian distribution[5], and that generation is typically carried out coefficient by coefficient in the polynomial rep-

[5] In the scheme of [GLP12], the distribution of each coefficient is uniform in some interval, but this doesn't affect our attack strategy at all.

resentation. Therefore, if we can use faults to cause an early termination of that generation process, we should obtain signatures in which the element $\mathbf{y}_1$ 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 generated from an element $\mathbf{y}_1$ of degree $m \ll n$. We get in particular 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 from that standpoint, we expect it to be the case with probability about $(1-1/q)^n$, which is over 95% for all proposed BLISS parameters. We thus get an equation of the form:

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

Thus, the vector $\mathbf{v} = \mathbf{c}^{-1}\mathbf{z}_1$ is very close to the sublattice of $\mathbb{Z}^n$ generated by $\mathbf{w}_i = \mathbf{c}^{-1}\mathbf{x}^i \bmod q$ for $i = 0, \ldots, m$ 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 I be a subset of $\{0, \ldots, n-1\}$, and $\varphi_I \colon \mathbb{Z}^n \to \mathbb{Z}^I$ be the projection $(u_i)_{0 \le i < n} \mapsto (u_i)_{i \in I}$. Then we also have that $\varphi_I(\mathbf{z}_1)$ is a close vector to the sublattice L_I of $\mathbb{Z}^I$ generated by $q\mathbb{Z}^I$ and the images under φ_I of the $\mathbf{w}_i$'s; and the difference should be $\varphi(\mathbf{s}_1)$.

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

The volume of L'_I is given by:

$$\mathrm{vol}(L'_I) = B \cdot \mathrm{vol}(L_I) = B \cdot \frac{\mathrm{vol}(q\mathbb{Z}^I)}{[L_I : q\mathbb{Z}^I]} = Bq^{\ell - r}$$

where ℓ is the cardinality of I and r is the rank of the family $\big(\varphi_I(\mathbf{w}_0), \ldots, \varphi_I(\mathbf{w}_m)\big)$ in $\mathbb{Z}_q^I$, which is at most $m+1$. Hence $\mathrm{vol}(L'_I) \ge Bq^{\ell-(m+1)}$, and the Gaussian heuristic predicts that the shortest vector should be of norm:

$$\lambda_I \approx \sqrt{\frac{\ell+1}{2\pi e}} \cdot \mathrm{vol}(L'_I)^{1/(\ell+1)} \gtrsim \sqrt{\frac{\ell+1}{2\pi e}} \cdot B^{1/(\ell+1)} q^{1-(m+2)/(\ell+1)}.$$

Thus, we expect that $\big(\varphi_I(\mathbf{s}_1), B\big)$ will actually be the shortest vector of L'_I provided that its norm is significantly smaller than this bound λ_I. Now $\varphi_I(\mathbf{s}_1)$ has roughly $\delta_1 \ell$ entries equal to ± 1, $\delta_2 \ell$ entries equal to ± 2 and the rest are zeroes; therefore, the norm of $\big(\varphi_I(\mathbf{s}_1), B\big)$ is around $\sqrt{(\delta_1 + 4\delta_2)\ell + B^2}$. Let us choose $B = \lceil \sqrt{\delta_1 + 4\delta_2} \rceil$. The condition for $\mathbf{s}_1$ to be the shortest vector L_I can thus be written as:

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

or equivalently:

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

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 $\varphi_I(\mathbf{s}_1)$ 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 or 60, 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 I 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 $\mathbf{s}_2 = \mathbf{a}_1\mathbf{s}_1/2 \bmod q$.

Remark 1. A variant of that attack which is possibly slightly simpler consists in observing that $\varphi_I(\mathbf{s}_1)$ should be the shortest vector in the lattice generated by L_I and $\varphi_I(\mathbf{v})$. The bound on the lattice dimension becomes essentially the same as (2). The drawback of that approach, however, is that we obtain each $\varphi_I(\mathbf{s}_1)$ up to sign, and so one needs to use overlapping subsets I 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 $\mathbf{y}_1$ 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 [DDLL13], and even smaller with the improved analysis of BLISS–B [Duc14].

Remark 3. Finally, we note that in certain hardware settings, fault injection may yield a faulty value of $\mathbf{y}_1$ in which all coefficients upwards of a certain degree bound are non zero but equal to a common constant (see the discussion in Section 6.3). Our attack adapts to that setting in a straightforward way: that simply means that $\mathbf{y}_1$ is a linear combination of the $\mathbf{x}^i$ for small i and of the all-one vector $(1, \ldots, 1)$, so it suffices to add that vector to the set of lattice generators.

5 Attack on the GPV scheme

Our second attack targets the practical hash-and-sign signature scheme of Ducas, Lyubashevsky and Prest [SI14], 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.

5.1 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 $m \leq n$ iterations (in the first half of the loop), the resulting signature simply has the form:

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

Any such faulty signature is, in particular, in the lattice L of rank m generated by the vectors $\mathbf{x}^{m-i}\mathbf{F}$, $i = 1, \ldots, m$, 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 ℓ is large enough (slightly more than m is sufficient; see §5.2 below for an analysis of success probability depending on ℓ), 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 $\pm\mathbf{x}^{m-i}\mathbf{F}$, $i = 1, \ldots, m$, 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 $\mathbf{F}$ 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, $\mathbf{G}$ is linked to $\mathbf{F}$ via the public key polynomial $\mathbf{h}$: $\mathbf{G} = \mathbf{F} \cdot \mathbf{h} \bmod q$, so we obtain it directly. On the other hand, the basis completion algorithm of Hoffstein et al. [HHP+03] 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 $(\mathbf{f}, \mathbf{g})$), but applying [HHP+03, 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 ζ 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 don't 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.

5.2 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 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 ℓ 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 ℓ random 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:

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

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 [MRW11] (see also [FW14]), 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. $p_2 \approx 71\%$, $p_3 \approx 86\%$).

6 Implementation of the faults

Once afain, due to the obvious similarities between the four instances of the Fiat-Shamir family that we choose to attack, we only give details of the attack on the BLISS scheme. Even though the second attack on the GPV scheme looks different in its exploitation, the underlying fault introduced is strictly identical and as such we let the reader refers to the following description.

In this section we investigate how an attacker may obtain helpful faulty signatures for the proposed attacks. We base our discussion on two available implementations of BLISS signature namely the software implementation from Ducas and Lepoint [DL] and the FPGA implementation by Pöppelmann *et al.* [PDG14]. The former one is based on `/dev/urandom` and SHA 512 while the later performs sampling based on TRIVIUM instances and uses KECCAK hash function as random oracle. Notice that discussion on the hardware implementation is also valid for the implementation of [GLP12] since both share some common components and architecture that we exploit (for instance BRAM storage).

We emphasize the fact that those 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 of dedicated protections against faults attacks (when attackers have such abilities).

6.1 Classical fault models

Faults during a computation may be induced by different means as a laser beam shot, electromagnetic injection, under-powering, clock/power glitches... 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 (resp. 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 from 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 overdone are actually obtained during lab experiments. One example is the work of Ordas et al. at CARDIS 2014 [OGT+14] showing that with finely tuned EM probes it is possible to induce a single-bit fault (bit-set or bit-reset).

In the next subsections we discuss which fault models[6] may lead to faulty signatures relevant with respect to the attacks presented in this paper. We did not investigate clock glitches or under-powering which induce violation of the setup time and which actual side-effects are implementation and compilation-dependent (with large ranges of possible parameters to test). Nevertheless, they may not be overseen in the evaluation of a chip since they may also lead to the generation of relevant faulty signatures.

6.2 Fault attacks on software implementations

Polynomial $\mathbf{y}_1$ can be generated using a loop over the n coefficients. This is, again, how the implementation in [DL] is made: a loop is constructing polynomials $\mathbf{y}_1$ and $\mathbf{y}_2$ one coefficient at a time using a Gaussian sampler (function `Sign::signMessage`).

The condition To perform the attack rather few restrictive since we only require $\mathbf{y}_1$ to have at most (roughly) a quarter of unknown coefficients. Such result can be obtain by going out the loop after a few iterations. Again, a random fault on the loop counter or skipping the jump operation will lead to such result.

Notice here that it is less trivial here to decide whether a faulty signature will be helpful or not. Hopefully, the timing precision is much less important here 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 decades of loop iterations. Moreover, we may use side-channel analysis to detect the loop iteration pattern to trigger the fault injection. Such pattern is likely to be detected after much less than 50 iterations and thus it seems that the synchronization here will be relatively easy.

To conclude, this attack also seems to be a real threat and even more than the attack on $\mathbf{c}$ since synchronization is ease in this case by the loose condition on the number of known coefficients in y_1.

6.3 Fault attacks on hardware implementations

Generation of polynomial $\mathbf{y}_1$ requires n random coefficients. It is very unlikely that all these coefficient are obtained at the same time (n is too large) thus $\mathbf{y}_1$ generation will 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 (here the bliss processor) and keep part

[6] We only focus on single fault attacks here.

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. The large number of unknown coefficients handled by the attack again helps the attacker 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 memory cells previous/initial value.

There is a second way of performing the fault injection here. The value of $\mathbf{y}_1$ has to be stored somehow until the computation of $\mathbf{z}_1$ (close to the end of the signature generation). In the example implementation a BRAM is used. We may fault BRAM access 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 of $\mathbf{y}_1$ used to compute $\mathbf{u}$ 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 we know that 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 Section 4.

Again a large time window is given to the attacker due to sequential read induced by the size of $\mathbf{y}_1$.

The BRAM storage of $\mathbf{y}_1$ helps here 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 seems to be possible according to [OGT+14] — and the `rstram` signal localization[7].

7 Conclusion and possible countermeasures

We have shown that unprotected implementations of the four lattice-based signature schemes that we considered are vulnerable to fault attacks, in certain 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, 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 here since they rely on processing signatures computed from polynomials that are too sparse. Let investigate the in-depth described attack on BLISS, and the generation of $\mathbf{y}_1$. The number of non-zero coefficients in that polynomial is not fixed by the sampling algorithm, and it can take several distinct values with non-negligible probability. Therefore, requiring that $\mathbf{y}_1$ has a specified number of non-zero coefficients affects the distribution of $\mathbf{z}_1$ and could thus cause invalidate the security of the scheme. One could, on the other hand, check that the top $\varepsilon \cdot n$ coefficients of $\mathbf{y}_1$ are not all zero for some small constant $\varepsilon > 0$, say $\varepsilon = 1/16$; indeed, the probability that they all vanish is roughly:

$$\left(\frac{1}{\sigma\sqrt{2\pi}}\right)^{\varepsilon n}$$

which is exponentially small. Thus, the resulting distribution of $\mathbf{y}_1$ 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 greater than $(1-\varepsilon)n$, which makes it impractical.

Another possible countermeasure is to generate the coefficients of $\mathbf{y}_1$ in random order instead of iterating from low to high degree or some other fixed pattern. This prevents our attack since, even if a faults causes the loop to terminate early, the attacker no longer knows which subset of the $\mathbf{c}^{-1}\mathbf{x}^i$ the vector $\mathbf{c}^{-1}\mathbf{z}_1 - \mathbf{s}_1$ is a linear combination of.

[7] Since $\mathbf{y}_1$ is not directly outputted checking if the attack actually worked is a bit more 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 then for a $\mathbf{z}_1$ corresponding to a dense $\mathbf{y}_1$.

References

ABBD15. Erdem Alkim, Nina Bindel, Johannes A. Buchmann, and Özgür Dagdelen. TESLA: tightly-secure efficient signatures from standard lattices. *IACR Cryptology ePrint Archive*, 2015:755, 2015.

Ajt96. Miklós Ajtai. Generating hard instances of lattice problems (extended abstract). In Gary L. Miller, editor, *STOC*, pages 99–108. ACM, 1996.

BDL01. Dan Boneh, Richard A. DeMillo, and Richard J. Lipton. On the importance of eliminating errors in cryptographic computations. *J. Cryptology*, 14(2):101–119, 2001.

Ben14. Josh Benaloh, editor. *Topics in Cryptology - CT-RSA 2014 - The Cryptographer's Track at the RSA Conference 2014, San Francisco, CA, USA, February 25-28, 2014. Proceedings*, volume 8366 of *Lecture Notes in Computer Science*. Springer, 2014.

BMM00. Ingrid Biehl, Bernd Meyer, and Volker Müller. Differential fault attacks on elliptic curve cryptosystems. In Mihir Bellare, editor, *CRYPTO*, volume 1880 of *LNCS*, pages 131–146. Springer, 2000.

CJL+16. Lily Chen, Stephen Jordan, Yi-Kai Liu, Dustin Moody, Rene Peralta, Ray Perlner, and Daniel Smith-Tone. Report on post-quantum cryptography. Technical report, National Institute of Standards and Technology, February 2016. Available at http://csrc.nist.gov/publications/drafts/nistir-8105/nistir_8105_draft.pdf.

DBG+14. Özgür Dagdelen, Rachid El Bansarkhani, Florian Göpfert, Tim Güneysu, Tobias Oder, Thomas Pöppelmann, Ana Helena Sánchez, and Peter Schwabe. High-speed signatures from standard lattices. In Diego F. Aranha and Alfred Menezes, editors, *Progress in Cryptology - LATINCRYPT 2014 - Third International Conference on Cryptology and Information Security in Latin America, Florianópolis, Brazil, September 17-19, 2014, Revised Selected Papers*, volume 8895 of *Lecture Notes in Computer Science*, pages 84–103. Springer, 2014.

DBI+15. 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*, December 2015.

DDLL13. Léo Ducas, Alain Durmus, Tancrède Lepoint, and Vadim Lyubashevsky. Lattice signatures and bimodal gaussians. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference. Proceedings, Part I*, volume 8042 of *LNCS*, pages 40–56. Springer, 2013.

Dev16. The Sage Developers. *Sage Mathematics Software (Version 7.0)*, 2016. http://www.sagemath.org.

DL. Léo Ducas and Tancrède Lepoint. A Proof-of-concept Implementation of BLISS. Available under the CeCILL License at http://bliss.di.ens.fr.

DLP14. Léo Ducas, Vadim Lyubashevsky, and Thomas Prest. Efficient identity-based encryption over NTRU lattices. In Sarkar and Iwata [SI14], pages 22–41.

DN12. Léo Ducas and Phong Q. Nguyen. Learning a zonotope and more: Cryptanalysis of NTRUSign countermeasures. In Xiaoyun Wang and Kazue Sako, editors, *ASIACRYPT*, volume 7658 of *LNCS*, pages 433–450. Springer, 2012.

Duc14. Léo Ducas. Accelerating BLISS: the geometry of ternary polynomials. Cryptology ePrint Archive, Report 2014/874, 2014. http://eprint.iacr.org/.

FS86. Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *CRYPTO*, volume 263 of *LNCS*, pages 186–194. Springer, 1986.

FW14. Felix Fontein and Pawel Wocjan. On the probability of generating a lattice. *Journal of Symbolic Computation*, 64:3–15, 2014.

GJSS01. Craig Gentry, Jakob Jonsson, Jacques Stern, and Michael Szydlo. Cryptanalysis of the NTRU signature scheme (NSS) from Eurocrypt 2001. In Colin Boyd, editor, *ASIACRYPT*, volume 2248 of *LNCS*, pages 1–20. Springer, 2001.

GLP12. Tim Güneysu, Vadim Lyubashevsky, and Thomas Pöppelmann. Practical lattice-based cryptography: A signature scheme for embedded systems. In Emmanuel Prouff and Patrick Schaumont, editors, *CHES*, volume 7428 of *LNCS*, pages 530–547. Springer, 2012.

GPV08. Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In Cynthia Dwork, editor, *STOC*, pages 197–206. ACM, 2008.

GS02. Craig Gentry and Michael Szydlo. Cryptanalysis of the revised NTRU signature scheme. In Lars R. Knudsen, editor, *EUROCRYPT*, volume 2332 of *LNCS*, pages 299–320. Springer, 2002.

HHP+03. Jeffrey Hoffstein, Nick Howgrave-Graham, Jill Pipher, Joseph H. Silverman, and William Whyte. NTRUSign: Digital signatures using the NTRU lattice. In Marc Joye, editor, *CT-RSA*, volume 2612 of *Lecture Notes in Computer Science*, pages 122–140. Springer, 2003.

HPS+14. Jeffrey Hoffstein, Jill Pipher, John M. Schanck, Joseph H. Silverman, and William Whyte. Practical signatures from the partial fourier recovery problem. In Ioana Boureanu, Philippe Owesarski, and Serge Vaudenay, editors, *Applied Cryptography and Network Security - 12th International Conference, ACNS 2014, Lausanne, Switzerland, June 10-13, 2014. Proceedings*, volume 8479 of *Lecture Notes in Computer Science*, pages 476–493. Springer, 2014.

KY12. Abdel Alim Kamal and Amr M. Youssef. Fault analysis of the NTRUSign digital signature scheme. *Cryptography and Communications*, 4(2):131–144, 2012.

LM06. Vadim Lyubashevsky and Daniele Micciancio. Generalized compact knapsacks are collision resistant. In Michele Bugliesi, Bart Preneel, Vladimiro Sassone, and Ingo Wegener, editors, *ICALP*, volume 4052 of *LNCS*, pages 144–155. Springer, 2006.

LM08. Vadim Lyubashevsky and Daniele Micciancio. Asymptotically efficient lattice-based digital signatures. In Ran Canetti, editor, *TCC*, volume 4948 of *LNCS*, pages 37–54. Springer, 2008.

LPR13. Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. *J. ACM*, 60(6):43, 2013.

Lyu09. Vadim Lyubashevsky. Fiat–Shamir with aborts: Applications to lattice and factoring-based signatures. In Mitsuru Matsui, editor, *ASIACRYPT*, volume 5912 of *LNCS*, pages 598–616. Springer, 2009.

Lyu12. Vadim Lyubashevsky. Lattice signatures without trapdoors. In David Pointcheval and Thomas Johansson, editors, *EUROCRYPT*, volume 7237 of *LNCS*, pages 738–755. Springer, 2012.

MRW11. Gérard Maze, Joachim Rosenthal, and Urs Wagner. Natural density of rectangular unimodular integer matrices. *Linear Algebra and its Applications*, 434(5):1319–1324, 2011.

NNTW05. David Naccache, Phong Q. Nguyen, Michael Tunstall, and Claire Whelan. Experimenting with faults, lattices and the DSA. In Serge Vaudenay, editor, *PKC*, volume 3386 of *LNCS*, pages 16–28. Springer, 2005.

NR09. Phong Q. Nguyen and Oded Regev. Learning a parallelepiped: Cryptanalysis of GGH and NTRU signatures. *J. Cryptology*, 22(2):139–160, 2009.

NSA16. CNSA Suite and quantum computing FAQ. Technical report, National Security Agency, January 2016. Available at https://www.iad.gov/iad/library/ia-guidance/ia-solutions-for-classified/algorithm-guidance/cnsa-suite-and-quantum-computing-faq.cfm.

OGT+14. Sébastien Ordas, Ludovic Guillaume-Sage, Karim Tobich, Jean-Max Dutertre, and Philippe Maurine. Evidence of a larger EM-induced fault model. In Marc Joye and Amir Moradi, editors, *CARDIS*, volume 8968 of *LNCS*, pages 245–259. Springer, 2014.

PDG14. Thomas Pöppelmann, Léo Ducas, and Tim Güneysu. Enhanced lattice-based signatures on reconfigurable hardware. In Lejla Batina and Matthew Robshaw, editors, *CHES*, volume 8731 of *LNCS*, pages 353–370. Springer, 2014.

Pei15. Chris Peikert. A decade of lattice cryptography. Cryptology ePrint Archive, Report 2015/939, 2015. http://eprint.iacr.org/.

PR06. Chris Peikert and Alon Rosen. Efficient collision-resistant hashing from worst-case assumptions on cyclic lattices. In Shai Halevi and Tal Rabin, editors, *TCC*, volume 3876 of *LNCS*, pages 145–166. Springer, 2006.

PS96. David Pointcheval and Jacques Stern. Security proofs for signature schemes. In Ueli M. Maurer, editor, *Advances in Cryptology - EUROCRYPT '96, International Conference on the Theory and Application of Cryptographic Techniques, Saragossa, Spain, May 12-16, 1996, Proceeding*, volume 1070 of *Lecture Notes in Computer Science*, pages 387–398. Springer, 1996.

PV06. Dan Page and Frederik Vercauteren. A fault attack on pairing-based cryptography. *IEEE Trans. Computers*, 55(9):1075–1080, 2006.

SI14. Palash Sarkar and Tetsu Iwata, editors. *Advances in Cryptology - ASIACRYPT 2014 - 20th International Conference on the Theory and Application of Cryptology and Information Security, Kaoshiung, Taiwan, R.O.C., December 7-11, 2014, Proceedings, Part II*, volume 8874 of *Lecture Notes in Computer Science*. Springer, 2014.

TE15. Mostafa Taha and Thomas Eisenbarth. Implementation attacks on post-quantum cryptographic schemes. In Eisa A. Aleisa, editor, *ICACC*. IEEE Social Implications of Technology Society, 2015.

A Listings of POC code

```
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^n+1))

sampler = DiscreteGaussianDistributionIntegerSampler(sigma=sigma, algorithm='
    uniform+table')

def s1gen():
    s1vec=[0]*n
    d1=ceil(delta1*n)
    d2=ceil(delta2*n)

    while d1>0:
        i=randint(0,n-1)
        if s1vec[i]==0:
            s1vec[i]=(-1)^randint(0,1)
            d1-=1

    while d2>0:
        i=randint(0,n-1)
        if s1vec[i]==0:
            s1vec[i]=2*(-1)^randint(0,1)
            d2-=1

    return sum([s1vec[i]*xx^i for i in range(n)])


def faultyz1gen(s1,d):
    #y1 should be generated with gaussian coefficients, but let's do it
    #uniformly for now
    #y1=sum([randint(-2*sigma,2*sigma)*xx^i for i in range(d)])
    y1=sum([sampler()*xx^i for i in range(d)])

    #c is a random binary polynomial of weight kappa
    dc=kappa
    cvec=[0]*n
    while dc>0:
        i=randint(0,n-1)
        if cvec[i]==0:
            cvec[i]=1
            dc-=1

```

```
    c=sum([cvec[i]*xx^i for i in range(n)])
    z1=y1+c*s1

    return (c,z1)

def faultattack(d,e):
    s1=s1gen()
    (c,z1)=faultyz1gen(s1,d)

    try:
        cinv=1/Rq(c.lift())
    except ZeroDivisionError:
        print "c not invertible"
        return

    """
    Try to recover the first e coefficients of s1
    (of course, if we succeed, we should succeed for *all* sets of
    e coefficients of s1, so we can recover the whole secret key).
    """

    latvec=[(cinv*xxx^i).lift().list()[:e] for i in range(d)]
    latvec=[(cinv*Rq(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))).LLL()
    v=M[d+e]
    v=v*(2*q/v[-1])

    print "Recovered vector:", v[:-1]
    print "Truncated key:", s1.lift().list()[:e]

    return s1,c,z1,M
```

Listing 1.1. 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))

K=QuotientRing(QQ[x], QQ[x].ideal(x^n+1))

```

```
def norml2(l):
    return sqrt(sum([RR(x)^2 for x in l]))

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))

        f2 = q*fbar / (f*fbar + g*gbar)
        g2 = q*gbar / (f*fbar + g*gbar)

        norm = max(norml2(f.list() + g.list()), \
                   norml2(f2.list() + g2.list()))

        if norm > sigma:
            continue

        Rf, rhof, _ = xgcd(f.lift(), x^n+1)
        Rg, rhog, _ = xgcd(g.lift(), x^n+1)
        gg, u,    v = xgcd(Rf, Rg)

        if gg == 1 and gcd(Rf,q) == 1:
            break

    F = q*v*rhog
    G =-q*u*rhof

    while True:
        k = (F*fbar + G*gbar) / (f*fbar + g*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:
            break

        F-= k*f
        G-= k*g

    h = Rq(g)/Rq(f)

    B = block_matrix( [[anticirculant(g), anticirculant(-f)], \
                       [anticirculant(G), anticirculant(-F)]] )
    return h,f,g,B

def rndvec(v):
    return [x if 2*x<q else x-q for x in v]

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):
        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:
            print z,RR(s),RR(k)

        c = c - z*B.row(i)
        v = v + z*B.row(i)


    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.
    One should need m>d in general?
    """
    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()
    else:
        print "Trying to reduce with BKZ-%d" % bkz
        M = Matrix(ZZ,sigs).BKZ(block_size=bkz)

    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)])
    print "P_i/F =", [K(Pi)/K(F) for Pi in P]

    return P
```

Listing 1.2. Attack on GPV scheme