Cryptanalysis of a code-based one-time signature

Jean-Christophe Deneuville and Philippe Gaborit

[1] INSA-CVL, LIFO Orléans, France
jean-christophe.deneuville@insa-cvl.fr
[2] University of Limoges, France
philippe.gaborit@unilim.fr

Abstract. In 2012, Lyubashevsky introduced a new framework for building lattice-based signature schemes without resorting to any trapdoor (such as GPV [6] or NTRU [8]). The idea is to sample a set of short lattice elements and construct the public key as a Short Integer Solution (SIS for short) instance. Signatures are obtained using a small subset sum of the secret key, hidden by a (large) gaussian mask. (Information leakage is dealt with using rejection sampling.) In this paper, we show that this framework cannot be adapted to coding theory. In particular, we show that any code-based signature obtained through a direct translation from the lattice setting is doomed to fail, due to an inherent difference between bounds in Hamming and Euclidean metrics. The attack consists in rewriting a signature as a noisy syndrome decoding problem, which can be handled efficiently using the extended bit flipping decoding algorithm. We illustrate our results by breaking Persichetti's one-time signature scheme built upon this approach [13]: using a single signature, we recover the secret (signing) key in about the same amount of time as required for a couple of signature verifications.

Keywords: Post-Quantum Cryptography, Coding Theory, Efficient, Signature, One-time, Cryptanalysis

1 Introduction

Building efficient and secure full-time (stateless) signature schemes from coding theory assumptions is a long standing open problem. Few years ago, Lyubashevsky proposed a new method for obtaining digital signatures from lattice assumptions, that does not require the use of a trapdoor [9]. This method follows the baseline of Pointcheval-Stern [14]. The construction works by sampling relatively short lattice vectors, used as the secret key. The public key is a Short Integer Solution (SIS) instance. To produce a digital signature, the signer commits a masking value, receives a challenge depending on the message to sign and the committed value, and computes a combination of the challenge and the secret key, hidden by the committed mask (the scheme is recalled in more details in Sec. 2.3). The verifier accepts the signature if it satisfies some property (small euclidean norm, or hamming weight) and the verifier did use the challenge.

Recently, Persichetti proposed an efficient adaptation of Lyubashevsky's framework to coding theory. The scheme works similarly to Lyubashevsky's, with two major differences: the underlying hard problem is different and there is no rejection sampling step. The underlying problem is the renowned Syndrome Decoding (SD) problem, which has been proved NP-hard [2]. The secret key is a vector of small Hamming weight, and the public key is the syndrome of this vector by a public (random) parity-check matrix. According to the author, rejection sampling seems useless since the signature only depends on the message, the commitment and the challenge.

One of the most technical aspects in the design of a signature scheme is to make the signature distribution statistically independent from the secret key. This allows (by programming the random in the security reduction) the forger to produce valid signatures without knowing the secret key, which can then be used to solve the underlying hard problem. This technicality provides guidance for the choice of the parameters, especially for the Hamming weight (or ℓ_1 norm for Lyubashevsky) of the challenge. Indeed, in order for SD problem to admit a unique solution, the weight of the signature must be below the Gilbert-Varshamov bound. In the meantime, the weight of the secret key should be big enough in order not to be exhibited easily. This implies that the challenge should have an exceptionally low weight for the signature scheme to work. This is indeed the case for all the proposed parameters: the "biggest" (least sparse) challenge has weight $\delta = 10$ for length $n = 4801$.

From a cryptanalytic point of view, a signature can be rewritten as a noisy decoding problem with known generator matrix: the cyclic matrix obtained through the challenge. Roughly speaking, signatures can be viewed as McEliece encryptions of the secret key under public *unscrambled* sparse generator matrix. Using such an approach, the matrix corresponds to a Low/Moderate Density Parity-Check (LDPC / MDPC) code. We show that it is possible to use the extended Bit Flipping algorithm [5, 10, 1] to decrypt these ciphertexts, hence retrieving both the secret key and one time randomness from a single signature, for all the proposed parameters.

Conceptually speaking, the cryptanalysis is possible because the Hamming weight of the challenge is way too small. Increasing this weight would require to lower the weight of the secret key, opening the door for other small weight codeword finding attacks.

1.1 Contributions

The contributions of this work are two-fold:

– We show that a direct translation of Lyubashevsky's framework to build signatures without trapdoors from lattice assumptions to coding theory assumptions can only yield insecure signatures. It was suspected [13] that such signatures could *not* reach full-time security due to a statistical bias of the information leaked by the signature. We show that the information leakage is —by construction— so important that such signatures cannot even be one-time secure.
– As to illustrate our claims, the second contribution of this paper is a full cryptanalysis of all the parameters of Persichetti's OTS scheme based upon an adaptation of Lyubashevsky's framework. As an example, our attack recovers the signing key of the most secure instance ($n = 9857$, 128 bits of security) in $\approx$ 450ms (versus 100ms for signature verification).

1.2 Techniques

To conduct a full-cryptanalysis of efficient code based signatures without trapdoors, we begin by formulating the signature cryptanalysis as a decoding problem. The decoding involves a relatively sparse generator matrix (similar to LDPC or MDPC codes). To do so, the signature is split into two halves.

The secret key $\boldsymbol{x} = (\boldsymbol{x}_0, \boldsymbol{x}_1)$ has a global weight of w_1, meaning that $wt(\boldsymbol{x}_0) + wt(\boldsymbol{x}_1) = w_1$. But for the cryptanalysis, no hint is provided about the weight of each part, and the same holds for the one-time randomness $\boldsymbol{y} = (\boldsymbol{y}_0, \boldsymbol{y}_1)$ used for signing. Therefore, we relax the requirements for decoding the first instance, and reverberate on the second instance using the solutions of the first problem.

Once the instances are set up, the extended Bit Flipping algorithm is used to efficiently solve both instances.

1.3 Organization of the paper

The remainder of this paper is organized as follows: Section 2 introduces the notations used throughout this work as long as relevant notions in coding theory and Lyubashevsky's signature scheme. Section 3 presents a general adaptation of Lyubashevsky's framework to coding theory, not restricted to specific (quasi-cyclic) codes. Section 4 is devoted to expressing key recovery from a single signature as a decoding problem, and arguing that this problem is efficiently solvable. A general purpose algorithm to solve the latter problem is presented in Section 5. We finally instantiate the key recovery with Persichetti's OTS in Section 6, presenting a full cryptanalysis, before concluding in Section 7.

2 Preliminaries

2.1 Notations

Throughout the paper, $\mathbb{F}_2$ denotes the binary field. Vectors (resp. matrices) will be represented in lower-case (resp. upper-case) bold letters. A vector $\boldsymbol{u} = (u_0, \dots, u_{n-1}) \in \mathbb{F}_2^n$ will be interchangeably seen as a vector or polynomial in $\mathbb{F}_2[x]/\langle x^n - 1\rangle$. Hence for $\boldsymbol{u}, \boldsymbol{v} \in \mathbb{F}_2^n$, $\boldsymbol{w} = \boldsymbol{u}\boldsymbol{v}$ denotes the vector such that:

$$w_k = \sum_{i+j=k} u_i v_j x^k, \text{ for } k \in \{0, \dots, n-1\}.$$

Finally, the set of binary vectors of length n and weight exactly w is denoted $\mathcal{S}_w^n(\mathbb{F}_2)$.

2.2 Coding theory

We now recall some basic definitions and facts about coding theory that will be helpful for the comprehension of Persichetti's OTS and its cryptanalysis.

Definition 1 (Parity-check matrix). *Let n, k be integers. The parity-check matrix of an $[n, k]$ linear code $\mathcal{C}$ is a matrix $\boldsymbol{H} \in \mathbb{F}_2^{(n-k)\times n}$ that generates the dual code $\mathcal{C}^\perp$. Formally, if $\boldsymbol{G} \in \mathbb{F}_2^{n\times k}$ is a generator matrix of $\mathcal{C}$, then $\boldsymbol{H}$ satisfies $\boldsymbol{G}\boldsymbol{H}^\top = \boldsymbol{0}$.*

Definition 2 (Syndrome Decoding problem). *Let $\boldsymbol{H} \in \mathbb{F}_2^{(n-k)\times n}$ be a parity-check matrix of some $[n, k]$ linear code over $\mathbb{F}_2$, and $s \in \mathbb{F}_2^{n-k}$ a syndrome, and w an integer. The* Syndrome Decoding problem *asks to find a vector $\boldsymbol{e} \in \mathbb{F}_2^n$ of weight less than or equal to w such that $\boldsymbol{s} = \boldsymbol{H}\boldsymbol{e}^\top$.*

The SD problem has been proved to be NP-hard [2]. Assuming a solution to the SD problem exists, the target weight w determines whether the solution is unique or not. This property is captured through the well-known Gilbert-Varshamov (GV) bound [7, 15].

Definition 3 (Gilbert-Varshamov bound). *Let $\mathcal{C}$ be an $[n, k]$ linear code over $\mathbb{F}_2$. The* Gilbert-Varshamov bound *d_{GV} is the maximum value d such that*

$$\sum_{i=0}^{d-1} \binom{n}{i} (q-1)^i \leq q^{n-k}.$$

When the target weight w of an instance of the SD problem is below the GV bound, the solution is guaranteed to be unique. This property is useful for the design of Persichetti's OTS. The GV bound is plotted (together with Hamming and Singleton bounds) in Fig. 1.

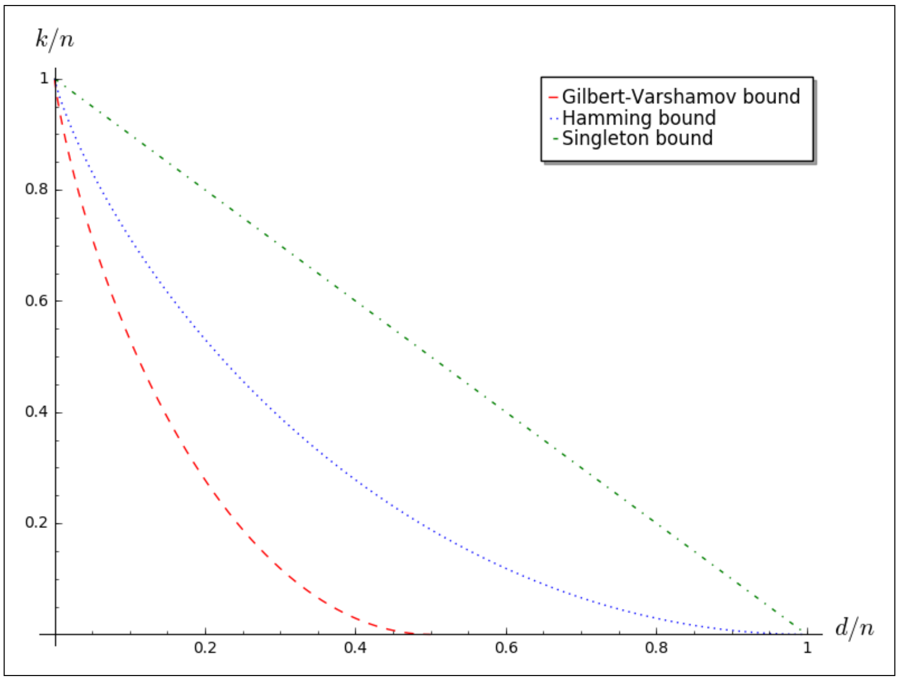

Fig. 1. Gilbert-Varshamov, Hamming and Singleton bounds.

2.3 Lattice signatures without trapdoors

In 2012, Lyubashevsky proposed a new approach for building lattice-based signatures without trapdoors [9]. Contrarily to NTRUSign [8] which embeds very short vectors in the secret key, and GPV [6] which uses gaussian sampling to avoid information leakage when generating the public key from the secret key, Lyubashevsky's keys are an SIS instance, an analogue to the syndrome decoding problem in the lattice setting.

We now recall Lyubashevsky's signature scheme. We keep the description in its general form but as mentioned by the author, key sizes can be shrunk by a factor k using more structured matrices and relying on the ring version of the SIS problem. Private and public keys are respectively uniformly random matrices $\boldsymbol{S} \in \{-d, \ldots, 0, \ldots, d\}^{m \times k}$ and $\boldsymbol{A} \in \mathbb{F}_2^{n \times m}$ ($\boldsymbol{T} = \boldsymbol{AS}$ also belongs to pk) and the signature process invokes a hash function $H : \{0,1\}^* \to \left\{\boldsymbol{v} : \boldsymbol{v} \in \{0,1\}^k, \|\boldsymbol{v}\|_1 \leq \kappa\right\}$. A signature $(\boldsymbol{z}, \boldsymbol{c})$ of a message $\boldsymbol{m}$ corresponds to a combination of the secret key and the hash of this message, shifted by a committed value also used in the hash function. The entire scheme is depicted in Fig. 2.

Algorithm 1 KeyGen(n, m, k, q, d)

```
Input: $n, m, k, q, d \in \mathbb{Z}$
Output: $(\mathsf{pk}, \mathsf{sk})$ with $\mathsf{pk} = (\boldsymbol{A}, \boldsymbol{T}) \in \mathbb{F}_2^{n \times m} \times \mathbb{F}_2^{n \times k}$ and $\mathsf{sk} = \boldsymbol{S} \in \mathbb{F}_2^{m \times k}$
1: $\boldsymbol{S} \xleftarrow{\$} \{-d, \ldots, 0, \ldots, d\}^{m \times k}$
2: $\boldsymbol{A} \xleftarrow{\$} \mathbb{F}_2^{n \times m}$
3: $\boldsymbol{T} \leftarrow \boldsymbol{AS}$
4: return $(\mathsf{pk} = (\boldsymbol{A}, \boldsymbol{T}), \mathsf{sk} = \boldsymbol{S})$
```

Algorithm 2 Sign$(\mathsf{pk}, \mathsf{sk}, \boldsymbol{m})$

```
Input: Public and private keys, message $\boldsymbol{m} \in \{0,1\}^*$ to be signed
Output: Signature $(\boldsymbol{z}, \boldsymbol{c}) \in \mathbb{F}_2^m \times \mathbb{F}_2^k$ of message $\boldsymbol{m}$
1: $\boldsymbol{y} \xleftarrow{\$} D_\sigma^m$
2: $\boldsymbol{c} \leftarrow H(\boldsymbol{Ay}, \boldsymbol{m})$
3: $\boldsymbol{z} \leftarrow \boldsymbol{Sc} + \boldsymbol{y}$
4: return $(\boldsymbol{z}, \boldsymbol{c})$ with probability $\min\left(\frac{D_\sigma^m(\boldsymbol{z})}{M \cdot D_{\boldsymbol{Sc},\sigma}^m(\boldsymbol{z})}, 1\right)$
```

Algorithm 3 Verify$(\mathsf{pk}, (\boldsymbol{z}, \boldsymbol{c}), \boldsymbol{m})$

```
Input: Public key, message $\boldsymbol{m}$, and the signature $(\boldsymbol{z}, \boldsymbol{c})$ to verify
Output: Accept if $(\boldsymbol{z}, \boldsymbol{c})$ is a valid signature of $\boldsymbol{m}$, Reject otherwise
1:
2: if $H(\boldsymbol{Az} - \boldsymbol{Tc}, \boldsymbol{m}) = \boldsymbol{c}$ and $\|\boldsymbol{z}\| \leq \eta\sigma\sqrt{m}$ then
3:     return Accept
4: else
5:     return Reject
```

Fig. 2. Lyubashevsky's (full-time) lattice-based signature scheme.

3 Code-based signatures without trapdoors

In this Section we describe two general code-based adaptations of Lyubashevsky's signature scheme, not restricted to quasi-cyclic codes: a vectorial one, similar to Persichetti's OTS, and a matrix version. The aim of this description is to demonstrate that the weakness of such an adaptation comes from the vectorial version, *not* from the additional structure added for efficiency. We discuss about the relative (UN)security of the matrix adaptation at the end of this section.

Both adaptations require a hash function that outputs pseudorandom words of length m and small weight δ. Formally, we denote such a function $\mathcal{H}_m : \{0,1\}^* \mapsto \mathcal{S}_\delta^m(\mathbb{F}_{\mathbb{F}_2})$.

3.1 Efficient one-time signatures from codes (not necessarily quasi-cyclic)

This subsection is devoted to the presentation of the generalization of Persichetti's OTS to not just quasi-cyclic codes. In this vectorial version, the secret key is a vector $\boldsymbol{x}$ of small weight w_1, and the public key is a random parity-check matrix $\boldsymbol{H}$ together with the syndrome of the secret key:

$\boldsymbol{s_x} = \boldsymbol{H}\boldsymbol{x}^\top$. In Persichetti's proposal, the matrix $\boldsymbol{H}$ admits a quasi-cyclic systematic representation: $\boldsymbol{H} = (\mathbf{1}\ \ \boldsymbol{h})$, allowing to reduce the pk size.

Algorithm 4 KeyGen($\mathsf{params} = (n, k, w_1, w_2, \delta)$)

```
Input: Public parameters $\mathsf{params} = (n, k, w_1, w_2, \delta)$
Output: $(\mathsf{pk}, \mathsf{sk})$ with $\mathsf{pk} = (\boldsymbol{H}, \boldsymbol{s_x}) \in \mathbb{F}_2^{(n-k)\times n} \times \mathbb{F}_2^{(n-k)}$ and $\mathsf{sk} = \boldsymbol{x} \in \mathbb{F}_2^n$
1: $\boldsymbol{x} \xleftarrow{\$} \mathcal{S}_{w_1}^n(\mathbb{F}_2) \subset \mathbb{F}_2^n$
2: $\boldsymbol{H} \xleftarrow{\$} \mathbb{F}_2^{(n-k)\times n}$
3: $\boldsymbol{s_x} \leftarrow \boldsymbol{H}\boldsymbol{x}^\top \in \mathbb{F}_2^{(n-k)}$
4: return $(\mathsf{pk} = (\boldsymbol{H}, \boldsymbol{s_x}), \mathsf{sk} = \boldsymbol{x})$
```

Algorithm 5 Sign($\mathsf{pk}, \mathsf{sk}, \boldsymbol{m}$)

```
Input: Public and private keys, message $\boldsymbol{m} \in \{0,1\}^*$ to be signed
Output: Signature $(\boldsymbol{z}, \boldsymbol{c}) \in \mathbb{F}_2^n \times \mathbb{F}_2^n$ of message $\boldsymbol{m}$
1: $\boldsymbol{y} \xleftarrow{\$} D_{w_2}^n$
2: $\boldsymbol{c} \leftarrow \mathcal{H}_n\left(\boldsymbol{H}\boldsymbol{y}^\top, \boldsymbol{m}\right)$
3: $\boldsymbol{z} \leftarrow \boldsymbol{c}\boldsymbol{x} + \boldsymbol{y}$
4: return $(\boldsymbol{z}, \boldsymbol{c})$ with some probability          ▷ See end of this Section
```

Algorithm 6 Verify($\mathsf{pk}, (\boldsymbol{z}, \boldsymbol{c}), \boldsymbol{m}$)

```
Input: Public key, message $\boldsymbol{m}$, and the signature $(\boldsymbol{z}, \boldsymbol{c})$ to verify
Output: Accept if $(\boldsymbol{z}, \boldsymbol{c})$ is a valid signature of $\boldsymbol{m}$, Reject otherwise
1: if $\mathcal{H}_n(\boldsymbol{H}\boldsymbol{z}^\top - \boldsymbol{s_x}\boldsymbol{c}, \boldsymbol{m}) = \boldsymbol{c}$ and $wt(\boldsymbol{z}) \leq w = \delta w_1 + w_2$ then
2:     return Accept
3: else
4:     return Reject
```

Fig. 3. A code-based adaptation of Lyubashevsky's signature scheme.

To sign a message $\boldsymbol{m}$, a mask $\boldsymbol{y}$ of small weight w_2 is sampled uniformly at random, then committed by its syndrome, together with the message, to get the challenge $\boldsymbol{c} = \mathcal{H}_n\left(\boldsymbol{m}, \boldsymbol{H}\boldsymbol{y}^\top\right)$. The response to this challenge is the polynomial product of the secret key and the challenge, hidden by the committed mask: $\boldsymbol{z} = \boldsymbol{c}\boldsymbol{x} + \boldsymbol{y}$. The signature consists of the challenge and the response: $\sigma = (\boldsymbol{c}, \boldsymbol{z})$. The OTS algorithms are depicted in details in Fig. 3.

3.2 A matrix adaptation of Lyubashevsky's signature

We now describe a generalization of Persichetti's OTS to matrices. Our generalization is actually closer to Lyubashevsky's original work [9] for general lattices, not just ideals. It is also more connected to the SD problem in some sense since the response computation involves a syndrome computation instead of just a polynomial multiplication.

Yet while this generalization permits to avoid the full cryptanalysis directly from one signature, it still leaks some information that reveals the secret key with a few signatures. Actually, this construction is similar to a submission to NIST post-quantum standardization[1] process named RaCoSS [11]. One of the main difference with this proposal lies in the distribution of the secret key rows (probabilistic vs deterministic). RaCoSS has already been attacked [3], then patched [12], then attacked again.

The secret key consists of m vectors $\boldsymbol{x}_0, \ldots, \boldsymbol{x}_{m-1}$ of small weights w_1, that constitute the row of the private matrix $\boldsymbol{X} \in \mathbb{F}_2^{m\times n}$. As in the previous subsection, the public key is a random parity-check matrix $\boldsymbol{H} \in \mathbb{F}_2^{(n-k)\times n}$ (not necessarily quasi-cyclic) together with the syndromes of the secret key $\boldsymbol{S_X} = \boldsymbol{H}\boldsymbol{X}^\top \in \mathbb{F}_2^{(n-k)\times m}$.

[1] See https://csrc.nist.gov/projects/post-quantum-cryptography.

```
Algorithm 7 KeyGen($\mathsf{params} = (n, m, k, w_1, w_2, \delta)$)
Input: Public parameters $\mathsf{params} = (n, k, w_1, w_2, \delta)$
Output: $(\mathsf{pk}, \mathsf{sk})$ with $\mathsf{pk} = (\boldsymbol{H}, \boldsymbol{S_X}) \in \mathbb{F}_2^{(n-k)\times n} \times \mathbb{F}_2^{(n-k)\times m}$ and $\mathsf{sk} = \boldsymbol{X} \in \mathbb{F}_2^{m\times n}$
 1: $\boldsymbol{X} \overset{\$}{\leftarrow} \mathcal{S}_{w_1}^n(\mathbb{F}_2)^m \subset \mathbb{F}_2^{m\times n}$      ▷ each row is a low weight vector
 2: $\boldsymbol{H} \overset{\$}{\leftarrow} \mathbb{F}_2^{(n-k)\times n}$
 3: $\boldsymbol{S_X} \leftarrow \boldsymbol{H}\boldsymbol{X}^\top \in \mathbb{F}_2^{(n-k)\times m}$
 4: return $(\mathsf{pk} = (\boldsymbol{H}, \boldsymbol{S_X}), \mathsf{sk} = \boldsymbol{X})$
```

```
Algorithm 8 Sign($\mathsf{pk}, \mathsf{sk}, \boldsymbol{m}$)
Input: Public and private keys, message $\boldsymbol{m} \in \{0,1\}^*$ to be signed
Output: Signature $(\boldsymbol{z}, \boldsymbol{c}) \in \mathbb{F}_2^n \times \mathbb{F}_2^m$ of message $\boldsymbol{m}$
 1: $\boldsymbol{y} \overset{\$}{\leftarrow} D_{w_2}^n$
 2: $\boldsymbol{c} \leftarrow \mathcal{H}_m\left(\boldsymbol{H}\boldsymbol{y}^\top, \boldsymbol{m}\right) \in \mathbb{F}_2^m$
 3: $\boldsymbol{z} \leftarrow \boldsymbol{c}\boldsymbol{X} + \boldsymbol{y}$
 4: return $(\boldsymbol{z}, \boldsymbol{c})$ with some probability      ▷ See end of this Section
```

```
Algorithm 9 Verify($\mathsf{pk}, (\boldsymbol{z}, \boldsymbol{c}), \boldsymbol{m}$)
Input: Public key, message $\boldsymbol{m}$, and the signature $(\boldsymbol{z}, \boldsymbol{c})$ to verify
Output: Accept if $(\boldsymbol{z}, \boldsymbol{c})$ is a valid signature of $\boldsymbol{m}$, Reject otherwise
 1: if $\mathcal{H}_m\left(\boldsymbol{H}\boldsymbol{z}^\top - \boldsymbol{S_X}\boldsymbol{c}^\top, \boldsymbol{m}\right) = \boldsymbol{c}$ and $wt(\boldsymbol{z}) \le w = \delta w_1 + w_2$ then
 2:     return Accept
 3: else
 4:     return Reject
```

Fig. 4. Matrix adaptation of Lyubashevsky's signature scheme.

The main difference between the vector version of subsection 3.1 and the matrix version lies in the signature computation. Indeed, while the first steps are identical, the response computation is pretty different. It resembles much more a McEliece encryption of "message" $\boldsymbol{c}$, with generator matrix $\boldsymbol{X}$ and error $\boldsymbol{y}$. However, the message $\boldsymbol{c}$ is public here, while matrix $\boldsymbol{X}$ is not.

We will argue in the next section that it is possible to learn the support of $\boldsymbol{X}$ using several signatures.

3.3 On the uselessness of rejection sampling

The most important (and costly) step in Lyubashevsky's full-time signature scheme is the final one: rejection sampling. This step is performed before publishing any signature to ensure that the candidate response $\boldsymbol{z} = \boldsymbol{S}\boldsymbol{c} + \boldsymbol{y}$ does not leak any information about the secret key. In other words, it enforces the signature distribution to be statistically (or at least computationally) independent from the secret key. As mentioned before, this is done to let an adversary against the existential unforgeability under chosen message attack (EUF-CMA for short) successfully produce valid signatures without knowing the secret key, in order to then exploit this forgeries to solve the underlying hard problem (namely SIS for [9]).

This has for main consequence that the candidate response is output only with some probability smaller than one. Persichetti's OTS does not use rejection sampling at all: the probability that Algo. 5 and 8 outputs the candidate signature in line 4 is always one!

This is probably done in the hope that the information leaked in the one-time signature is not sufficient to retrieve the secret key. The next sections are devoted to showing that no rejection sampling can prevent Algo. 5 from revealing the secret key in only one signature. We will also argue that Algo. 8 can only be secure for a (very) limited amount of signatures.

4 One-time signature as a decoding problem

In this section, we only consider the vector formulation of subsection 3.1, and rewrite the cryptanalysis problem as a decoding problem.

Recall that in (the general version of) Persichetti's OTS, the signature is a couple $(\boldsymbol{z}, \boldsymbol{c})$ with $\boldsymbol{z} = \boldsymbol{c}\boldsymbol{x} + \boldsymbol{y}$, $wt(\boldsymbol{x}) = w_1$, $wt(\boldsymbol{y}) = w_2$ and $wt(\boldsymbol{c}) = \delta$ so that $wt(\boldsymbol{z}) \leq w = \delta w_1 + w_2$. The author claims [13, Sec. 4 p. 6]:

"*A big advantage of our proposal is that this issue (introducing extra algebraic structure can compromise the secrecy of the private matrix used for decoding) does not apply. In fact, since there is no decoding involved, an entirely random code can be used, and the code itself is public, so there is no private matrix to hide. In this sense, our scheme is closer, to an extent, to the work of [1], which is centered on random quasi-cyclic codes.*"

We show that this statement is not accurate, and that the problem of recovering the secret key (and one time randomness) from the one-time signature can indeed involve decoding.

Polynomial multiplication in $\mathbb{F}_2[x]/\langle x^n - 1\rangle$ can be interchangeably seen as a matrix-vector multiplication in $\mathbb{F}_2^{n\times n} \times \mathbb{F}_2^n$. To do so, we use the following notation: for a vector $\boldsymbol{v} = (\boldsymbol{v}_0, \ldots, \boldsymbol{v}_{n-1}) \in \mathbb{F}_2^n$, we denote by $\mathbf{rot}(\boldsymbol{v})$ the cyclic matrix obtained using the cyclic right shifts of $\boldsymbol{v}$. Formally:

$$\mathbf{rot}(\boldsymbol{v}) = \begin{pmatrix} \boldsymbol{v}_0 & \boldsymbol{v}_{n-1} & \ldots & \boldsymbol{v}_1 \\ \boldsymbol{v}_1 & \boldsymbol{v}_0 & \ldots & \boldsymbol{v}_2 \\ \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{v}_{n-1} & \boldsymbol{v}_{n-2} & \ldots & \boldsymbol{v}_0 \end{pmatrix} \in \mathbb{F}_2^{n\times n} \tag{1}$$

Using the notation above, any polynomial multiplication $\boldsymbol{a} \times \boldsymbol{b}$ can now be written $\mathbf{rot}(\boldsymbol{a})\,\boldsymbol{b}^\top$. We can apply this rewriting to line 3 of the Sign algo. 5 (we re-introduce transpose $^\top$ notation only for dimension correctness):

$$\boldsymbol{z} = \boldsymbol{c}\boldsymbol{x} + \boldsymbol{y} = \mathbf{rot}(\boldsymbol{c})\,\boldsymbol{x}^\top + \boldsymbol{y}^\top. \tag{2}$$

Due to the constraint mentioned in the previous section, namely the GV bound, $\boldsymbol{c}$ has to be of particularly low weight δ. To give an idea of the order of magnitude, if n is the length of the code being used, the challenge should have weight approximately $\delta \in \mathcal{O}\left(n^{1/4}\right)$ for the signature to be unique. This implies in particular that the matrix $\mathbf{rot}(\boldsymbol{c})$ is sparse, and defines a Low or Moderate Density Parity-Check code $\mathcal{C}$.

From a cryptanalytic point of view, we have that the response $\boldsymbol{z}$ in the signature is equal to the syndrome of the secret key $\boldsymbol{x}$ by the sparse matrix $\mathbf{rot}(\boldsymbol{c})$, hidden by a random error $\boldsymbol{y}$ of small weight w_2. But the challenge $\boldsymbol{c}$ is part of the signature so that any adversary $\mathcal{A}$ against the EUF-CMA of the scheme has access to $\boldsymbol{c}$ (and hence $\mathbf{rot}(\boldsymbol{c})$).

Therefore, to recover the secret key $\boldsymbol{x}$ (and one time randomness $\boldsymbol{y}$), $\mathcal{A}$ is left with a noisy version of the syndrome decoding problem, involving a public MDPC code, which contradicts Persichetti's claim as stated.

5 Extended Bit Flipping algorithm

In this Section, we briefly describe a simple extended bit flipping algorithm version. The bit flipping algorithm was originally introduce by Gallager [5] to decode Low Density Parity-Check (LDPC) codes. It later proved to be much more versatile, allowing to efficiently decode Moderate Density Parity-Check (MDPC) codes [10], even with noisy syndromes [4].

The bit flipping algorithm is actually a natural approach for decoding: using the fact that every codeword has a null syndrome, the algorithm aims at reducing the number of unsatisfied parity-check equations at each iteration. By maximum likelihood, a bit x_i of $\boldsymbol{x}$ is flipped if it allows to reduce more than a certain number (threshold) τ of unsatisfied parity check equations $\boldsymbol{s}_j = \boldsymbol{H}_j\boldsymbol{x}$. The algorithm stops when the updated syndrome is null, or has weight less than some bound for the noisy version (the algorithm can also fail and stop after a predefined maximum number N of iterations). The complete algorithm is described in Algo. 10.

We are now equipped with all the tools to perform the full cryptanalysis of the (generalization of the) efficient one-time signature of Persichetti.

Algorithm 10 extended-Bit-Flipping$(\boldsymbol{H}, \boldsymbol{s}, n, k, w, w_{\boldsymbol{e}}, \tau, N)$

```
Input: Parity-check matrix $\boldsymbol{H} \in \mathbb{F}_2^{(n-k)\times n}$, noisy syndrome $\boldsymbol{s} \in \mathbb{F}_2^{n-k}$
Output: $(\boldsymbol{x}, \boldsymbol{e}) \in \mathbb{F}_2^n \times \mathbb{F}_2^{n-k}$ such that $\boldsymbol{s} = \boldsymbol{H}\boldsymbol{x}^\top + \boldsymbol{e}$, $wt(\boldsymbol{x}) \leq w$, and $wt(\boldsymbol{e}) \leq w_{\boldsymbol{e}}$
 1: $\boldsymbol{t} \leftarrow \boldsymbol{s}$, $\boldsymbol{x} \leftarrow \boldsymbol{0} \in \mathbb{F}_2^n$, $\boldsymbol{e} \leftarrow \boldsymbol{0} \in \mathbb{F}_2^{n-k}$, $round \leftarrow 0$.
 2: repeat
 3:     $\boldsymbol{y} \leftarrow \boldsymbol{0} \in \mathbb{F}_2^n$
 4:     for $i \in \{0, \dots, n-1\}$ do
 5:         $count \leftarrow 0$
 6:         for $j \in \{0, \dots, n-k-1\}$ do
 7:             if $t_j = 1$ and $H_{j,i} = 1$ then
 8:                 $count \leftarrow count + 1$
 9:         if $t_j$ and then
10:             $y_i \leftarrow 1$
11:
        $\boldsymbol{x} \leftarrow \boldsymbol{x} \oplus \boldsymbol{y}$
12:     $\boldsymbol{t} \leftarrow \boldsymbol{t} \oplus \boldsymbol{H}\boldsymbol{y}$
13:     $round \leftarrow round + 1$
14: until $wt(\boldsymbol{t}) \leq w_{\boldsymbol{e}}$ or $round > N$
15: if $round \leq N$ then
16:     $\boldsymbol{e} \leftarrow \boldsymbol{s} - \boldsymbol{H}\boldsymbol{x}^\top$
17:     return $(\boldsymbol{x}, \boldsymbol{e})$
18: else
19:     return $\perp$
```

6 Full cryptanalysis of Persichetti's one time signature scheme

In this section, we put the previous pieces together and report as an example a full cryptanalysis of Persichetti's one-time signature. We show that it is possible to recover the secret key (and hence the one time randomness used for signing too) from a single signature in less than a second, for all the proposed parameters. The cryptanalysis is summarized in Algo. 11.

Persichetti uses a special ring instantiation to try to add more confusion to the signature. Let $n = 2p$. In Persichetti's scheme, the secret key consists of $\boldsymbol{x} = (\boldsymbol{x}_0, \boldsymbol{x}_1) \in \mathbb{F}_2^p \times \mathbb{F}_2^p$ of global weight w_1, meaning that $wt(\boldsymbol{x}_0) + wt(\boldsymbol{x}_1) = w_1$. A one-time signature is a couple $(\boldsymbol{z}, \boldsymbol{c}) \in \mathbb{F}_2^n \times \mathbb{F}_2^p$ with $\boldsymbol{z} = (\boldsymbol{z}_0, \boldsymbol{z}_1)$ and $\boldsymbol{z}_i = \boldsymbol{x}_i\boldsymbol{c} + \boldsymbol{y}_i$, such that $wt(\boldsymbol{z}) = wt(\boldsymbol{z}_0) + wt(\boldsymbol{z}_1) \leq w = \delta w_1 + w_2$, and $wt(\boldsymbol{c}) \leq \delta \approx n^{1/4}$. The one-time randomness $\boldsymbol{y} = (\boldsymbol{y}_0, \boldsymbol{y}_1) \in \mathbb{F}_2^p \times \mathbb{F}_2^p$ has global weight w_2. The goal of the cryptanalysis is to retrieve $\boldsymbol{x}_0$ and $\boldsymbol{x}_1$ from $\boldsymbol{z}$ and $\boldsymbol{c}$.

The first step of the cryptanalysis is to decompose the target in two halves:

$$\begin{cases} \boldsymbol{z}_0 = \boldsymbol{x}_0\boldsymbol{c} + \boldsymbol{y}_0 \\ \boldsymbol{z}_1 = \boldsymbol{x}_1\boldsymbol{c} + \boldsymbol{y}_1 \end{cases}$$

Each line of the above system can then be viewed independently as a noisy syndrome decoding problem, with public moderate density parity-check matrix **rot** $(\boldsymbol{c})$ as explained in Section 4.

Using the extended bit flipping algorithm described in Section 5, one can solve each line of the system (τ and N will be specified later in this Section):

$$(\boldsymbol{x}_i, \boldsymbol{y}_i) \leftarrow \text{extended-Bit-Flipping}\left(\mathbf{rot}(\boldsymbol{c}), \boldsymbol{z}_i, p, 0, w_1/2, w_2/2, \tau, N\right). \tag{3}$$

A tiny technical caveat needs to be handled for breaking the scheme in practice: the repartition of the noise. Indeed, while the global weight of $\boldsymbol{x}$ (resp. $\boldsymbol{y}$) is w_1 (resp. w_2), it is unlikely to always have $wt(\boldsymbol{x}_0) = wt(\boldsymbol{x}_1) = w_1/2$ and $wt(\boldsymbol{y}_0) = wt(\boldsymbol{y}_1) = w_2/2$. We therefore introduce a relaxation parameter: the integer $\mathsf{relax} \in \{0, \dots, \min(w_1, w_2)\}$, and will allow the weight of the candidate solution $\tilde{\boldsymbol{x}}_i$ and $\tilde{\boldsymbol{y}}_i$ to be respectively within $w_1/2 \pm \mathsf{relax}$ and $w_2/2 \pm \mathsf{relax}$.

In order to include almost all the possible weight distributions, we choose the parameter relax as follows. We start by approximating the distribution of random binary words of length n and weight w by the binomial distribution of parameters n and $\frac{w}{n}$. This distribution has mean w and standard deviation $\delta = \sqrt{w(n-w)/n}$. According to the 68–95–99.7 rule, choosing $\mathsf{relax} \approx 3 \times \delta$ ensures that almost all random words sampled from the binomial distribution of parameters n and w/n will have an Hamming weight within $w \pm \mathsf{relax}$. Experimentally, setting this parameter this way seems to guarantee the success of the algorithm almost all the time at the expense of a higher complexity. Of course, any other experimental choice can be made for choosing this parameter.

Algorithm 11 BreakOTS($\mathsf{params}, \boldsymbol{z}, \boldsymbol{c}, \tau, N, \mathsf{relax}$)

```
Input: Public parameters $n, w_1, w_2, \delta$, valid signature $(\boldsymbol{z}, \boldsymbol{c})$ on message $\boldsymbol{m}$
Output: $(\boldsymbol{x}, \boldsymbol{y}) \in \mathcal{S}^n_{w_1}(\mathbb{F}_2) \times \mathcal{S}^n_{w_2}(\mathbb{F}_2)$ such that $\boldsymbol{x} = \mathsf{sk}$ and $\boldsymbol{z} = \boldsymbol{cx} + \boldsymbol{y}$
1: $(\boldsymbol{s}_0, \boldsymbol{s}_1) \leftarrow (\boldsymbol{z}_0, \boldsymbol{z}_1)$, $(\boldsymbol{x}_0, \boldsymbol{x}_1) \leftarrow (\boldsymbol{0}, \boldsymbol{0})$, $(\boldsymbol{y}_0, \boldsymbol{y}_1) \leftarrow (\boldsymbol{0}, \boldsymbol{0})$
2: repeat
3:     $(\boldsymbol{x}_0, \boldsymbol{y}_0) \leftarrow \mathsf{extended\text{-}Bit\text{-}Flipping}(\boldsymbol{c}, \boldsymbol{z}_0, n, k, w_1/2, w_2/2 + \mathsf{relax}, \tau, N)$
4:     $(\boldsymbol{x}_1, \boldsymbol{y}_1) \leftarrow \mathsf{extended\text{-}Bit\text{-}Flipping}(\boldsymbol{c}, \boldsymbol{z}_1, n, k, w_1 - wt(\boldsymbol{x}_0), w_2 - wt(\boldsymbol{y}_0), \tau, N)$
5: until
6: return $(\mathsf{sk} = (\boldsymbol{x}_0, \boldsymbol{x}_1), \boldsymbol{y})$
```

Finally, a basic implementation of the cryptanalysis is available at `https://github.com/deneuville/PersichettiOTScryptanalysis`. The code was compiled using GCC 5.4.0 using flags `-std=c++11 -fpermissive -O3`, and run on a single Intel® Core™ i7-6920HQ CPU @ 2.90GHz with TurboBoost disabled. The timings reported in Tab. 1 are expressed in milliseconds. The verification timings come from the original paper [13]. While they were obtained on a seemingly less powerful device, they compare favorably to our highly unoptimized proof of concept implementation of Persichetti's OTS. Therefore we conservatively chose to refer to these timings.

	Persichetti's OTS parameters				xBF parameters		Verification	Cryptanalysis
security	n	w_1	w_2	δ	τ	N	t_{verify} (ms)	t_{break} (ms)
80	4801	90	100	10	7	5	22.569	165.459
	3072	85	85	7	5	5	14.271	68.858
128	9857	150	200	12	9	10	99.492	453.680
	6272	125	125	10	7	10	42.957	288.442

Table 1. Parameters for Persichetti's OTS (from [13]) and for the extended Bit Flipping (xBF) algorithm. All timings are in milliseconds. The timings for the cryptanalysis roughly correspond to two xBF runs (one for each part of the secret key). The verification timings were taken directly from [13]. The last xBF parameter relax (not shown in this table) is always set to $w_2/4$.

Notice that Algo. 11 is presented using quasi-cyclic codes: the parity-check matrix given to the extended bit flipping algorithm consists of the cyclic shift of vector $\boldsymbol{c}$. The cryptanalysis timings reported in Tab. 1 correspond to a generic version of this extended bit flipping algorithm. Due to the very peculiar structure of the parity-check matrix (cyclic and sparse), is actually possible to optimize much better the extended bit flipping algorithm. Persichetti's verification requires:

- one syndrome computation: $\boldsymbol{s}_{\boldsymbol{z}} = \boldsymbol{z}_0 + \boldsymbol{h}\boldsymbol{z}_1$, equivalent to one full-sparse polynomial multiplication and one addition,
- a sparse-full polynomial multiplication: $\boldsymbol{c}\boldsymbol{s}_{\boldsymbol{x}}$, and
- another polynomial addition: $\boldsymbol{c}\boldsymbol{s}_{\boldsymbol{x}} + \boldsymbol{s}_{\boldsymbol{z}}$

An extended bit flipping essentially corresponds to a syndrome computation, plus some polynomial additions on positions flipped during execution. Therefore, an optimized xBF algorithm taking advantage from this cyclic and sparse structure would require:

- one syndrome computation: $\boldsymbol{c}\tilde{\boldsymbol{x}}_b$ $(b \in \{0, 1\})$, $\tilde{\boldsymbol{x}}_b$ being the guessed secret, equivalent to one sparse-sparse polynomial multiplication,
- w_1 polynomial additions (equivalent to another sparse-sparse polynomial multiplication), and
- some other overhead polynomial additions and memory access for threshold verification and syndrome updates.

Two extended bit flipping runs are required for the full cryptanalysis, involving twice as many polynomial multiplications (the most expansive operation) as for the signature verification. This reasonably lets us believe that a fully optimized cryptanalysis implementation should completely break Persichetti's OTS scheme in no longer than twice the verification time.

7 Conclusion

In this paper, we have presented an attack on efficient one-time signatures without trapdoors, based on codes (not necessarily quasi-cyclic). This attack targets the vectorial adaptation of Lyubashevsky's signature scheme. Viewing the commitment as an LDPC/MDPC code, it is possible to rewrite the signature as a noisy syndrome decoding problem, for which the extended bit flipping is especially suited. Applied to Persichetti's scheme, we retrieve the secret key in less than a second for all parameters, disproving the claimed 80 to 128 bits security. While the matrix version of this adaptation seems immune – or at least less sensitive – to this attack, it clearly leaks information on the support of the secret key, that can be retrieved using a few signatures, as noticed in the original adaptation [13].

References

[1] Aguilar Melchor, C., Blazy, O., Deneuville, J., Gaborit, P., Zémor, G.: Efficient encryption from random quasi-cyclic codes. IEEE Trans. Information Theory **64**(5) (2018) 3927–3943 *2, 7*

[2] Berlekamp, E.R., McEliece, R.J., van Tilborg, H.C.A.: On the inherent intractability of certain coding problems (corresp.). IEEE Trans. Information Theory **24**(3) (1978) 384–386 *1, 3*

[3] Daniel Julius, B., Andreas, H., Tanja, L., Panny, L.: OFFICIAL COMMENT: RaCoSS. Official comments about NIST PQC submissions (December 2017) *5*

[4] Deneuville, J.C., Gaborit, P., Zémor, G.: Ouroboros: A simple, secure and efficient key exchange protocol based on coding theory. In: International Workshop on Post-Quantum Cryptography, Springer (2017) 18–34 *7*

[5] Gallager, R.: Low-density parity-check codes. IRE Transactions on information theory **8**(1) (1962) 21–28 *2, 7*

[6] Gentry, C., Peikert, C., Vaikuntanathan, V.: Trapdoors for hard lattices and new cryptographic constructions. In Ladner, R.E., Dwork, C., eds.: 40th ACM STOC, ACM Press (May 2008) 197–206 *1, 3*

[7] Gilbert, E.N.: A comparison of signalling alphabets. Bell System Technical Journal **31**(3) (1952) 504–522 *3*

[8] Hoffstein, J., Pipher, J., Silverman, J.H.: NSS: An NTRU lattice-based signature scheme. In Pfitzmann, B., ed.: EUROCRYPT 2001. Volume 2045 of LNCS., Springer, Heidelberg (May 2001) 211–228 *1, 3*

[9] Lyubashevsky, V.: Lattice signatures without trapdoors. In Pointcheval, D., Johansson, T., eds.: EUROCRYPT 2012. Volume 7237 of LNCS., Springer, Heidelberg (April 2012) 738–755 *1, 3, 5, 6*

[10] Misoczki, R., Tillich, J.P., Sendrier, N., Barreto, P.S.: Mdpc-mceliece: New mceliece variants from moderate density parity-check codes. In: Information Theory Proceedings (ISIT), 2013 IEEE International Symposium on, IEEE (2013) 2069–2073 *2, 7*

[11] Partha Sarathi, R., Rui, X., Kazuhide, F., Shinsaku, K., Kirill, M., Tsuyoshi, T.: RaCoSS: Random code-based signature scheme. Submission to NIST post-quantum standardization process (November 2017) *5*

[12] Partha Sarathi, R., Rui, X., Kazuhide, F., Shinsaku, K., Kirill, M., Tsuyoshi, T.: Code-based signature scheme without trapdoors. IEICE Tech. Rep., vol. 118, no. 151, ISEC2018-15, pp. 17–22 (July 2018) https://www.ieice.org/ken/paper/20180725L1FF/eng/. *5*

[13] Persichetti, E.: Efficient one-time signatures from quasi-cyclic codes: A full treatment. Cryptography **2**(4) (2018) 30 *1, 2, 7, 9, 10*

[14] Pointcheval, D., Stern, J.: Security arguments for digital signatures and blind signatures. Journal of cryptology **13**(3) (2000) 361–396 *1*

[15] Varshamov, R.: Estimate of the number of signals in error correcting codes. Docklady Akad. Nauk, SSSR **117** (1957) 739–741 *3*