Breaking HWQCS: a code-based signature scheme from high weight QC-LDPC codes

Alex Pellegrini[1] , Giovanni Tognolini[2]

[1] Department of Mathematics, Eindhoven University of Technology
[2] Department of Mathematics, University of Trento
alex.pellegrini@live.com, giovanni.tognolini@unitn.it

Abstract. We analyse HWQCS, a code based signature scheme presented at ICISC 2023, which uses quasi-cyclic low density parity check codes (QC-LDPC). The scheme introduces high Hamming weight errors and signs each message using a fresh ephemeral secret key rather than using only one secret key, so to avoid known attacks on QC-LDPC signature schemes. In this paper, we show that the signatures of HWQCS leak substantial information concerning the ephemeral keys and formally describe this behaviour. Furthermore, we show that for each security level, we can exploit the leakage to efficiently reconstruct partial secret data from very few signatures, and finally mount a universal forgery attack.

Keywords: post-quantum cryptography, code-based cryptography, hash&sign signature, universal forgery, information leakage

1 Introduction

Linear codes are well known in cryptography, and their use spans multiple branches of this subject. In this work we deal with a code based digital signature scheme, and here we provide an overview of how linear codes have been used with this aim. Two important computationally-hard algebraic problems arising in coding theory are the *Syndrome Decoding Problem* (SDP) and the *Codeword Finding Problem* (CFP). Several code-based digital signature schemes are built upon SDP, and an adversary who wants to break such schemes is usually left with the only option of solving these problems. Currently, the two main approaches for solving these problems are information set decoding (ISD) algorithms and the generalized birthday algorithm (GBA). ISD algorithms demonstrate greater efficiency when the decoding problem involves only a limited number of solutions, while GBA is more effective when the expected number of solutions is high. We refer the reader to [22] for more details on these techniques.

Code-based digital signature schemes can be divided into two variants. On one hand there are the schemes based on proofs of knowledge, which in turn can follow the Fiat-Shamir [9] or the Schnorr-Lyubashevsky [13] approach, while on the other side there are the schemes following the hash&sign paradigm.

According to the aim of this work, we focus on digital signature schemes which follow the hash&sign paradigm. Among these schemes, CFS [2, 3] and KKS [11] were the first to be introduced. The idea behind both of them is the same: the digest of a message to be signed is considered as the syndrome of a corrupted codeword, and the owner of the private key is the only one capable of decoding. The main difference between CFS and KKS is that in the former the digest is considered as a random syndrome, leading to an inefficient scheme, while the latter manipulates the digest to output a decodable syndrome, leading to a very efficient but insecure scheme [16]. In order to mitigate the efficiency issues of CFS, some authors have proposed similar schemes in which the hash function is replaced by a map whose output is a syndrome with small-weight coset-leader, combining some ideas behind both CFS and KKS. This approach has been adopted e.g. in [18], where the authors use a hash function based on

the works of Augot, Finiasz and Sendrier [1] and on the Merkle-Damgard construction [4, 15]. However, this approach has been proved insecure in [8].

In 2014, the first rank metric code based signature scheme RankSign [10] has been designed, which uses augmented low-rank parity check codes (LRPC). A structural attack [6] on RankSign was developed in 2018. During the last years there have been different attempts to build similar digital signature schemes [17, 10, 12, 19] but it seems that the cryptographic community is far from achieving a trustable proposal [20, 23, 7]. In this context, a remarkable hash&sign code based digital signature scheme that still remains unbroken is Wave [5], which has been proposed in 2019 and relies on the indistinguishability of the normalized generalized $(U, U+V)$ codes.

In 2023, a new attempt to build a code based signature scheme, called HWQCS [21], has been made. HWQCS uses QC-LDPC codes with the Hamming metric and introduces the use of high weight errors to make the decoding problem harder for an attacker. The construction of this scheme is devised in such a way to prevent other known attacks from being effective.

Contributions. This work addresses the cryptanalysis of [21]. First, we show that every signature reveals some information about the ephemeral keys used in the signing process. We formally describe the stochastic behaviour of the exposed information leakage and provide experimental evidence in support of our claims.

As a second contribution, we show that, for any security level (128,192 and 256 bits) an attacker only needs a handful of signatures in order to be able to recover partial secret data, which is meant to be known only by the signer. We compute lower bounds of the success probability of this event for every intercepted signature. After, we proceed to mount a universal forgery attack using the data recovered using our strategy.

Finally, we provide a probabilistic algorithm along with a SageMath implementation and supporting experimental data, that efficiently outputs the secret data needed to forge valid signatures for arbitrary messages. We also propose a slight speed-up for the security level 256 which turns out to be the best scenario for our attack.

Related works. Part of this work follows a strategy similar to that of [20]. The idea is to analyse a valid signature and exploit the bias on the distribution of the signature coefficients to perform a statistical test and thus obtain part of the key.

Organization of this paper. Section 2 defines the necessary background and notation which will be used throughout the paper. In Section 3 we recall the setup, key generation, sign and verification phases of the HWQCS scheme. In Section 4 we describe how the scheme leaks information and how to exploit very few signatures to completely reconstruct the private ephemeral values used to sign. Finally, Section 5 presents our attack in its full generality, showing how to mount a universal forgery attack, and we propose a speed-up for the security level 256. We provide Sagemath code and experimental data in support of our results.

2 Background and Notation

Let $\mathbb{F}_2$ be the finite field with two elements. The Hamming weight of a vector $\mathbf{a} \in \mathbb{F}_2^n$, with $\mathbf{a} = (a_1, a_2, \ldots, a_n)$, is defined as

$$\mathsf{wt}_H(\mathbf{a}) := \#\left\{i \in [n] \mid a_i \neq 0\right\},$$

where we denote with $[n]$ the interval of values $\{1, \ldots, n\}$. The Hamming distance between two vectors $\mathbf{a}, \mathbf{b} \in \mathbb{F}_2^n$ is defined as $d_H(\mathbf{a}, \mathbf{b}) = \mathsf{wt}_H(\mathbf{a} - \mathbf{b})$. We denote with $V_{n,w} \subset \mathbb{F}_2^n$ the set of vectors of length n and Hamming weight w. Let $k, n \in \mathbb{N}$, then a linear $[n, k, d]$-code C is a k-dimensional subspace of $\mathbb{F}_2^n$ with minimum distance

$$d := \min_{\mathbf{a},\mathbf{b} \in C, \mathbf{a} \neq \mathbf{b}} d_H(\mathbf{a} - \mathbf{b}).$$

In this work, we consider vectors as row vectors, and thus their transposes as column vectors. Let $R := \mathbb{F}_2[x]/(x^n - 1)$ be the ring of all polynomials over $\mathbb{F}_2$ of degree less than n. For $\mathbf{a} \in \mathbb{F}_2^n$, denote by $\mathbf{a}(x)$ the unique polynomial $\mathbf{a}(x) = a_1 + a_2 x + \cdots + a_n x^{n-1} \in R$. Given

a vector $\mathbf{a} \in \mathbb{F}_2^n$, we denote its j-th entry by a_j when $\mathbf{a} = (a_1, \ldots, a_n)$ is specified, and by $(\mathbf{a})_j$ otherwise. Similarly, we denote the j-th coefficient of a polynomial $\mathbf{a}(x) \in R$ as $(\mathbf{a}(x))_j$. Consider two elements $\mathbf{a}, \mathbf{b} \in \mathbb{F}_2^n$, then the vector $\mathbf{c} \in \mathbb{F}_2^n$, such that $\mathbf{c}(x) = \mathbf{a}(x)\mathbf{b}(x)$, can be computed by the formula $\mathbf{c}^\top = \mathsf{circ}(\mathbf{a}) \cdot \mathbf{b}^\top$, where $\mathsf{circ}(\mathbf{a})$ is the $n \times n$ circulant matrix obtained from $\mathbf{a}$ as

$$\mathsf{circ}(\mathbf{a}) = \begin{pmatrix} a_1 & a_n & \cdots & a_2 \\ a_2 & a_1 & \cdots & a_3 \\ \vdots & \vdots & \ddots & \vdots \\ a_n & a_{n-1} & \cdots & a_1 \end{pmatrix},$$

whose j-th row is a circular right shift of the first row of j positions. Due to this relationship, it is possible to link the description of the scheme presented in [21] to the Syndrome Decoding Problem. In particular, HWQCS uses two ephemeral values $\mathbf{e}_1(x), \mathbf{e}_2(x) \in R$ with low weight, a public element $\mathbf{h}(x) \in R$, and computes the public value $\mathbf{b}(x) = \mathbf{e}_1(x) \cdot \mathbf{h}(x) + \mathbf{e}_2(x) \cdot \mathbf{h}^{-1}(x)$. An attacker who wants to retrieve the ephemeral values is therefore asked to solve the following decoding problem:

$$\mathbf{b} = \left(\mathsf{circ}(\mathbf{h}) \,|\, \mathsf{circ}(\mathbf{h})^{-1}\right) (\mathbf{e}_1, \mathbf{e}_2)^\top.$$

3 The HWQCS Signature Scheme

This section describes the algorithms of HWQCS [21] for key generation KeyGen, signature Sign and the verification Verify. We assume that the global parameters $k, w_f, w_u, w_e, w_c, w_s, w_t$, R and hash_{ω_c} provided in the setup are public, and we do not specify them as input of the algorithms.

Setup. The parameters of the scheme are the positive integers $k, w_f, w_u, w_e, w_c, w_s, w_t$, the quotient ring $R = \mathbb{F}_2[x]/(x^k - 1)$ and hash_{ω_c} a hash function with fixed output Hamming weight w_c

Key Generation. The Key Generation algorithm KeyGen is the following.

Algorithm 3.1 Key generation algorithm KeyGen

Input : $\emptyset$.
Output : a public key $\mathsf{pk} \in R$ and a private key $\mathsf{sk} \in R^2$.

1. Choose random $\mathbf{f}_1, \mathbf{f}_2 \in V_{k,w_f}$ such that $\mathbf{f}_1(x), \mathbf{f}_2(x) \in R$ are invertible.
2. Compute $\mathbf{h}(x) := \mathbf{f}_1(x)^{-1} \cdot \mathbf{f}_2(x) \in R$.
3. Output the public key $\mathsf{pk} = \mathbf{h}$ and the private key $\mathsf{sk} = (\mathbf{f}_1, \mathbf{f}_2)$.

Signature Algorithm. The signing algorithm Sign is as follows.

Algorithm 3.2 Signature algorithm Sign

Input : the message to sign m, a public key $\mathsf{pk} = \mathbf{h} \in \mathbb{F}_2^n$ and a secret key $\mathsf{sk} = (\mathbf{f}_1, \mathbf{f}_2) \in \mathbb{F}_2^n \times \mathbb{F}_2^n$.
Output : a valid signature $\sigma \in (\mathbb{F}_2^n)^4$ for m under pk and sk.

1. Choose random $\mathbf{e}_1, \mathbf{e}_2 \in V_{k,w_e}$ and $\mathbf{u}_1, \mathbf{u}_2 \in V_{k,w_u}$.
2. Set
   - $\mathbf{b}(x) := \mathbf{e}_1(x)\mathbf{h}(x) + \mathbf{e}_2(x)\mathbf{h}^{-1}(x) \in R$ and
   - $\mathbf{d}(x) = \mathbf{u}_1(x)\mathbf{f}_2(x) + \mathbf{u}_2(x)\mathbf{f}_1(x)$.
3. Compute $\mathbf{c} := \mathsf{hash}_{\omega_c}(m, \mathbf{b}, \mathbf{d}, \mathbf{h}) \in V_{k,w_c}$.
4. Compute $\mathbf{s}_i(x) := \mathbf{u}_i(x)\mathbf{f}_i(x) + \mathbf{c}(x)\mathbf{e}_i(x)$ for $i = 1, 2$.
5. If $w(\mathbf{s}_1) > w_s$ or $w(\mathbf{s}_2) > w_s$ or $w(\mathbf{d}) > w_t$ then repeat from Step 1.
6. With this notation, the signature is given by $\sigma = (\mathbf{c}, \mathbf{b}, \mathbf{s}_1, \mathbf{s}_2)$.

Verification Algorithm. The verification algorithm Verify is as following.

Algorithm 3.3 Verification algorithm Verify

Input : A message m, a public key $\mathsf{pk} = \mathbf{h} \in \mathbb{F}_2^n$ and a signature $\sigma = (\mathbf{c}, \mathbf{b}, \mathbf{s}_1, \mathbf{s}_2) \in (\mathbb{F}_2^n)^4$.
Output : accept or reject signature σ for m.

1. Compute $\mathbf{t}(x) := \mathbf{s}_1(x)\mathbf{h}(x) + \mathbf{s}_2(x)\mathbf{h}^{-1}(x) - \mathbf{c}(x)\mathbf{b}(x)$.
2. Compute $\mathbf{c}' := \mathsf{hash}_{\omega_c}(m, \mathbf{b}, \mathbf{t}, \mathbf{h}) \in V_{k,w_c}$.
3. If $\mathbf{c} = \mathbf{c}'$, $\mathsf{wt}_H(\mathbf{t}) \leq w_t$ and $\mathbf{t}(x) \neq 0$, then accept, otherwise reject.

The sets of parameters suggested for 128,192 and 256 bits of security are reported in Table 1.

Security	$(k, \omega_f, \omega_u, \omega_e, \omega_c, \omega_s, \omega_t)$
128	(12539, 145, 33, 141, 31, 4844, 4937)
192	(18917,185,41,177,39,7450,7592)
256	(25417,201,51,191,51,10111,10216)

Table 1. Suggested parameters for security levels 128,192 and 256 of HWQCS.

4 Information Leakage of HWQCS

We start this section by giving an overview of our attack, which serves as a motivation for the study of the objects developed in the rest of the section, where we show that the signatures generated by the Sign algorithm of HWQCS leak a critical amount of information about the ephemeral values $\mathbf{e}_1$ and $\mathbf{e}_2$.

4.1 Overview of the attack

The idea is to use information set decoding techniques in a simplified scenario, which allows us to perform fast recovery of secret data, hence forge signatures. In other words, we aim at finding error free positions in the vector $\mathbf{e}$. To this end, we apply a similar analysis to [20], which manipulates each intercepted signature in such a way that it is possible to separate a good amount of zero bits of $\mathbf{e}_1$ and $\mathbf{e}_2$ from the one bits. Furthermore, we can carefully guess additional zero bits. Once k, zero bits have been recovered, we can perform linear algebra using a submatrix of the public matrix

$$\mathbf{H} = \left[\mathsf{circ}(\mathbf{h}) \mid \mathsf{circ}(\mathbf{h})^{-1}\right]. \tag{1}$$

to reconstruct the values $\mathbf{e}_1, \mathbf{e}_2$ completely. Finally, compute $\mathbf{u}_i(x)\mathbf{f}_i(x) = \mathbf{s}_i(x) - \mathbf{c}(x)\mathbf{e}_i(x)$ for $i = 1, 2$. At this point it is possible to start forging valid signatures, see Section 5 for a detailed explanation and Algorithm 5.1 for the concrete attack.

In the next section, we will formally describe the behaviour of the information leakage, which will be useful to determine the necessary objects for our attack. Our analysis is identical for both $i = 1, 2$, therefore we won't fix a value of i.

4.2 Statistical analysis of the information leakage

We describe the technique introduced in [20] adapted to the setting of HWQCS. Let $\sigma = (\mathbf{c}, \mathbf{b}, \mathbf{s}_1, \mathbf{s}_2)$ be an intercepted signature where $\mathbf{s}_i(x) := \mathbf{u}_i(x)\mathbf{f}_i(x) + \mathbf{c}(x)\mathbf{e}_i(x)$, as per Sign. We can expose many one bits of $\mathbf{e}$ as follows:

– Let $v \in \mathsf{supp}(\mathbf{c})$ and write

$$\begin{aligned} x^{-v}\mathbf{s}_i(x) &= x^{-v}\left(\mathbf{u}_i(x)\mathbf{f}_i(x) + \mathbf{c}(x)\mathbf{e}_i(x)\right) \\ &= x^{-v}\mathbf{u}_i(x)\mathbf{f}_i(x) + \left(1 + x^{-v}\sum_{l \in \mathsf{supp}(\mathbf{c})\setminus\{v\}} x^l\right)\mathbf{e}_i(x) \\ &= \mathbf{e}_i(x) + x^{-v}\mathbf{u}_i(x)\mathbf{f}_i(x) + \sum_{l \in \mathsf{supp}(\mathbf{c})\setminus\{v\}} x^{l-v}\mathbf{e}_i(x). \end{aligned} \tag{2}$$

– Finally, compute

$$\mathbf{d}_i(x) := \sum_{v \in \mathsf{supp}(\mathbf{c})} x^{-v}\mathbf{s}_i(x) \in \mathbb{Z}[x], \tag{3}$$

where the sum in 3 is taken over $\mathbb{Z}$. Notice that $x^{-v}\mathbf{u}_i(x)\mathbf{f}_i(x)$ is a polynomial whose coefficient vector is a circular left shift of the coefficient vector of $\mathbf{u}_i(x)\mathbf{f}_i(x)$ of v positions. The same holds for $x^{l-v}\mathbf{e}_i(x)$, so that for every $v \in \mathsf{supp}(\mathbf{c})$, the value $x^{-v}\mathbf{s}_i(x)$ is given by $\mathbf{e}_i(x)$ plus some random noise, see (2). Therefore, we expect the larger coefficients of $\mathbf{d}_i(x)$ to be associated with the entries of $\mathbf{e}_i(x)$ equal to 1. Figures 1, 2 and 3 give a visual representation of the information leakage for a random signature, of an instance of HWQCS for security levels 128, 192 and 256.

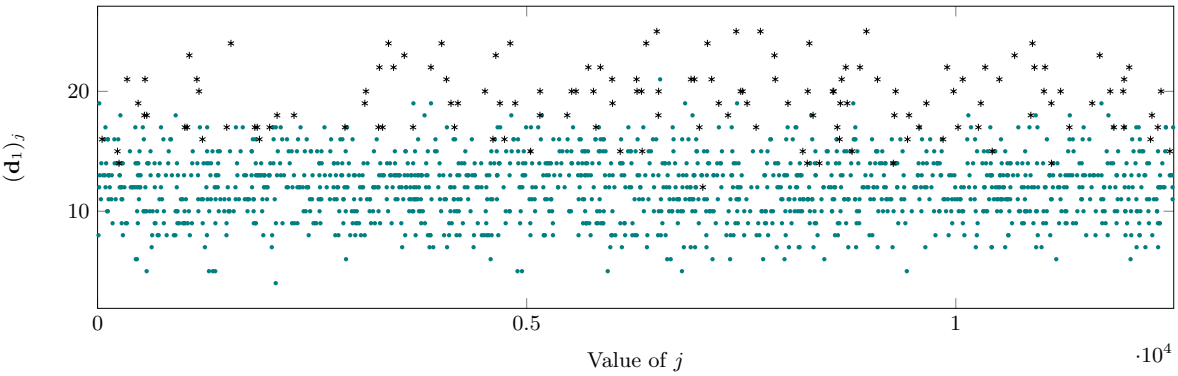

Fig. 1. Information leakage of $\mathbf{e}_1(x)$ of an instance of HWQCS with security parameter 128. We highlighted with green dots the entries j for which $(\mathbf{e}_1)_j = 0$, and with black asterisks the entries j for which $(\mathbf{e}_1)_j = 1$.

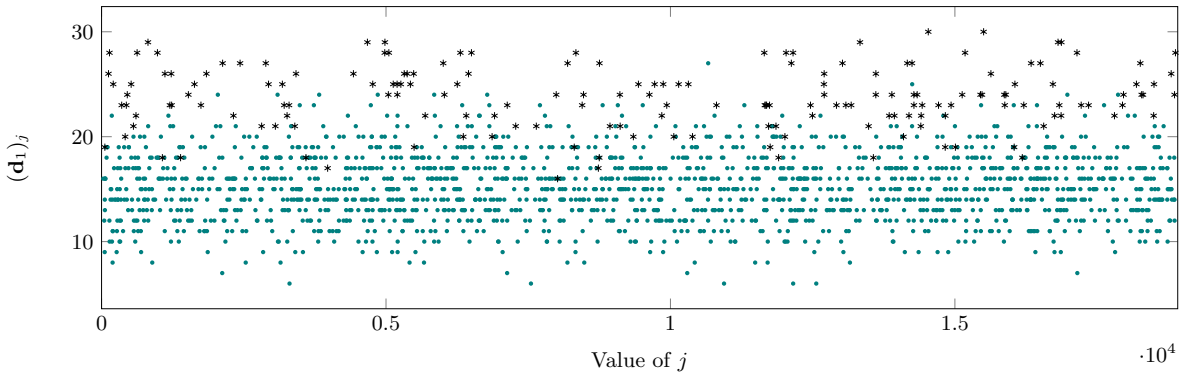

Fig. 2. Information leakage of $\mathbf{e}_1(x)$ of an instance of HWQCS with security parameter 192. We highlighted with green dots the entries j for which $(\mathbf{e}_1)_j = 0$, and with black asterisks the entries j for which $(\mathbf{e}_1)_j = 1$.

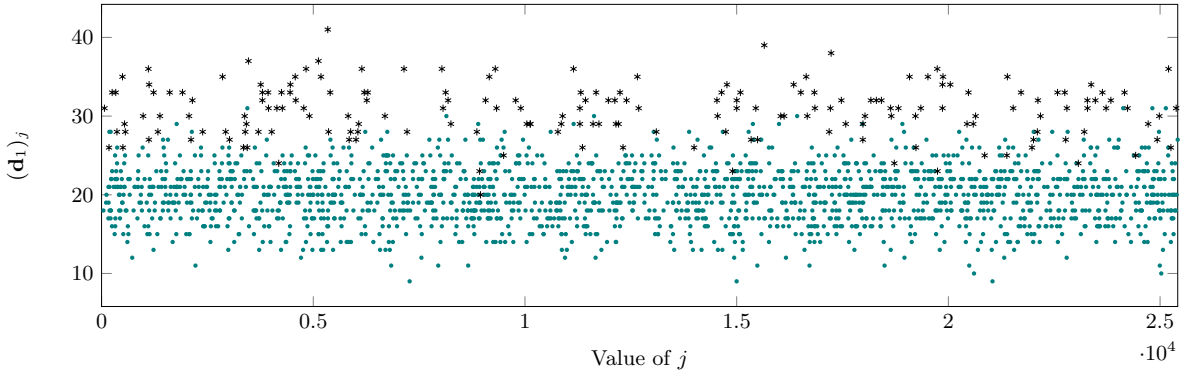

Fig. 3. Information leakage of $\mathbf{e}_1(x)$ of an instance of HWQCS with security parameter 256. We highlighted with green dots the entries j for which $(\mathbf{e}_1)_j = 0$, and with black asterisks the entries j for which $(\mathbf{e}_1)_j = 1$.

From figures 1, 2 and 3, one can easily see that, for each security level, the one entries of $\mathbf{e}_i$ (in the examples in the figures $i = 1$) are pushed to the upper half of the plot, meaning that they correspond to the higher coefficients of $\mathbf{d}_i(x)$. This means that, with great probability, a random entry $(\mathbf{d}_i)_j \in \mathbb{Z}$ which is small enough will correspond to a zero entry $(\mathbf{e}_i)_j$. The rest of this section is dedicated to the proof of Theorem 1, which describes the behaviour of the entries of $\mathbf{d}_i \in \mathbb{Z}^k$.

Theorem 1. *Let $\mathbf{u}(x), \mathbf{f}(x), \mathbf{e}(x)$ and $\mathbf{c}(x)$ be random elements of R, in such a way that $\mathsf{wt}_H(\mathbf{u}) = w_u, \mathsf{wt}_H(\mathbf{f}) = w_f, \mathsf{wt}_H(\mathbf{e}) = w_e$ and $\mathsf{wt}_H(\mathbf{c}) = w_c$, respectively. Let $v \in \mathsf{supp}(\mathbf{c})$ and define*

$$\mathbf{d}(x) := \sum_{v \in \mathsf{supp}(\mathbf{c})} \left(\mathbf{e}_i(x) + x^{-v}\mathbf{u}_i(x)\mathbf{f}_i(x) + \sum_{l \in \mathsf{supp}(\mathbf{c}) \setminus \{v\}} x^{l-v}\mathbf{e}_i(x) \right) \in \mathbb{Z}[x].$$

Assume that the random variables $x^{-v}\mathbf{u}_i(x)\mathbf{f}_i(x)$ and $\sum_{l \in \mathsf{supp}(\mathbf{c}) \setminus \{v\}} x^{l-v}\mathbf{e}(x)$ are independently distributed, as v and l vary. Then, for $j \in [k-1]$, we have that $(\mathbf{d}(x))_j$ is binomially distributed

as

$$(\mathbf{d}(x))_j \sim Bin\left(w_c, \prod_{i=1}^{3} p_i + \sum_{i=1}^{3} p_i \prod_{j\in[3], j\neq i} (1-p_j)\right), \tag{4}$$

where

$$p_1 = w_e/k, \qquad p_2 = (1-(1-2w_e/k)^{w_c-1})/2$$

and

$$p_3 = \frac{1}{\binom{k}{w_u}\binom{k}{w_f}} \sum_{\substack{1\leq l\leq \min(w_u,w_f)\\ l \text{ odd}}} \binom{k}{l}\binom{k-l}{w_u-l}\binom{k-w_u}{w_f-l}.$$

Let $j \in [k-1]$. We will analyse the distribution of $(x^{-v}\mathbf{s}_i(x))_j$ for a fixed $v \in \mathsf{supp}(\mathbf{c})$, breaking down the analysis on the three summands of the last equation in (2), treated as random variables. Namely, we are going to compute the probability distributions of $(x^{-v}\mathbf{u}_i(x)\mathbf{f}_i(x))_j$ and of $\left(\sum_{l\in\mathsf{supp}(\mathbf{c})\setminus\{v\}} x^{l-v}\mathbf{e}_i(x)\right)_j$, while we consider $(\mathbf{e}_i)_j$ to be Bernoulli distributed with parameter ω_e/n. After, we will provide the explicit probability distribution of the sum of the three random variables and describe the distribution of $(\mathbf{d}_i(x))_j = \left(\sum_{v\in\mathsf{supp}(\mathbf{c})} x^{-v}\mathbf{s}_i(x)\right)_j$, where the sum is taken over $\mathbb{Z}$.

The following result comes in handy to compute the probability distribution of the entry $(x^{-v}\mathbf{u}_i(x)\mathbf{f}_i(x))_j$.

Lemma 1. *[14, Proposition 2.4.1] Let $\mathbf{u}(x), \mathbf{f}(x) \in R$ be random elements such that $\mathsf{wt}_H(\mathbf{u}) = \omega_u$ and $\mathsf{wt}_H(\mathbf{f}) = \omega_f$. Set $\mathbf{z}(x) = \mathbf{u}(x)\mathbf{f}(x)$, then for every $j \in \{0,\dots,k-1\}$, we have that $(\mathbf{z}(x))_j$ is distributed as a Bernoulli random variable with parameter $p = P(z_j = 1)$ equal to:*

$$p = \frac{1}{\binom{k}{w_u}\binom{k}{w_f}} \sum_{\substack{1\leq l\leq min(w_u,w_f)\\ l \text{ odd}}} \binom{k}{l}\binom{k-l}{w_u-l}\binom{k-w_u}{w_f-l}. \tag{5}$$

Lemma 1 provides us with a way to compute the distribution of $(\mathbf{u}_i(x)\mathbf{f}_i(x))_j$.

Remark 1. Note that $x^{-v}\mathbf{u}_i(x)\mathbf{f}_i(x)$ is a polynomial whose coefficient vector is a circular left shift of the coefficient vector of $\mathbf{u}_i(x)\mathbf{f}_i(x)$ of v positions. Therefore, the two random variables $(\mathbf{u}_i(x)\mathbf{f}_i(x))_j$ and $\left(x^{-v}\mathbf{u}_i(x)\mathbf{f}_i(x)\right)_j$ are identically distributed.

We move on to the distribution of $\sum_{l\in\mathsf{supp}(\mathbf{c})\setminus\{v\}} x^{l-v}\mathbf{e}_i(x)$. We will treat $(x^{l-v}\mathbf{e}_i(x))_j$ as independent random variables, as l and v vary.

Lemma 2. *Let $X_1, X_2, \dots, X_k$ be k independent random variables following a Bernoulli distribution with parameter $q < \frac{1}{2}$ and let $X = \sum_{h=1}^{k} X_h$ be their sum over $\mathbb{F}_2$. Then X is Bernoulli distributed with parameter $p = \mathbb{P}(X = 1)$ equals to:*

$$p = \left(\frac{1}{2} - \frac{(1-2q)^k}{2}\right). \tag{6}$$

Proof. The random variable X is Bernoulli distributed with parameter recursively given by $T(k) = T(k-1)(1-q) + q(1-T(k-1))$, with $T(0) = 0$. Our goal is to convert this expression into a closed formula. Consider the formal power series $f(x) := \sum_{h=1}^{\infty} T(h)x^h$ where we consider

the real interval $x \in (0,1)$. We have that

$$\begin{aligned}
f(x) &= \sum_{h=1}^{\infty} T(h)x^h \\
&= qx + \sum_{h=2}^{\infty} T(h)x^h \\
&= qx + \sum_{h=2}^{\infty} (T(h-1)(1-q) + q(1-T(h-1)))x^h \\
&= qx + (1-q)\sum_{h=2}^{\infty} T(h-1)x^h + \sum_{h=2}^{\infty} qx^h - q\sum_{h=2}^{\infty} T(h-1)x^h \\
&= \sum_{h=1}^{\infty} qx^h + (1-q)xf(x) - qxf(x) \\
&= \sum_{h=1}^{\infty} qx^h + (1-2q)xf(x).
\end{aligned}$$

We can rewrite $f(x)$ as

$$\begin{aligned}
f(x) &= \frac{\sum_{h=1}^{\infty} qx^h}{1-x+2qx} \\
&= \frac{q \cdot (\frac{x}{1-x})}{1-x+2qx} \\
&= \frac{qx}{(1-x)\cdot(1-(1-2q)x)} \\
&= \frac{\frac{1}{2}}{(1-x)} - \frac{\frac{1}{2}}{(1-(1-2q)x)}
\end{aligned}$$

For every constant A such that $|Ax| < 1$ we have that $\sum_{h=0}^{\infty}(Ax)^h = 1/(1-Ax)$, therefore we can rewrite both $1/(1-x)$ and $1/(1-(1-2q)x)$ as formal power series, obtaining

$$f(x) = \frac{1}{2}\sum_{h=0}^{\infty} x^h - \frac{1}{2}\sum_{h=0}^{\infty}(1-2q)^h x^h = \sum_{h=1}^{\infty}\left(\frac{1}{2} - \frac{(1-2q)^h}{2}\right)x^h.$$

We conclude that $T(h) = \frac{1}{2} - \frac{(1-2q)^h}{2}$, as claimed. □

We have all the necessary tools to be able to take a big step forward towards the proof of Theorem 1.

Proposition 1. *Let $\mathbf{u}(x), \mathbf{f}(x), \mathbf{e}(x)$ and $\mathbf{c}(x)$ be random elements of R, in such a way that $\mathsf{wt}_H(\mathbf{u}) = w_u, \mathsf{wt}_H(\mathbf{f}) = w_f, \mathsf{wt}_H(\mathbf{e}) = w_e$ and $\mathsf{wt}_H(\mathbf{c}) = w_c$, respectively. Let $v \in \mathsf{supp}(\mathbf{c})$, define $\mathbf{s}(x) := \mathbf{u}(x)\mathbf{f}(x) + \mathbf{c}(x)\mathbf{e}(x)$ and consider the expression*

$$x^{-v}\mathbf{s}(x) = \mathbf{e}_i(x) + x^{-v}\mathbf{u}_i(x)\mathbf{f}_i(x) + \sum_{l \in \mathsf{supp}(\mathbf{c})\setminus\{v\}} x^{l-v}\mathbf{e}_i(x).$$

Assume that the random variables $\sum_{l\in\mathsf{supp}(\mathbf{c})\setminus\{v\}} x^{l-v}\mathbf{e}(x)$ are independently distributed, as l and v vary. Then, for $j \in [k-1]$, we have that $(x^{-v}\mathbf{s}(x))_j$ is distributed as a Bernoulli random variable with parameter $p = \mathbb{P}((x^{-v}\mathbf{s}(x))_j = 1)$ equal to:

$$p = \prod_{i=1}^{3} p_i + \sum_{i=1}^{3} p_i \prod_{j\in[3], j\neq i}(1-p_j), \tag{7}$$

where

$$p_1 = w_e/k, \qquad p_2 = (1-(1-2w_e/k)^{w_c-1})/2$$

and

$$p_3 = \frac{1}{\binom{k}{w_u}\binom{k}{w_f}} \sum_{\substack{1 \le l \le \min(w_u, w_f) \\ l \text{ odd}}} \binom{k}{l}\binom{k-l}{w_u-l}\binom{k-w_u}{w_f-l}.$$

Proof. Write

$$(\mathbf{d}(x))_j = \left(\sum_{v \in \mathsf{supp}(\mathbf{c})} x^{-v}\mathbf{s}(x) \right)_j = \sum_{v \in \mathsf{supp}(\mathbf{c})} \left(x^{-v}\mathbf{s}(x)\right)_j$$

Fix the value of $v \in \mathsf{supp}(\mathbf{c})$ and consider the summand $(x^{-v}\mathbf{s}(x))_j$. We can expand it as in (2) as

$$(x^{-v}\mathbf{s}(x))_j = (\mathbf{e}(x))_j + (x^{-v}\mathbf{u}(x)\mathbf{f}(x))_j + \left(\sum_{l \in \mathsf{supp}(\mathbf{c}) \setminus \{v\}} x^{l-v}\mathbf{e}(x) \right)_j. \tag{8}$$

The random variable $(\mathbf{e}(x))_j$ is Bernoulli distributed with parameter

$$p_1 = w_e/k.$$

Note that $(x^{l-v}\mathbf{e})_j$ and $(\mathbf{e}(x))_j$ are identically distributed, as the coefficient vector of the former is the right circular shift of the coefficient vector of the latter, by $l-v$ positions. Therefore, from Lemma 2 we obtain that $\left(\sum_{l \in \mathsf{supp}(\mathbf{c}) \setminus \{v\}} x^{l-v}\mathbf{e}(x)\right)_j$ is Bernoulli distributed with parameter

$$p_2 = (1 - (1 - 2w_e/k)^{w_c - 1})/2.$$

Finally, from Lemma 1 and Remark 1 we have that the random variable $(x^{-v}\mathbf{u}(x)\mathbf{f}(x))_j$ is Bernoulli distributed with parameter

$$p_3 = \frac{1}{\binom{k}{w_u}\binom{k}{w_f}} \sum_{\substack{1 \le l \le \min(w_u, w_f) \\ l \text{ odd}}} \binom{k}{l}\binom{k-l}{w_u-l}\binom{k-w_u}{w_f-l}.$$

Bearing in mind that we are considering events over $\mathbb{F}_2$, the sum of the three random variables is Bernoulli distributed with parameter

$$p = \prod_{i=1}^{3} p_i + \sum_{i=1}^{3} p_i \prod_{j \in [3], j \neq i} (1 - p_j),$$

i.e. $(x^{-v}\mathbf{s}(x))_j$ succeeds if either only one or all the three variables produce success. □

At this point the proof of Theorem 1 is straightforward.

Proof of Theorem 1. The statement directly follows from Prop. 2 and the aforementioned assumptions of independence of the random variables involved in the sum. □

In the settings of our attack, we are going to specialize Theorem 1 to the case where $p_1 \in \{0, 1\}$. This allows us to study the effect of the j-th coordinate of the vector $\mathbf{e}_i$ on the distribution of the j-th coordinate of $\mathbf{d}_i$. Notice that, in order to have a clear understanding of the behaviour of the j-th component of $\mathbf{d}(x)$, as defined in Eq. 3, we needed to sum the $(x^{-v}\mathbf{s}(x))_j$ random variables over all the $v \in \mathsf{supp}(\mathbf{c})$, where in this case the events are intended over $\mathbb{Z}$. While modelling this, we made the simplifying assumption that the random variables $x^{-v}\mathbf{u}_i(x)$ and $\sum_{l \in \mathsf{supp}(\mathbf{c}) \setminus \{v\}} x^l\mathbf{e}(x)$ are independently distributed, as v varies in the support of $\mathbf{c}$. Experimental results in support of this assumption will be provided, showing that this assumption still manages to capture satisfactorily the behaviour of $\mathbf{d}(x)$, and, even more importantly, it does not affect the outcome of the cryptanalysis. Figure 4 gives a visual representation of how Theorem 1 approximates the behaviour of $\mathbf{d}_i$.

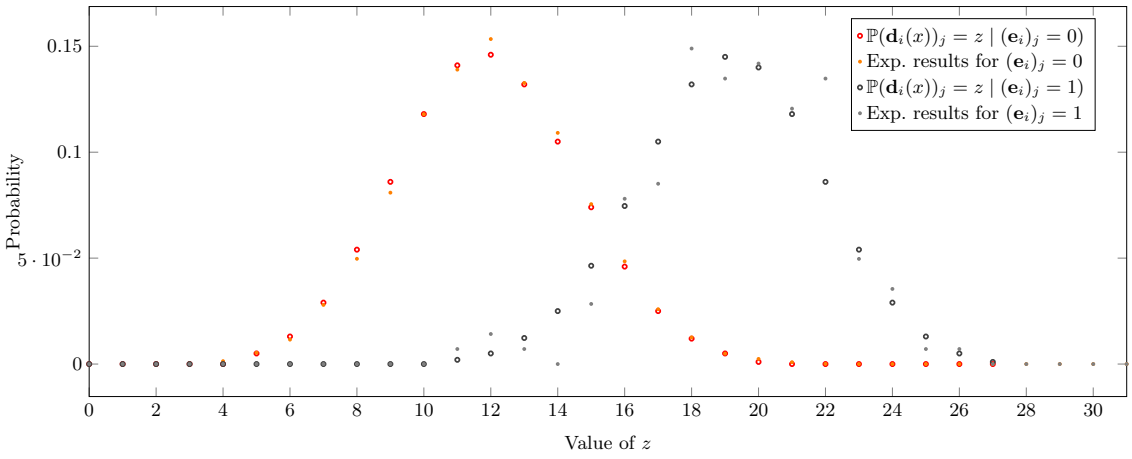

Fig. 4. In red (resp. dark gray) the theoretical estimates $\mathbb{P}(\mathbf{d}_i(x))_j = h \mid (\mathbf{e}_i)_j = 0)$ (resp. $\mathbb{P}(\mathbf{d}_i(x))_j = h \mid (\mathbf{e}_i)_j = 1)$), as h varies in the interval $[w_c]$, for security level 128. In orange (resp. light gray) the associated experimental results obtained for a random instance of HWQCS with 128 bit of security.

4.3 Recovering k zero bits of $\mathbf{e}$

As mentioned in Section 4.1, we aim at finding k zero entries in $\mathbf{e}$. The columns of $\mathbf{H}$ corresponding to the remaining k positions will constitute a square $k \times k$ matrix which, in the case of being invertible, can be used to compute the entire value $\mathbf{e} = (\mathbf{e}_1, \mathbf{e}_2)$. The idea is to use the knowledge of the distribution of the entries $(\mathbf{d}_i)_j$, provided in Theorem 1, and find $\lceil k/2 \rceil$ error free positions in both the left side $\mathbf{e}_1$ and the right side $\mathbf{e}_2$ of $\mathbf{e}$, adding up to (at least) k error free positions. We search for an optimal threshold value τ such that the probability $\mathbb{P}((\mathbf{e}_i)_j = 0 \mid (\mathbf{d}_i)_j < \tau)$ is large enough. For our attack, we choose τ as follows:

$$\tau := \max \left\{ h \in [w_c] \mid w_e \sum_{l=0}^{h} \mathbb{P}((\mathbf{d}_i)_j = l \mid (\mathbf{e}_i)_j = 1) < 1 \right\}, \tag{9}$$

i.e. the largest integer such that, for each signature, the expected number of entries j such that $(\mathbf{e}_i)_j = 1$ and $(\mathbf{d}_i)_j < \tau$ is less than 1. This means that, on average, all the coordinates j such that $((\mathbf{d}_i)_j) < \tau$ will be error free. Let N_0 and N_1 be the expected number of error free positions and error positions that we can find by taking all j with $(\mathbf{d}_i)_j < \tau$, respectively. Then

$$N_0 := (k - w_e) \sum_{h=0}^{\tau-1} \mathbb{P}((\mathbf{d}_i)_j = h \mid (\mathbf{e}_i)_j = 0),$$

and

$$N_1 := (w_e) \sum_{h=0}^{\tau-1} \mathbb{P}((\mathbf{d}_i)_j = h \mid (\mathbf{e}_i)_j = 1).$$

Therefore, the probability that $(\mathbf{e}_i)_j = 0$ for all $j \in [k]$ such that $(\mathbf{d}_i)_j < \tau$ is given by

$$p_{succ} := \prod_{j \in [k], (\mathbf{d}_i)_j < \tau} \mathbb{P}((\mathbf{e}_i)_j = 0 \mid (\mathbf{d}_i)_j < \tau) = \left(\frac{N_0}{N_0 + N_1} \right)^{N_0 + N_1}.$$

The values for τ, N_0, N_1 and of p_{succ} for each suggested parameter set of HWQCS are given in Table 2. These values indicate how many error free bits of $\mathbf{e}_i$ we can expect to recover with our strategy. Surprisingly enough, for security level 256 we can reconstruct more than $\lceil k/2 \rceil$ error free positions of $\mathbf{e}_i$. The same does not hold for levels 128 and 192, meaning that we need to guess $\lceil k/2 \rceil - N_0$ error free positions on each side of $\mathbf{e}$.
Table 6 reports the probability distribution and expected values using our approximation of $(\mathbf{d}_i)_j$ given by Theorem 1.

Security level	$\lceil \frac{k}{2} \rceil$	τ	N_0	N_1	p_{succ}	$\lceil \frac{k}{2} \rceil - N_0$
128	6270	12	5564.3997	0.3995	0.6707	706
192	9459	15	7574.2963	0.2521	0.7771	1885
256	12709	21	13560.6279	0.3457	0.7077	-

Table 2. For each security level, we report the threshold value τ, the values N_0 and N_1 of zero and one bits that we expect to find among the j's such that $(\mathbf{d}_i)_j < \tau$, and the probability p_{succ} that this event occurs. The value $\lceil \frac{k}{2} \rceil - N_0$ is the number of positions that we still need to guess to fully reconstruct $\lceil k/2 \rceil$ entries.

Remark 2. We are considering $\lceil k/2 \rceil$ in order to make sure we are recovering enough positions as k is an odd number for all three parameter sets. This implies that the probabilities we are computing are actually underestimates. Also, for security 256 we might select a subset of $\lceil k/2 \rceil$ of the N_0 expected error free positions, which would increase the value of the success probability to $p_{succ} = 0.7233$. We will exploit this in section 5.2 to give a speed-up of our attack.

Remark 3. The security level 256 would have values as reported in Table 3.

Security level	$\lceil \frac{k}{2} \rceil$	τ	N_0	N_1	p_{succ}
256	12709	22	16336.7984	0.8193	0.4407

Table 3. Value of τ as given in Equation 9, and the values of N_0 and N_1 of zero and one bits that we expect to find among the j's such that $(\mathbf{d}_i)_j < \tau$, as well as the probability p_{succ} that this event occurs.

As mentioned at the beginning of this subsection, we are searching for a total of $\lceil k/2 \rceil$ error free entries in $\mathbf{e}_i$. The value of N_0 for security level 256 with $\tau = 22$, accounts for many more positions than actually needed, with somewhat smaller success probability compared to the levels 128 and 192. Decreasing the value of τ to 21 still gives N_0 larger than $\lceil k/2 \rceil$ and increases the success probability by more than 0.25.

We are left with the problem of finding the missing $\lceil \frac{k}{2} \rceil - N_0$ zero positions of $\mathbf{e}_i$ for security levels 128 and 192. According to the behaviour of the $(\mathbf{d}_i)_j$ entries, we aim at finding these values among the j's such that $(\mathbf{d}_i)_j = \tau$, i.e. the set of positions that have the least probability of containing an error, after those we already chose. Let M_0 and M_1 be the expected number of error free and error positions j such that $(\mathbf{d}_i)_j = \tau$, respectively. Then

$$M_0 = (k - w_e)\mathbb{P}((\mathbf{d}_i)_j = \tau \mid (\mathbf{e}_i)_j = 0)$$

and

$$M_1 = (w_e)\mathbb{P}((\mathbf{d}_i)_j = \tau \mid (\mathbf{e}_i)_j = 1).$$

Therefore the probability such that $\lceil \frac{k}{2} \rceil - N_0$ randomly chosen positions j such that $(\mathbf{d}_i)_j = \tau$ will be error free is given by

$$q_{succ} := \left(\frac{M_0}{M_0 + M_1} \right)^{\lceil \frac{k}{2} \rceil - N_0}.$$

The values of M_0, M_1 and q_{succ} for levels 128 and 192 are reported in Table 4.
The overall probability of a correct recovery of $\lceil k/2 \rceil$ zero bits of $\mathbf{e}_i$ is then

$$p_{tot} := p_{succ} \cdot q_{succ}.$$

For the case of security level 256 we consider $q_{succ} = 1$ as $N_0 > \lceil \frac{k}{2} \rceil$. Applying this technique on both $\mathbf{e}_1$ and $\mathbf{e}_2$ we can recover k error free positions of $\mathbf{e}$ with probability of success p_{tot}^2. Values for each security level are reported in Table 5.

Security level	$\lceil \frac{k}{2} \rceil - N_0$	M_0	M_1	q_{succ}
128	706	1806.5588	0.7402	0.7491
192	1885	2433.8479	0.4371	0.7129

Table 4. For security parameters 128 and 192, $\lceil \frac{k}{2} \rceil - N_0$ denotes the number of entries which is still necessary to guess to perform our attack. Values M_0 and M_1 represent the expected number of error free and error positions among the j's such that $(\mathbf{d}_i)_j = \tau$.

Security level	k	p_{tot}^2
128	12539	0.2524
192	18917	0.3070
256	25417	0.5008

Table 5. For each security level, k and p_{tot}^2 are the number of error free positions we need to recover and the probability of recovering them correctly, respectively.

In other words, Table 5 says that we can successfully recover k error free coordinates of $\mathbf{e}$ of around one fourth of the signatures of security level 128, slightly less than one third of those of level 192 and of half of the signatures of level 256. In the next section, we show how to completely reconstruct $\mathbf{e}$ using the k error free positions we recovered in this section.

h	$\mathbb{P}((\mathbf{d}_i(x))_j = h \mid (\mathbf{e}_i)_j = 0)$	Exp. number of entries	$\mathbb{P}((\mathbf{d}_i(x))_j = h \mid (\mathbf{e}_i)_j = 1)$	Exp. number of entries
0	$3.078 \cdot 10^{-7}$	0.004	$1.248 \cdot 10^{-13}$	$1.759 \cdot 10^{-11}$
1	$5.935 \cdot 10^{-6}$	0.074	$6.220 \cdot 10^{-12}$	$8.770 \cdot 10^{-10}$
2	$0.006 \cdot 10^{-2}$	0.686	$1.500 \cdot 10^{-10}$	$2.115 \cdot 10^{-8}$
3	$0.003 \cdot 10^{-1}$	4.128	$2.331 \cdot 10^{-9}$	$3.286 \cdot 10^{-7}$
4	0.001	17.973	$2.623 \cdot 10^{-8}$	$3.698 \cdot 10^{-6}$
5	0.005	60.371	$2.277 \cdot 10^{-7}$	$0.003 \cdot 10^{-2}$
6	0.013	162.726	$1.586 \cdot 10^{-6}$	$0.002 \cdot 10^{-1}$
7	0.029	361.500	$9.109 \cdot 10^{-6}$	0.001
8	0.054	674.586	$0.004 \cdot 10^{-2}$	0.006
9	0.086	1072.334	$0.001 \cdot 10^{-1}$	0.025
10	0.118	1467.441	$0.006 \cdot 10^{-1}$	0.090
11	0.141	1742.590	0.002	0.276
12	0.146	1806.558	0.005	0.740
13	0.132	1642.366	0.0123	1.739
14	0.105	1313.477	0.025	3.595
15	0.074	925.951	0.0464	6.550
16	0.046	575.964	0.0746	10.530
17	0.025	316.115	0.105	14.937
18	0.012	152.936	0.132	18.678
19	0.005	65.088	0.145	20.545
20	0.001	24.292	0.140	19.818
21	0.0006	7.914	0.118	16.688
22	0.0001	2.237	0.086	12.195
23	0.00004	0.544	0.054	7.671
24	$9.109 \cdot 10^{-6}$	0.112	0.029	4.11
25	$1.586 \cdot 10^{-6}$	0.019	0.013	1.850
26	$2.277 \cdot 10^{-7}$	0.002	0.005	0.686
27	$2.623 \cdot 10^{-8}$	$0.003 \cdot 10^{-1}$	0.001	0.204

Table 6. Values of the distribution of $(\mathbf{d}_i(x))_j$ for security level 128. For every h, we report the probability that $(\mathbf{d}_i(x))_j = h$, conditioning on the event $(\mathbf{e}_i)_j = 0, 1$. Two more columns report the expected number of entries j associated to $(\mathbf{d}_i(x))_j = h$.

4.4 Completing the reconstruction of e

In this section, we exploit the analysis performed so far to fully reconstruct the ephemeral values $\mathbf{e}_1, \mathbf{e}_2$ with a certain probability. Recall that

$$\mathbf{b} = \mathbf{H}\mathbf{e}^\top, \tag{10}$$

where $\mathbf{e} = (\mathbf{e}_1, \mathbf{e}_2)$ and $\mathbf{H} = \left(\mathsf{circ}(\mathbf{h}), \mathsf{circ}(\mathbf{h})^{-1}\right)$.

Let $J \subset [2k]$ the set of k positions recovered using the strategy outlined in Section 4.3 and let $I := [2k] \setminus J$. Finally, let $\mathbf{H}_I$ be the submatrix of $\mathbf{H}$ which consists of the columns indexed by I. We can treat $\mathbf{H}_I$ as a random matrix in $\mathbb{F}_2^{k\times k}$. Assume that $\mathbf{H}_I$ is invertible. Then given the syndrome equation $\mathbf{b} = H\mathbf{e}^\top$ of $\mathbf{e}$ as in Sign, we can compute $\bar{\mathbf{e}} = \mathbf{H}_I^{-1}\mathbf{b}$ and thus reconstruct $\mathbf{e} = (e_0, \ldots, e_{2k-1})$ as

$$e_h = \begin{cases} \bar{e}_h & \text{if} \quad h = i \text{ for some } i \in I \\ 0 & \text{otherwise,} \end{cases} \tag{11}$$

for each $h \in [2k]$. The probability for a random $k \times k$ matrix to be invertible is given by Proposition 2.

Proposition 2. *Let $\mathbf{H} \in \mathbb{F}_2^{k\times k}$ be a random square matrix. The probability that $\mathbf{H}$ is invertible is given by*

$$p_{inv} := \prod_{i=0}^{k-1} \left(1 - \frac{1}{2^{k-i}}\right). \tag{12}$$

Proof. For $k = 1$ the result is trivial. Suppose k is greater than one. If we have $k-1$ independent vectors, the probability that the k-th vector is independent of the previous ones is $(2^k - 2^{k-1})/2^k$. Multiplying the probability for which these $k-1$ vectors are independent and the probability that the k-th vector is independent of the others yields

$$\frac{2^k - 2^{k-1}}{2^k} \cdot \prod_{i=0}^{k-2} \left(1 - \frac{1}{2^{k-i}}\right) = \prod_{i=0}^{k-1} \left(1 - \frac{1}{2^{k-i}}\right).$$

For each set of suggested parameters of HWQCS, the value of the probability of $\mathbf{H}_I$ to be invertible is $p_{inv} := 0.2888$.

Remark 4. The code used to compute data for this section can be found at https://github.com/triki96/Cryptanalysis-of-HWQCS

5 Universal Forgery

In this section, we exploit the reconstruction of $\mathbf{e}_1, \mathbf{e}_2$ to mount a universal forgery attack. Suppose an adversary $\mathcal{A}$ intercepts a signature $(\mathbf{c}, \mathbf{b}, \mathbf{s}_1, \mathbf{s}_2)$ used by a honest signer to sign a given message m. Note that:

$$\begin{aligned} \mathbf{b}(x) &= \mathbf{e}_1(x)\mathbf{h}(x) + \mathbf{e}_2(x)\mathbf{h}^{-1}(x), \\ \mathbf{c}(x) &= \mathsf{hash}_{w_c}(m, \mathbf{b}, \mathbf{u}_1(x)\mathbf{f}_2(x) - \mathbf{u}_2(x)\mathbf{f}_1(x), \mathsf{pk}), \\ \mathbf{s}_i(x) &= \mathbf{u}_i(x)\mathbf{f}_i(x) + \mathbf{c}(x)\mathbf{e}_i(x). \end{aligned}$$

Now, suppose the $\mathcal{A}$ is given a random message m', and it is asked to sign that message. The adversary needs $\sigma' = (\mathbf{c}', \mathbf{b}', \mathbf{s}_1', \mathbf{s}_2')$ which satisfies $\mathsf{Verify}(m', \mathsf{pk}, \sigma') = 1$. Therefore, $\mathcal{A}$ computes the signature in the following way:

$$\begin{aligned} \mathbf{b}'(x) &= \mathbf{b}(x), \\ \mathbf{c}'(x) &= \mathsf{hash}_{w_c}(m', \mathbf{b}', \mathbf{s}_1(x)\mathbf{h}(x) + \mathbf{s}_2(x)\mathbf{h}(x)^{-1} - \mathbf{c}(x)\mathbf{b}(x), \mathsf{pk}), \\ \mathbf{s}_i'(x) &= \mathbf{s}_i(x) - \mathbf{c}(x)\mathbf{e}_i(x) + \mathbf{c}'(x)\mathbf{e}_i(x) = \mathbf{u}_i(x)\mathbf{f}_i(x) + \mathbf{c}'(x)\mathbf{e}_i(x). \end{aligned}$$

The verifier computes

$$\begin{aligned}\mathbf{t}'(x) &= \mathbf{s}_1'(x)\mathbf{h}(x) + \mathbf{s}_2'(x)\mathbf{h}^{-1}(x) - \mathbf{c}'(x)\mathbf{b}'(x)\\ &= \mathbf{s}_1'(x)\mathbf{h}(x) + \mathbf{s}_2'(x)\mathbf{h}^{-1}(x) - \mathbf{c}'(x)\mathbf{b}(x)\\ &= (\mathbf{u}_1(x)\mathbf{f}_1(x) + \mathbf{c}'(x)\mathbf{e}_1(x))\mathbf{h}(x) + (\mathbf{u}_2(x)\mathbf{f}_2(x) + \mathbf{c}'(x)\mathbf{e}_2(x))\mathbf{h}^{-1}(x) +\\ &\quad - \mathbf{c}'(x)(\mathbf{e}_1(x)\mathbf{h}(x) + \mathbf{e}_2(x)\mathbf{h}^{-1}(x))\\ &= \mathbf{u}_1(x)\mathbf{f}_2(x) + \mathbf{u}_2(x)\mathbf{f}_1(x).\end{aligned}$$

Clearly $\mathbf{t}' \neq 0$ and $w(\mathbf{t}') \leq w_t$, as this is the same value computed in the verification phase of the first sign. It remains to check whether $\mathbf{c}'' := \mathsf{hash}_{w_c}(m', \mathbf{b}', \mathbf{t}', \mathsf{pk})$ is equal to $\mathbf{c}'$, but

$$\mathbf{c}'' = \mathsf{hash}_{w_c}(m', \mathbf{b}', \mathbf{t}', \mathsf{pk}) = \mathsf{hash}_{w_c}(m', \mathbf{b}, \mathbf{u}_1(x)\mathbf{f}_2(x) + \mathbf{u}_2(x)\mathbf{f}_1(x), \mathsf{pk}) = \mathbf{c}'.$$

This concludes the forgery.

Algorithm 5.1 is a probabilistic algorithm that sequentially analyses HWQCS signatures in order to retrieve the ephemeral values $\mathbf{e}_1, \mathbf{e}_2$ and forge a new signature for m'.

Algorithm 5.1 Attack on HWQCS

Input $\mathsf{pk} = (\mathbf{h})$ and τ according to Table 2.
Output True/False.

function ATTACKHWQCS($\mathbf{h}, \tau$)
  Compute $\mathbf{H} = \left(\mathsf{circ}(\mathbf{h}), \mathsf{circ}(\mathbf{h})^{-1}\right)$;
  Request $(m, \sigma) = (m, (\mathbf{c}, \mathbf{b}, \mathbf{s}_1, \mathbf{s}_2))$;
  Compute $\mathbf{d}_i(x) := \sum_{v \in \mathsf{supp}(\mathbf{c})} x^{-v}\mathbf{s}_i(x) \in \mathbb{Z}[x]$;
  Set
  - $J_{1,1} := \{j \in [k] \mid (\mathbf{d}_1)_j < \tau\}$ and
  - $J_{1,2} := \{j \in [k] \mid (\mathbf{d}_2)_j < \tau\}$;

  Set random
  - $J_{2,1} \subset \{j \in [k] \mid (\mathbf{d}_1)_j = \tau\}$ and
  - $J_{2,2} \subset \{j \in [k] \mid (\mathbf{d}_2)_j = \tau\}$

  of size $\lceil k/2 \rceil - \#J_1$;
  Set $I := [2k] \setminus (J_{1,1} \cup J_{1,2} \cup J_{2,1} \cup J_{2,2})$;
  Set $\mathbf{H}_I$ the submatrix of $\mathbf{H}$ made by columns of $\mathbf{H}$ indexed by I;
  if $\mathbf{H}_i$ is not invertible **then**
    Go to (3);
  else
    Compute $\bar{\mathbf{e}} = \mathbf{H}_I^{-1}\mathbf{b}$;
    Set $\mathbf{e}' = (e'_0, \ldots, e'_{2k-1}) = (\mathbf{e}'_1, \mathbf{e}'_2)$ as

$$\mathbf{e}'_h = \begin{cases} \bar{\mathbf{e}}_h & \text{if} \quad h = i \text{ for some } i \in I, \\ 0 & \text{otherwise.} \end{cases}$$

  end if
  Let m' be a new message to be signed;
  Compute $\mathbf{b}'(x) = \mathbf{b}(x)$;
  Compute $\mathbf{c}'(x) = \mathsf{hash}_{w_c}(m', \mathbf{b}', (\mathbf{s}_1(x)\mathbf{h}(x) + \mathbf{s}_2(x)\mathbf{h}^{-1}(x) - \mathbf{c}(x)\mathbf{b}'(x), \mathbf{h})$;
  Compute $\mathbf{s}'_i(x) = \mathbf{s}_i(x) - \mathbf{c}(x)\mathbf{e}'_i(x) + \mathbf{c}'(x)\mathbf{e}'_i(x)$;
  if $\mathsf{Verify}(m', \mathbf{h}, (\mathbf{c}', \mathbf{b}', \mathbf{s}'_1, \mathbf{s}'_2))$ **then**
    1. Return true;
  else
    1. Go to (3);
  end if
end function

5.1 Complexity of our attack

We estimate the success probability of Algorithm 5.1. The attack succeeds at reconstructing the data needed for a universal forgery if we can correctly obtain k error free positions of $\mathbf{e}$ and if the matrix $\mathbf{H}_I$, which depends on the recovered positions, is invertible. We computed the success probabilities p_{tot}^2 and p_{inv} of both the events in sections 4.3 and 4.4, respectively. Therefore, the success probability of Algorithm 5.1 is given by the product

$$p_{break} := p_{tot}^2 \cdot p_{inv},$$

and thus, the expected number of attempts to achieve success is given by $\lceil 1/p_{break} \rceil$. The cost of each attempt is dominated by that of inverting the matrix $\mathbf{H}_I \in \mathbb{F}_2^{k \times k}$, which is in $\mathcal{O}(k^{2.37})$. Therefore, Algorithm 5.1 runs in time $\mathcal{O}(2^{\log(k^{2.37}/p_{break})})$. The values of p_{break}, the expected number of needed attempts and the cost of the attack for each security level of HWQCS are reported in Table 7.

Security level	p_{break}	Number of signatures	Cost
128	0.0727	14	36.04
192	0.0887	12	37.25
256	0.1446	7	37.48

Table 7. For each security level, p_{break} denotes the probability that our attack successfully retrieves the ephemeral values $\mathbf{e}_1, \mathbf{e}_2$ from a given signature. The expected number of signatures for our attack to succeeds and the $\log_2$ of the cost of the attack are reported.

We implemented an unoptimized version of our attack in SageMath and ran it on a Linux Mint virtual machine. The code stops after recomputing $\mathbf{e}'$ and checking it against the real error vector $\mathbf{e}$. If the two match, the forgery succeeds. Table 8 reports the average, on 10 runs of the attack for each security level, of the number of signatures needed to recover $\mathbf{e}$ and the average time consumed analysing each signature.

Security level	Number of signatures	Consumed time (s)
128	17	40.33
192	9	92.50
256	3	133.25

Table 8. Average number of signatures needed to mount our universal forgery attack, and average time needed to analyse each signature, on 10 runs of our attack for each security level.

Remark 5. The code for our attack can be found in Appendix A. More code concerning our analysis can be found at `https://github.com/triki96/Cryptanalysis-of-HWQCS`.

5.2 BONUS: Breaking HWQCS security level 256 with one signature.

Due to the large value of N_0 as an effect of our choice of $\tau = 21$ for security level 256, see Table 2, it is possible to slightly improve on our attack by intercepting only one signature. Indeed, we can take advantage of this surplus of potentially error free positions that we obtain by taking all j such that $(d_i)_j < \tau$. The idea is to randomly pick a subset J_i of cardinality $\lceil k/2 \rceil$ of the $N_0 + N_1$ positions such that $(d_i)_j < \tau$ for both $i = 1, 2$. In other words, we are applying plain information set decoding on a potentially error free set of positions aiming at finding an invertible submatrix of H.

Note that J_i has probability of being error free equal to $\left(\frac{N_0}{N_0+N_1}\right)^{\lceil k/2 \rceil}$. Since we are doing this for both left and right-hand side of $\mathbf{e}$, the overall probability of $J := J_1 \cup J_2$ of being error free is $p := \left(\frac{N_0}{N_0+N_1}\right)^{k}$. According to the values in Table 2, we have $p = 0.5231$. Set now $I := [2k] \setminus J$ and let $\mathbf{H}_I$ be the matrix consisting of the columns of $\mathbf{H}$ indexed by I. As per proposition 2, the probability of $\mathbf{H}_I$ of being invertible is $p_{inv} = 0.2888$. In the case that $\mathbf{H}_I$ is invertible, we continue the same way as in our attack, leading to a forgery.

The success probability of recovering the correct $\mathbf{e}$ becomes then

$$p_{break} = p \cdot p_{inv} = 0.1511,$$

which gives an improvement of 0.065 on the probability 0.1446 reported in Table 7. The expected number of attempts of finding an invertible matrix $\mathbf{H}_I$ is $\lceil 1/p_{break} \rceil = 5$, which improves on table 7 by 2.

Remark 6. The SageMath code in Appendix A implements this *single signature* version of the attack.

Bibliography

[1] D. Augot, M. Finiasz, and N. Sendrier. A family of fast syndrome based cryptographic hash functions. In *International Conference on Cryptology in Malaysia*, pages 64–83. Springer, 2005.

[2] N. T. Courtois, M. Finiasz, and N. Sendrier. How to achieve a mceliece-based digital signature scheme. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 157–174. Springer, 2001.

[3] L. Dallot. Towards a concrete security proof of courtois, finiasz and sendrier signature scheme. In *Western European Workshop on Research in Cryptology*, pages 65–77. Springer, 2007.

[4] I. B. Damgård. A design principle for hash functions. In *Conference on the Theory and Application of Cryptology*, pages 416–427. Springer, 1989.

[5] T. Debris-Alazard, N. Sendrier, and J.-P. Tillich. Wave: A new family of trapdoor one-way preimage sampleable functions based on codes. In *Advances in Cryptology–ASIACRYPT 2019: 25th International Conference on the Theory and Application of Cryptology and Information Security, Kobe, Japan, December 8–12, 2019, Proceedings, Part I*, pages 21–51. Springer, 2019.

[6] T. Debris-Alazard and J.-P. Tillich. Two attacks on rank metric code-based schemes: Ranksign and an ibe scheme. In *Advances in Cryptology–ASIACRYPT 2018: 24th International Conference on the Theory and Application of Cryptology and Information Security, Brisbane, QLD, Australia, December 2–6, 2018, Proceedings, Part I 24*, pages 62–92. Springer, 2018.

[7] J.-C. Deneuville and P. Gaborit. Cryptanalysis of a code-based one-time signature. *Designs, Codes and Cryptography*, 88:1857–1866, 2020.

[8] G. D'Alconzo, A. Meneghetti, and P. Piasenti. Security issues of cfs-like digital signature algorithms. *Journal of Discrete Mathematical Sciences and Cryptography*, 27(1):175–187, 2024.

[9] A. Fiat and A. Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *Advances in Cryptology—CRYPTO'86: Proceedings 6*, pages 186–194. Springer, 1987.

[10] P. Gaborit, O. Ruatta, J. Schrek, and G. Zémor. Ranksign: an efficient signature algorithm based on the rank metric. In *International Workshop on Post-Quantum Cryptography*, pages 88–107. Springer, 2014.

[11] G. Kabatianskii, E. Krouk, and B. Smeets. A digital signature scheme based on random error-correcting codes. In *IMA International Conference on Cryptography and Coding*, pages 161–167. Springer, 1997.

[12] J.-L. Kim, J. Hong, T. S. C. Lau, Y. Lim, and B.-S. Won. Redog and its performance analysis. *Cryptology ePrint Archive*, 2022.

[13] V. Lyubashevsky. Fiat-shamir with aborts: Applications to lattice and factoring-based signatures. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 598–616. Springer, 2009.

[14] C. A. Melchor, N. Aragon, S. Bettaieb, L. Bidoux, O. Blazy, J.-C. Deneuville, P. Gaborit, E. Persichetti, G. Zémor, and I. Bourges. Hamming quasi-cyclic (hqc). *NIST PQC Round*, 2:4–13, 2018.

[15] R. C. Merkle. One way hash functions and des. In *Conference on the Theory and Application of Cryptology*, pages 428–446. Springer, 1989.

[16] A. Otmani and J.-P. Tillich. An efficient attack on all concrete kks proposals. In *International Workshop on Post-Quantum Cryptography*, pages 98–116. Springer, 2011.

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

[18] F. Ren, D. Zheng, W. Wang, et al. An efficient code based digital signature algorithm. *Int. J. Netw. Secur.*, 19(6):1072–1079, 2017.

[19] S. Ritterhoff, G. Maringer, S. Bitzer, V. Weger, P. Karl, T. Schamberger, J. Schupp, and A. Wachter-Zeh. Fuleeca: A lee-based signature scheme. *Cryptology ePrint Archive*, 2023.

[20] P. Santini, M. Baldi, and F. Chiaraluce. Cryptanalysis of a one-time code-based digital signature scheme. In *2019 IEEE International Symposium on Information Theory (ISIT)*, pages 2594–2598. IEEE, 2019.

[21] C. H. Tan and T. F. Prabowo. High weight code-based signature scheme from qc-ldpc codes. In H. Seo and S. Kim, editors, *Information Security and Cryptology – ICISC 2023*, pages 306–323, Singapore, 2024. Springer Nature Singapore.

[22] V. Weger, N. Gassner, and J. Rosenthal. A survey on code-based cryptography. *arXiv preprint arXiv:2201.07119*, 2022.

[23] K. Xagawa. Practical attack on racoss-r. *Cryptology ePrint Archive*, 2018.

Appendix A

```
from sage.doctest.util import Timer

# Level 128
#lam,k,wf,wu,we,wc,ws,wt,tau = 128,12539,145,33,141,31,4844, 4937,12

# Level 192
#lam,k,wf,wu,we,wc,ws,wt,tau = 192,18917,185,41,177,39,7450,7592,15

# Level 256
lam,k,wf,wu,we,wc,ws,wt,tau = 256,25417,145,51,191,51,10111,10216,21


F = GF(2)
R1.<y> = F[]
R.<x> = R1.quo(y^k - 1)

def weight(v):
  return sum(1 for i in v if i==1)

def getmultmatrix(R,P2,nrows):
  M = []
  for i in range(k):
    el = list((R*x^i).mod(P2))
    M.append(el + [0 for j in range(k-len(el))])
  M = matrix(Fqm,M)
  return M[:nrows,:]

def keygen():
  f = []
  for i in range(2):
    inv = False
    while not inv:
      P = Permutations(k)
      p = P.random_element()
      meta_pol = [1 for i in range(wf)] + [0 for i in range(k-wf)]
      meta_pol = [meta_pol[p[i]-1] for i in range(k)]
      fi = R(meta_pol)
      inv = fi.is_unit()
    f += [fi]

  return f[0].inverse_of_unit()*f[1], f


def sign(H,sk):
  while True:
    e = []
    u = []
    P = Permutations(k)
    for i in range(2):
      p = P.random_element()
      ei = [1 for i in range(we)] + [0 for i in range(k-we)]
      ei = [ei[p[i]-1] for i in range(k)]
      e += [ei]
    for i in range(2):
      p = P.random_element()
```

```
      ui = [1 for i in range(wu)] + [0 for i in range(k-wu)]
      ui = [ui[p[i]-1] for i in range(k)]
      u += [ui]
    e = vector(e[0]+e[1])
    b = H*e

    p = P.random_element()
    c = [1 for i in range(wc)] + [0 for i in range(k-wc)]
    c = [c[p[i]-1] for i in range(k)]
    s = [R(u[i])*sk[i]+ R(c)*R(list(e)[k*i:k*(i+1)]) for i in range(2)]
    if weight(s[1]) <= ws and weight(s[0]) <= ws and
            weight(R(u[0])*sk[1] + R(u[1])*sk[0]) <= wt:
      return s,b,c,e

pk,sk = keygen()
print("computing public matrix, might take a while ...")
h1 = matrix.circulant(list(pk))
H = h1.augment(h1.inverse())
print("done")

times = []
numsig = []
timer = Timer()
for i in range(1):
  broken = False
  count = 1;
  time = 0
  timer.start()
  while not broken:
    print(f"generate one signature and analyze it ... {count}")
    count +=1
    s,b,c,e = sign(H,sk)
    c = R1(c)
    stats0 = [ZZ(0)]*k
    stats1 = [ZZ(0)]*k
    p0s0=[]
    p1s0=[]
    p0s1=[]
    p1s1=[]
    for m in c.monomials():
      v = m.degree()
      s0v = s[0]*x^(k-v)
      s1v = s[1]*x^(k-v)
      ls0 = list(s0v)
      ls1 = list(s1v)
      for i in range(k):
        stats0[i] += ZZ(ls0[i])
        stats1[i] += ZZ(ls1[i])

    s0zeros = []
    s0ones = []
    s1zeros = []
    s1ones = []
    for i in range(k):
      if e[i] == 0:
        p0s0 += [i]
        s0zeros += [stats0[i]]
```

```
  else:
    p1s0 += [i]
    s0ones += [stats0[i]]

  if e[k+i] == 0:
    p0s1 += [i]
    s1zeros += [stats1[i]]

  else:
    p1s1 += [i]
    s1ones += [stats1[i]]



z0 = [i for i in range(k) if stats0[i]<tau]
print(len(z0),min(s0ones),len([i for i in s0ones if i<=tau]))

z1 = [k+i for i in range(k) if stats1[i]<tau]
print(len(z1),min(s1ones),len([i for i in s1ones if i<=tau]))

z013 = [i for i in range(k) if stats0[i]==tau]

z113 = [k + i for i in range(k) if stats1[i]==tau]

##### SEARCH  INVERTIBLE MATRIX
print("Start searching for an invertible submatrix")
Htmp = zero_matrix(k)
cols = []
if lam != 256:
  print("guess some/other columns to exclude on the left and on the right...")
  s0smpcoord = sample(range(len(z013)), ceil(k/2)-len(z0))
  s0smp = [z013[i] for i in s0smpcoord]
  cols0 = z0 + s0smp
  s1smpcoord = sample(range(len(z113)), k-ceil(k/2)-len(z1))
  s1smp = [z113[i] for i in s1smpcoord]
  cols = cols0 + z1 + s1smp
  print(len(z0), len(s0smp), len(z1),len(s1smp))
  sel = [i for i in range(2*k) if i not in cols]
  print("build the submatrix ...")
  Htmp = H[:,sel]
  print(f'Matrix is invertible : {Htmp.rank()==k}')
  if Htmp.rank() != k : continue
  print("Computing error vector...")
  etmp = Htmp.inverse()*b
  erec = [0]*(2*k)
  for i in range(k):
    erec[sel[i]] = etmp[i]
  broken = vector(erec) == e
  print(f'Found error vector is valid : {broken}')
  if broken:
    timer.stop()
    times += [timer.walltime]
    numsig += [count]

else:
  while not broken:
```

```
cols = []
coord = sample(range(len(z0+z1)), k)
for i in coord:
  cols += [(z0 + z1)[i]]
print("256 -- ",len(cols))
sel = [i for i in range(2*k) if i not in cols]
print("build the submatrix ...")
Htmp = H[:,sel]
print(f'Matrix is invertible : {Htmp.rank()==k}')
if Htmp.rank() != k : continue
print("Computing error vector...")
etmp = Htmp.inverse()*b
erec = [0]*(2*k)
for i in range(k):
  erec[sel[i]] = etmp[i]
broken = vector(erec) == e
print(f'Found error vector is valid : {broken}')
if broken:
  timer.stop()
  times += [timer.walltime]
  numsig += [count]
```