Converting Fixed-Length Binary Strings Into Constant Weight Words: Application on Post-Quantum Cryptography

Mariano López-García, David G. Farouk-Marei, and Enrique Cantó-Navarro

***Abstract*—This paper presents a new algorithm for encoding binary strings of fixed-length into a word of constant Hamming weight and constant length. The primary difference compared to previous publications is that the proposed algorithm can be directly applied to binary strings of fixed-length, without including a variable number of random bits needed to ensure the success of the encoding. The algorithm is useful for many post-quantum encryption schemes, in which a constant weight word is required. Additionally, a constant-time version of the algorithm is proposed, in order to mitigate information leakage vulnerabilities that could be exploited in potential timing attacks. An application based on a post-quantum Classic McEliece cryptosystem, using different security levels, is presented. Experimental results demonstrate the feasibility and correctness of our approach along with its advantages when compared with other solutions proposed in the past.**

***Index Terms*—Security and privacy, code-based cryptography, public key cryptosystem, post-quantum cryptosystems, theory of computation.**

I. Introduction

With the advent of quantum computers, featured by their outstanding computational capabilities, the security provided by the classical cryptographic algorithms is known to be compromised. Aware of this threat, in 2017 the NIST (National Institute of Standards and Technology) launched a call aimed at proposing the future post-quantum standardization scheme (PQC), by finding an efficient cryptographic standard resistant against attacks performed by quantum computers [1].

Classic McEliece is one of the candidate algorithms that is currently in fourth round [2]. Such post-quantum public-key encryption algorithm is suitable as a key-establishment method. It is based on the Niederreiter's dual version of McEliece's public key encryption (PKE) that, additionally, provides IND-CCA2 security [3], [4]. Thus, Classic McEliece is proposed as a confidence key encapsulation mechanism (KEM) when distributing cryptographic keys through non-secure communications channels.

BIKE and HQC are two additional post-quantum algorithms presented to the NIST standardization project, that are also being considered in fourth round. Both algorithms are based on structure codes, and are suitable as a general-purpose KEM. In July 2022, the NIST announced that at most one of these two candidates would be selected for standardization at the conclusion of the fourth round.

These three algorithms use a fixed-weight vector in one of the steps involved in the key generation, encapsulation or (de)capsulation stages. For instance, Classic McEliece begins the key encapsulation process generating a uniform random binary error-vector $e \in \mathbb{F}_2^n$ of weight t (the vector e has length n and a t number of '1'). Among other methods, the generation of such fixed-weight vector can be accomplished through either shuffling, rejection or a sorting method [5].

If the algorithm is employed just to encrypt an arbitrary plaintext or a message of length L (not used as a KEM, but as PKE), by using the Niederreiter's dual version, the plaintext should have the same features (length n and weight t) as the binary vector e. Hence, if any plaintext is accepted, a decoder must be included in order to transform such message into a fixed-length constant weight word. Note that, when the algorithm is wrapped within a KEM then, instead of using a shuffling method or other, the vector e could be generated straightforwardly by creating a random word of length L, and including the previously mentioned decoder to meet the requirements in terms of length and weight.

There are some other applications in which it is necessary to include an algorithm able of transforming a binary sequence into a constant weight word. For instance, in the proposal made by Kobara and Imai [6] to obtain IND-CCA2 security in a PKE scheme, a function is included in order to convert an integer into its corresponding error vector of length n and weight t. In fact, LEDACrypt, one of the proposals submitted to the NIST PQC, and considered as candidate until second round [7], [8], employs this conversion.

The challenge of creating, from an arbitrary string of length L, a new binary sequence with a predetermined length n and Hamming weight t was approached from two different perspectives. The first solution is based on a combinational approach, and it consists in mapping the set of binary words of length n

Received 9 November 2023; revised 20 December 2024; accepted 29 December 2024. Date of publication 31 December 2024; date of current version 15 May 2025. This work was supported by MCIN/AEI/ under Grant PID2019-107274RB-I00, Grant PID2023-148649OB-I00 and Grant 10.13039/501100011033. *(Corresponding author: Mariano López-García.)*

Mariano López-García and David G. Farouk-Marei are with the Department of Electronic Engineering, Universitat Politècnica de Catalunya, 08800 Vilanova i la Geltrú, Spain (e-mail: mariano.lopez@upc.edu).

Enrique Cantó-Navarro is with the Department of Electrical and Automatic Electronic Engineering, Universitat Rovira i Virgili, 43007 Tarragona, Spain.

This article has supplementary downloadable material available at https://doi.org/10.1109/TDSC.2024.3524626, provided by the authors.

Digital Object Identifier 10.1109/TDSC.2024.3524626

and weight t with the integers of the interval $[0, \binom{n}{t}[$ [9]. Note that, this encoding considers that $L = \log_2 \left\lceil \binom{n}{t} \right\rceil$. The main drawback of this map is that it requires the calculation of large binomials coefficients, which have a quadratic complexity in the input string length L. The second approach was presented by Sendrier [10], proposing a run-length encoding based on Golomb codes that provides linear complexity. However, the length L of the input string varies depending on its value, and then it is not clear how to stablish a lower and upper bound for such length. Hence, in those applications in which L is fixed to a constant value, the use of this solution could not be straightforward.

Encoding and decoding of constant weight-words has also been applied to code-based signature. In [20], authors proposed the implementation of a signature scheme that provides resistance against side channel attacks. Sendrier's algorithm is used to convert an input hash value into a n-bit vector with constant weight. Similarly, a new method to construct code-based signatures following the Lyubashevsky's lattice-based framework was proposed in [21]. Authors used the so-called weight restricted hash functions, which are created by combining collision-resistant hash functions with an encode algorithm that is able of converting any message into a new string with fixed Hamming weight.

Physically Unclonable Functions (PUF) can be used as an authentication mechanism without the need to store secret keys. However, PUFs can be affected by modelling attacks, where malicious adversaries use machine learning to replicate the PUF. In [22], a new method, that obfuscates the challenges using XOR operations followed by a transformation into constant weight vectors, is proposed. This transformation is based on the enumerative encoding method (combinational approach) introduced by Cover [9].

Constant weight encoding was also used in password-authenticated key exchange (PAKE) protocols. In [23], authors published the first post-quantum PAKE protocol based on Ouroboros, a scheme that gathers the best properties of the McEliece variant that uses moderate density parity-check codes and Hamming Quasi-Cyclic cryptosystems. As in [20], this proposal uses the Sendrier's algorithm to create random words with constant Hamming weight.

This papers presents a new algorithm aimed at creating constant weight words that can be incorporated in some of the algorithms presented to the NIST post-quantum standardization project. This algorithm is specially well suited for cryptographic algorithms based on correction error-codes. When compared to previous publications, our main contribution is that the algorithm works when the input string has a predetermined fixed-length L; it has a linear computation complexity in L; and it runs in constant time, so timing attacks can be avoided.

Next section resumes the main related works. Section III describes our proposed algorithm for a constant weight word encoding, and presents an implementation that allows its execution in constant time. Finally, experimental results are reported and commented in Section IV.

II. Related Works and Problem Statement

A. Notations and Conventions

For all post-quantum algorithms based on correction error-codes, n denotes the code length and t is the error-correction capability. $C = \{1,0\}^L$ represents the input binary string of length L that must be converted into a binary word $\lambda = \{1,0\}^n$ of length n and Hamming weight t. Additionally, such binary word λ can by represented by a t-tuple $\delta = \{\delta_1, .., \delta_k, .., \delta_t\}$ formed by t integers δ_k, being $\delta_k \in \mathbb{N}$, $1 \leq k \leq t$. Each integer δ_k represents the number of zeros existing between each two consecutive ones included in λ. For instance, if $\lambda = \{01000100001001001\}$, where $t = 5$ and $n = 17$, then $\delta = \{1,\ 3,\ 4,\ 2,\ 2\}$. Additionally, λ can also be represented by a different t-tuple of integers $\gamma = \{\gamma_1..\gamma_k..\gamma_t\}$, being $\gamma_k < \gamma_j$ if $k < j$. Each γ_k represents the coordinates (index) of the '1's in λ. Using the same example, γ would be $\{2,\ 6,\ 11,\ 14,\ 17\}$. Note that $\delta_k = \gamma_k - \gamma_{k-1} - 1$ being $\gamma_o = 0$. It is easy to prove that $\sum_{k=1}^{k=t} \delta_k \leq (n-t)$. Additionally, $Gol(\delta_k)$ is the Golomb coding of a positive integer δ_k. The characteristic of the Golomb coding is denoted as d. Bin2CW is the algorithm that transforms a binary string C into the constant weight word represented by the t-tuple δ. CW2Bin is the opposite function, i.e., it converts a constant weight word into a binary string C of length L. Finally, $W_{n,t}$ denotes the set of binary words of length n and Hamming weight t.

B. Previous Proposals

At the early 70s, Cover proposed one of the first solutions to fix the problem of converting binary strings into constant weight words [9]. This proposal is based on a combinatorial number system, which is a mixed representation of natural numbers, known as combinadics. As any non-negative integer i_s can be represented uniquely as sum of t binomial coefficients, Covers employed a simple method consisting in indexing the set $W_{n,t}$ using the following mapping:

$$\{\gamma_1..\gamma_j..\gamma_t\} \rightarrow i_s = \binom{\gamma_t - 1}{t} + .. + \binom{\gamma_1 - 1}{1} \quad (1)$$

For instance, if $\lambda = \{1010001\}$, then $\gamma = \{1,\ 3,\ 7\}$ ($t = 3$ and $n = 7$) and $i_s = 21$ (i.e., $C = \{10101\}_2$). The disadvantage of this method is that the computation of i_s involves the calculation of t binomial coefficients, which usually leads to a lower performance in terms of execution time. In [30], Borisenko et al. proposed a very efficient algorithm for generating binomial coding words. By using a simple transformation that adds a certain number of zeros or ones to the end of the word, the binomial coding combinations can be easily converted into constant weight words. Authors proved as the binomial numerical system is a prefix code, it is well-defined and it is compact. However, no additional information about how to find the inverse function is given, i.e., finding the unique integer that corresponds to the constant weight word, so that it would be necessary to calculate t binomials coefficients as in the method proposed by Cover.

Sendrier proposes in [10] a method that avoids the calculation of binomial coefficients. Let C be the plaintext of length L that should be converted into a constant weight word such as δ (or λ).

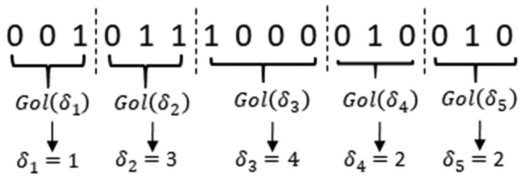


Fig. 1. Representation of the input string C as a concatenation of t=5 Golomb codewords ($d = 4$).

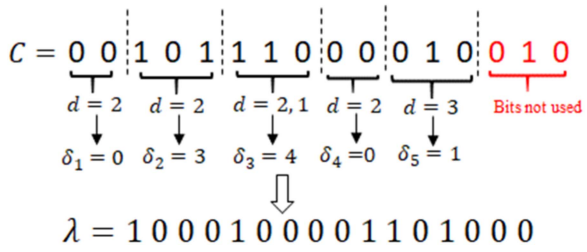


Fig. 2. Conversion of string C into a constant weight word using Sendrier's algorithm ($t = 5$, $n = 16$).

Algorithm 1: Binary String to Constant Weight Word (Bin2CW).

Input: length n, weight t, binary string C
Result: a t-tuple $\delta = \{\delta_1, .., \delta_k, .., \delta_t\}$

```
index=0, delta=0
while (t>0) do
    if n≤t then
        t--, n--
        δ[index]=delta
        delta=0, index++
    else
        d=best_d(n,t)
        if read(C,1) then
            n-=d, delta+=d
        else
            i=decodefd(d,C)
            δ[index]= δ+i
            n-=(i+1), t--
            delta=0, index++
return (δ1,δ2,.., δt)
```

Algorithm 2: Constant Weight Word to Binary String (CW2Bin).

Input: length n, weight t, t-tuple $\delta = \{\delta_1, .., \delta_k, .., \delta_t\}$
Result: a Binary string C

```
index=1
while (t!=0 and n>t) do
    d=best_d(n,t)
    if δ[index]≥d then
        n-=d
        δ[index]-=d
        write(C,1,1)
    else
        write (C,0,1)
        u=log2(d)
        write (C, encodefd(δ[index]),u)
        n-= δ[index]+1
        t--, index ++
return C
```

As shown in Fig. 1, Sendrier proposes interpreting this plaintext as the concatenation of t Golomb codewords, each one representing the codification of a particular δ_k. For instance, using the example presented before (i.e., $\delta = \{1, 3, 4, 2, 2\}$), the plaintext would be $C = 0010111000010010$, if the characteristic parameter of the Golomb code $d = 4$ and $L = 16$. Note that in this example $L \neq n$, which is the usual case.

Algorithms 1 and 2 represent the unfolded versions of Sendrier's algorithms as proposed in [12]. Algorithm 1 shows the pseudocode used to obtain the t-tuple δ from the input string C (i.e., Bin2CW), while Algorithm 2 represents the pseudocode that performs the inverse transformation (i.e., CW2Bin).

As Golomb mentioned in [11], the value of d is chosen to achieve the best compression rate. Sendrier proposed to adjust this parameter in each recursive call used to calculate the value of δ_k. Thus, the function $best\_d(n, t)$ returns the optimal value of d, that should be used in each iteration to obtain the maximum compression rate bounded, in average, by the entropy $\log_2(\binom{n}{t})$. Such best value is given by

$$d \approx \left(n - \frac{t-1}{2}\right)\left(1 - \frac{1}{2^{\frac{1}{t}}}\right) \quad (2)$$

In fact, in the experimental results presented in [10], the efficiency defined as the quotient between L and the entropy is close to 100%, which demonstrates the correctness when the value of d is chosen as defined in (2).

The function $write\ (C, j, k)$, writes the integer j, using k bits, into the string C and move forward the writing pointer k positions. Analogously, $read(C, k)$ returns the integer value of the following k bits of the string C and it advances the reading pointer by k positions. Function $decodefd(d, C)$ returns the integer value δ_k that represents the codeword $Gol(\delta_k)$ using d as parameter of the Golomb code, while $encodefd(\delta_k, d)$ performs the opposite operation. The result of executing Algorithm 1 (Bin2CW), on the string C shown in Fig. 1, is depicted in Fig. 2. As can be seen, the last 3 bits are not used by the mapping function, because the condition $t > 0$ is violated in the ninth iteration of the *while- loop*. The opposite circumstance can also occur, namely, there may not be enough bits in C to complete the conversion. Thus, it can be concluded that the number of bits needed to create the t-tuple δ varies depending on the value of those bits that form the binary sequence C.

In fact, Bin2CW described in Algorithm 1, is defined as a function $\Phi(C) : \{1, 0\}^* \rightarrow \delta = \{\delta_1, .., \delta_k, .., \delta_t\}$ that performs a mapping between a set of binary strings and a set of t natural integers ranged from 0 to $(n - t)$. Table I shows this function for $n = 4$ and $t = 2$. Note as, $\Phi(C)$ is a partial function that is bijective on the elements δ which do have an image. Again, as this table reveals, the domain of this function is a finite set of bit strings whose length varies between 2 and 3 bits. Thus, as it was pointed out by Sendrier, if any binary sequence C (e.g., $C = ''00''$) should be decoded into a sequence λ of length

TABLE I
FUNCTION $\Phi(C)$: $N = 4$, $T = 2$

Input String C	t-tuple δ	λ constant weight word	Length L of input string
"11"	$\delta = \{2,0\}$	λ = "0011"	2 bits
"101"	$\delta = \{1,1\}$	λ = "0101"	3 bits
"001"	$\delta = \{1,0\}$	λ = "0110"	3 bits
"10"	$\delta = \{0,2\}$	λ = "1001"	2 bits
"100"	$\delta = \{0,1\}$	λ = "1010"	3 bits
"000"	$\delta = \{0,0\}$	λ = "1100"	3 bits

n and weight t, it might be necessary to add a few bits on C (e.g., $C = ''000''$ or $C = ''100''$) in order to complete the last letter of the t-tuple δ. In [8], Barenghi et al. stated the problem from a similar point of view: the only way to determine if the input string C has an image, i.e., $\Phi(C) \neq \perp$, is computing the inverse $\Phi^{-1}(\delta)$ on all combinations.

Additionally, as the function $best\_d(n,t)$ is recalculated in each iteration, it is almost impossible to find analytically a lower and upper bound for the number of bits L. Besides, note as the value returned by (2) is a floating-point number, which is rounded by the closest integer. The author proposed a simplification by constraining the value of d to a power of two. This way, the implementation of functions $decode fd(d,C)$ and $encode fd(\delta_k, d)$ is easier, leading to a reduction of the execution time while the loss of coding efficiency is very limited.

In [13], [14] Heyse et al. used this approximation for d. Additionally, to speed up the algorithm, the set of values taken by d are store in a lookup table. However, this approach is challenging to implement in applications in which the value of n is high, since the memory needed to store such data is approximately $\mathcal{O}(n)$.

Hu et al. proposed in [12] an implementation of the Sendrier's algorithms on FPGA, performing the calculation of $best\_d(n,t)$ using fixed-point arithmetic. Additionally, their implementation uses a BRAM memory just to store some pre-computed data.

An improved solution, that addresses the problem related to the variability of L in the string C, was proposed by Barenghi et al. in [8]. The value of d is not recalculated in each round, being always constant. Now, a new string C', that consists in the concatenation of two sub-strings, is defined. The first sub-string has a fixed-length $L_{\min}$, and it corresponds to the shortest number of bits needed to decode C' as the t-tuple δ. The second sub-string is formed by adding as many random bits as needed to guarantee that $\sum_{k=1}^{k=t} \delta_k \leq (n-t)$. Thus, authors conclude that if the value of d is bounded by

$$2^{(L_{\min}/t)-1} - 1 \leq d \leq \frac{n-t}{L_{\min}} \tag{3}$$

any C' can be successfully decoded. Note that, if $L_{\min}$ is chosen in such a way that it matches the length of the string C, then all bits of C will contribute to the calculation of the t-tuple δ. The drawback of this proposal is that, depending on the values of $L_{\min}$ and d, the efficiency decreases as n increases. Table II shows an example based on a Classic McEliece KEM for different values of d restricted to a power of two.

TABLE II
EFFICIENCY AND PARAMETERS NEEDED USING THE METHOD PROPOSED IN [8]

Value d	Value of n	Efficiency $= L_{min}/log_2\binom{n}{t}$
32	$n \geq 16{,}992$	0,621
64	$n \geq 33{,}888$	0,558
128	$n \geq 67{,}680$	0,506
256	$n \geq 135{,}264$	0,464

Classic McEliece t = 96, $L_{\min}$ = 528 category 3.

Thus, the main shortcomings of previous approaches can be summarized in: i) the high computational cost when many binomials coefficients need to be calculated, ii) the low efficiency of some transformations, and iii) the input string C, that should be converted into a constant weight word, has a variable bit length. Note that, these limitations may become a problem in some applications. For instance, in [27], facial biometric features are protected by using a Classic McEliece post-quantum algorithm. Such features are represented by a binary string $b = \{0,1\}^K$ of length K. As b can be any of the 2^K possible combinations, its Hamming weight does not match the value of t, so that a direct encryption obtained by multiplying the parity check matrix H and b is not possible. Note that, the length of the biometric feature is fixed to K, and all bits of b (the whole biometric feature) must be used to obtain $\delta = \{\delta_1, .., \delta_k, .., \delta_t\}$. If, in addition to these constraints, a low computational cost and a high efficiency is demanded, previous approaches are not suitable solutions. Instead, the encoding algorithm proposed in this paper is designed to fulfil all these requirements.

III. PROPOSED ALGORITHM

A. Fundamentals of the Proposal

The aim of our proposed algorithm is that the number of bits used to create the constant weight word λ can be fixed in advance to $L_{\min}$. Such a value only depends on n and t. As proven in [8], if d was always constant, then the following inequality must hold if at least $L_{\min}$ bits must be decoded

$$t \cdot (1 + \lfloor \log_2(d) \rfloor) \geq L_{\min} \tag{4}$$

so that the minimum value for $d_{\min}$ is given by

$$d_{\min} = 2^{\lceil \frac{L_{\min}}{t} \rceil - 1} \tag{5}$$

Note as (4) only ensures that, even in the worst case given by $C = \{0\}^{L_{\min}}$, at least $L_{\min}$ bits will be decoded. However, since we are interested in the particular case in which this number is constant and fixed to $L_{\min}$, the following strategy is applied. Let $ReadBits$ be the number of bits used to decode the first k Golomb codewords $Gol(\delta_j)$, $j = 1..k$. Thus, once $Gol(\delta_1)$ is decoded $ReadBits = (1 + \log_2(d_{\min}))$. Then, in the following iteration of the *while-loop*, the values of t and $L_{\min}$ are updated to $(t-1)$ and $(L_{\min} - ReadBits)$, respectively. Next, $d_{\min}$ is re-calculated again using the updated values, so that it is guaranteed that at least $(L_{\min} - ReadBits)$ bits will be used to decode the remaining set of codewords $Gol(\delta_k)$ $(2 \leq k \leq t)$ that still form the string C. The same strategy is applied repetitively, until all the $Gol(\delta_k)$ are decoded. Algorithm 3 shows the modifications

Algorithm 3: Proposed Binary String to Constant Weight Word (Bin2CW) – Non Constant Time Version.

Input: length n, weight t, L_{min}, string C of length L_{min}
Result: a t-tuple $\delta = \{\delta_1, .., \delta_k, .., \delta_t\}$

```
index=0, delta=0, ReadBits=0
while (t>0) do
    if (L_min-ReadBits<=0) then
        t--, n--
        δ[index]=delta
        delta=0, index++
    else
        d=dmin(t,(L_min-ReadBits))
        if read(C,1) then
            n-=d, delta+=d
        else
            i=decodefd(d,C)
            δ[index]= δ+i
            n-=(i+1), t--
            delta=0, index++
            ReadBits+=1+log2(d)
return (δ1,δ2,.., δt)
```

Algorithm 4: Calculation of the Minimum Value for d as Indicated in (5).

Input: weight t', length L' (L'>0, t'>0)
Result: value for $d_{\min}$

```
1 index = ceil(L'/t')-1 //ceil(x) → returns the smallest integer > = x
2 dmin = 1<<index //shift-left
3 return dmin
```

that have been included in the pseudocode. Now, $best\_d(n,t)$ is substituted by the function $d_{\min}(t,(L_{\min} - ReadBits))$. As defined in (5), this function returns the minimum value that d should take. Its implementation is described in Algorithm 4.

As $d_{\min}$ is by definition always a power of two, the implementation of functions $encodefd$ a $decodefd$ is easier and, additionally, their computation is faster. Now, let's find what are the relationships between n and t in order to ensure that Algorithm 3 always works correctly for any input string of length $L_{\min}$. The main constraint is derived from the fact that the elements $\delta_k \in \mathbb{N},\ 1 \le k \le t$ of any t-tuple $\delta = \{\delta_1, .., \delta_k, .., \delta_t\}$ meet that $\sum_{k=1}^{k=t} \delta_k \le (n-t)$.

Lemma 1: The all ones bit-string represented as $C = \{1\}^{L_{\min}}$ is decoded as $\delta = \{\delta_{\max},\ 0,\ 0, \ldots, 0\}$, where

$$\delta_{\max} = \left(L_{\min} + 2t - t\left\lceil \frac{L_{\min}}{t} \right\rceil\right) 2^{\lceil \frac{L_{\min}}{t} \rceil - 1} - t \quad (6)$$

Proof: As C is an all one bit-string then only $\delta_{\max} \neq 0$, being $\delta_k = 0$ $(2 \le k \le t)$. As in each run of the *while-loop* only one bit of C is read, for each iteration the value of $L_{\min}$ decreases in one unit. That leads to change $d_{\min}$ each t iterations, except for the first ones, depending on if $L_{\min}$ is an integer multiple of t. Let r, $(1 \le r \le t)$, be the first ones iterations, then

$$r = t\left(1 - \left(\left\lceil \frac{L_{\min}}{t} \right\rceil - \left\lfloor \frac{L_{\min}}{t} \right\rfloor\right)\right) + \left(L_{\min} - t\left\lfloor \frac{L_{\min}}{t} \right\rfloor\right)$$
$$= L_{\min} + t \cdot \left(1 - \left\lceil \frac{L_{\min}}{t} \right\rceil\right) \quad (7)$$

If N_{iter} is the total number of iterations, as $C = \{1\}^{L_{\min}}$, then $N_{iter} = L_{\min}$. Additionally, as $d_{\min}$ is recalculated in each iteration as indicated in (5), then we have that

$$\delta_{\max} = \sum_{j=1}^{j=L_{\min}} d_j = r2^{\lceil \frac{L_{\min}}{t} \rceil - 1} + \underbrace{t2^{\lceil \frac{L_{\min}}{t} \rceil - 2} + \ldots + t2^1 + t2^0}_{\lceil \frac{L_{\min}}{t} \rceil - 1} \quad (8)$$

Applying that $2^\alpha - 1 = \sum_{j=0}^{j=\alpha-1} 2^j$, we conclude that

$$\delta_{\max} = (r+t)\ 2^{\lceil \frac{L_{\min}}{t} \rceil - 1} - t \quad (9)$$

By substituting (7) in (9), expression (6) is obtained (q.e.d.).

Note as it is still necessary to ensure that $\sum_{k=1}^{k=t} \delta_k \le (n-t)$ for any $C = \{1\}^{L_{\min}}$. Hence, as $\delta_1 = \delta_{\max}$ and $\delta_k = 0$ $(2 \le k \le t)$, then it should hold that $\delta_{\max} \le (n-t)$. Thus, using (6) the minimum value for n can be obtained

$$n_{\min} = \delta_{\max} + t = \left(L_{\min} + 2t - t\left\lceil \frac{L_{\min}}{t} \right\rceil\right) 2^{\lceil \frac{L_{\min}}{t} \rceil - 1} \quad (10)$$

which is the most important (and the only) constraint of the proposed algorithm.

Lemma 2: If a bit-string C_1 can be decoded as $\delta = \{\delta_1, \delta_2, 0, .., 0\}$, then the following relation holds

$$\delta_1 + \delta_2 \le \delta_{\max} \quad (11)$$

Proof: A general string C_1, decoded as $\delta = \{\delta_1, \delta_2, 0, \ldots, 0\}$, has a general form represented by

$$C_1 = \underbrace{\underbrace{11\ldots1}_{r}\ \underbrace{0x..xxxxx}_{1+\log_2(d_1)}}_{Gol(\delta_1)}\ \underbrace{\underbrace{11..1xxxx.xxx}_{s}}_{Gol(\delta_2)} \quad (12)$$

where "x" stands for bits that can be either one or zero. As (12) shows, string C_1 can be seen as the concatenation of two substrings, $Gol(\delta_1)$ and $Gol(\delta_2)$, respectively. Note as the highest values for δ_1 and δ_2 (worst case in (11)) are obtained when those bits marked as "x" are all ones. Thus, let's consider such a case as the worst one, so C_1 becomes

$$C_1 = \underbrace{11\ldots1}_{r}\ \underbrace{01..1111}_{1+\log_2(d_1)}\ \underbrace{111111\ldots.1}_{s} \quad (13)$$

For the sake of clarity, the demonstration that the Golomb decoding of C_1 leads to a value of $\delta_1 + \delta_2$ that complies (11) is annexed in Appendix 1.

Lemma 3: If a bit-string C_2 can be decoded as $\delta = \{\delta_1, \delta_2, \delta_3, 0, \ldots, 0\}$, then the following relation holds

$$\delta_1 + \delta_2 + \delta_3 \le \delta_{\max} \quad (14)$$

Proof: Now, the string C_2 can be represented as the concatenation of two sub-strings

$$C_2 = C_{21}||C_{22} = \underbrace{\underbrace{1..10x..xx}_{w}}_{C_{21}} \underbrace{\underbrace{1..10x..xx}_{r}\underbrace{1..10x..xx}_{s}}_{C_{22}} \quad (15)$$

Let C_{22} be the sub-string that contributes to create the elements δ_2 and δ_3 of the t-tuple δ, through the set of bits r and s, respectively, and C_{21} generates the element δ_1 from the first w bits. Note as C_{22} plays the same role as C_1 in (12). As it is stated in (13), the worst case for C_{22}, namely C'_{22}, is given when its contribution on $(\delta_2 + \delta_3)$ is maximum, then

$$C'_{22} = \underbrace{\underbrace{11\ldots1}_{r'}\underbrace{01..1111}_{1+\log_2(d_2)}}_{Gol(\delta'_2)}\underbrace{\underbrace{111111\ldots.1}_{s'}}_{Gol(\delta'_3)} \quad (16)$$

where δ'_2 and $\delta_3{}'$ represent the decoding of $C_{22}{}'$. Since $C_{22}{}'$ is the worst case, then $\delta_2 + \delta_3 \leq \delta_2{}' + \delta_3{}'$ and, as $C_2 = C_{21}\ ||C_{22}{}'$, then it also holds that $\delta_1 + \delta_2 + \delta_3 \leq \delta_1 + \delta_2{}' + \delta_3{}'$.

Additionally, we know that if C'_{22} was an all one bit-string, namely $C''_{22} = (1)^{r+s}$, then from (13) the worst case scenario for C_2 would be now

$$C_2 = C'_{21}||C''_{22} = \underbrace{\underbrace{11\ldots1}_{w'}\underbrace{01..1111}_{1+\log_2(d_3)}}_{Gol(\delta''_1)}\underbrace{\underbrace{111111\ldots.1}_{r+s}}_{Gol(\delta''_2)} \quad (17)$$

Now, if $\delta_1{}''$ and $\delta_2{}''$ represents the new encoding of C_2, then $\delta_1 + \delta_2{}' + \delta_3{}' \leq \delta_1{}'' + \delta_2{}''$. From Lemma 2, it holds that $\delta_1{}'' + \delta_2{}'' \leq \delta_{\max}$. Thus, it holds that $\delta_1 + \delta_2 + \delta_3 \leq \delta_1 + \delta_2{}' + \delta_3{}' \leq \delta_1{}'' + \delta_2{}'' \leq \delta_{\max}$ (q.e.d.).

Theorem: For any arbitrary bit-string C dedoded as $\delta = \{\delta_1, \delta_2, .., \delta_t\}$ always holds that

$$\sum_{k=1}^{k=t} \delta_k \leq \delta_{\max} \quad (18)$$

Proof: The theorem is proven by induction using Lemmas 1, 2 and 3.

Depending on the requirements of each particular application any value of $L_{\min}$ can be chosen. Thus, as (18) holds for any t, n should be designed to be equal or higher than its minimum value constrained by (10), which is the only condition to be satisfied in order to achieve a fully-decodable code.

B. Properties of the Proposed Encoding

The proposed method for encoding is aimed at creating constant weight words from a binary input string of a predetermined fixed length $L_{\min}$. Thus, the main property to be guaranteed is that any string $C = \{0,1\}^{L_{\min}}$ has its corresponding image $\Phi(C) = \delta$ on the set $W_{n,t}$. This property is always satisfied because (10) if met. It is easy to prove that $\binom{n}{t} > 2^{L_{\min}}$, so that $\varphi = \Phi^{-1}(\delta)$ is a partial function, i.e., the codomain of $\Phi(C)$ is a subset of $W_{n,t}$. However, the aim of the combinational approach, or the Sendrier's proposal, is to guarantee that any element of $W_{n,t}$ can be indexed with an integer codified with $L = \left\lceil \log_2 \binom{n}{t} \right\rceil$ bits. Thus, as $2^L \geq \binom{n}{t}$, the codomain of $\varphi = \Phi^{-1}(\delta)$ is now the subset of binary words that can be represented with L bits. Note that, in mathematical terminology, the objective of our proposal is to make the total function Φ injective, in contrast to the other approaches that are focused on φ.

Algorithm 5: Binary String to Constant Weight Word (Bin2CW) Using Loop for (Non-Optimized Version).

Input: length n, weight t, L_{min}, string C of length L_{min}
Result: a t-tuple $\delta = \{\delta_1, .., \delta_k, .., \delta_t\}$

```
index=0, delta=0, ReadBits=0, bitstop=0
for (j=0; j<L_min; j++) do
    d=dmin(t,(L_min-ReadBits))
    u_d=log_2(d)
    bitread=read(C,1)
    ReadBits++
    if bitread=1 then
        if bitstop=true then
            u--
            delta+=(1<<u)
            if u=0 then
                δ[index]=delta
                delta=0, index++
                t--, bitstop=false
        else
            delta+=d
            δ[index]=d
            u=u_d
    else
        if (bitstop=false) then
            bitstop=true
            u=u_d
        else
            u--
        if (u=0) then
            δ[index]= delta, delta=0,
            index++, t--, bitstop=false
    n--
return (δ_1,δ_2,.., δ_t)
```

C. Constant-Time Proposed Algorithm

In order to propose an implementation that can be executed in constant time, it is necessary to remove the *while-do* loop included in Algorithm 3. Since the number of bits read from C is always constant, then the number of iterations is fixed to $L_{\min}$ allowing the use of *for-do* loop. Algorithm 5 shows the first approach to achieve this goal. It should be noted that this algorithm is not optimized, and it is only included for a better understanding of the final constant time version. Since $d_{\min}$ is by definition a power of two, the decoding is very simple and is included as part of the pseudocode.

However, there are still some issues to address before this algorithm becomes constant time. The first one is the division $\lceil \frac{L'}{t'} \rceil$ included in the assessment of $d_{\min}$, while the second one is due to the use of branches created by the *if..then* statements. The former can be fixed by either, using a pre-computed lookup table

Non-constant time code

```
k=(1<<3)
if ((a=0 and b=1) or c=1) then
  d=e·f
  if (teta=0) then
    k=(1<<d)
  end
else
  d=g+h
  f=h+3·j
end
```

Constant time equivalent code

```
flag_a=(IsZero(a) and IsNotZero(b))or IsNotZero(c)
d=flag_a·(e·f) + (1-flag_a)(g+h)
flag_b= flag_a and IsZero(teta)
k=flag_b·(1<<d) + (1-flag_b)·(1<<3)
f=flag_a·f+(1-flag_a)·(h+3·j)
```

Fig. 3. Equivalence used to substitute the *if..then* statement by operations executed in constant time.

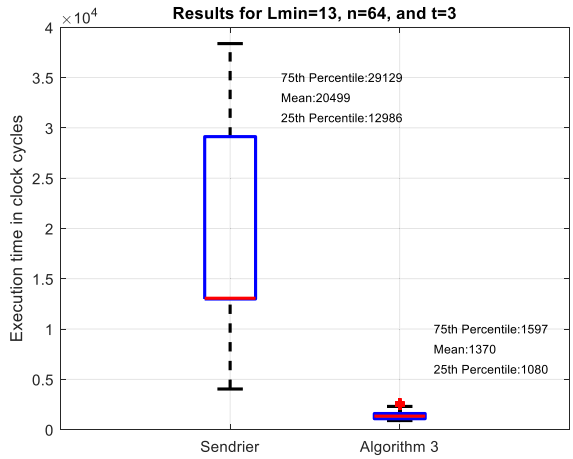


Fig. 4. Inter-quartile boxplots for the execution time of Sendrier and Algorithm 3.

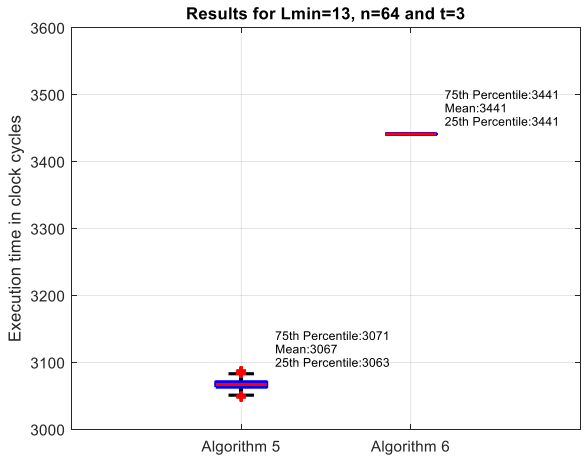


Fig. 5. Inter-quartile boxplots for the execution time of Algorithm 5 and 6.

or by implementing the long-division algorithm. The lookup table is only feasible for moderate values of $L_{\min}$ and t, where the available memory size allows the storage of the pre-computed divisions. The long-division algorithm is slower, but does not have memory space limitations. On the other hand, the *if..then* statements can be substituted and executed in constant time if the equivalence shown in the example code of Fig. 3 is used. Function *IsZero(a)* returns true or false depending on whether the integer a to be evaluated is zero or another value. The implementation of such a function in constant time is based on the rules and examples shown in [15]. The drawback of using this equivalence is that all the arithmetic operations are performed by the CPU, regardless of the result of the Boolean operation carried out between a, b and c. Algorithm 6 presents the final constant time version of our proposed algorithm.

IV. Experimental Results

In this section, the experimental results related to the proposed algorithm are presented. Performing a reliable measurement of timing on a CPU may be difficult, since the processor is continually switching from one process to another. Timing depends, among others, on factors such as the scheduling of processor resources, timer interrupts or the use of cache memory by one or several programs that can be executed concurrently [16], [17].

A. Results Using a Basic Microprocessor

In order to obtain timings not affected by all these issues, we demonstrate the correctness of our findings by executing the proposed algorithms in a basic microprocessor. We use an Avnet 7A50T development board, which includes an Artix-7 xc7a50ftg256-1 FPGA and 256 MB DDR3 SDRAM. Using Vivado 2019.1, a software suite developed by AMD for synthesis and analysis of HDL, a simple version of Microblaze soft microprocessor is implemented with 128 KB of data and instructions RAM memory. No floating-point unit is included. Only one single program is executed by the microprocessor and no cache memory is used, so accurate results can be collected. Additionally, an AXI-Timer peripheral is included. Its internal register TCR0 counts clock cycles, and it is used to obtain the execution time results. Due to the limitation of memory available, and since the aim of this test is just to serve as a proof of concept that validates our proposal, the parameters chosen are $L_{\min} = 13$, $t = 3$ and $n = 64$ (this value of n is based on (10)). The boxplots of Figs. 4 and 5 show the statistics of the execution time obtained for Sendrier's algorithm, Algorithm 3, Algorithm 5 and Algorithm 6, respectively. Such plots were obtained testing the $2^{13} = 8192$ possible combinations related to the input string. The extent of boxes corresponds to the inter-quartile interval; the central mark represents the median, whereas the bottom and top edges indicate the 25th and 75th percentiles, respectively; and the whiskers are related to those values that are 1.5 times or less than the inter-quartile interval above and below the two quartiles.

In a pure constant-time implementation the length of the inter-quartile interval should be null. In Fig. 4 the statistic for the Sendrier's algorithm is shown. Clearly, it is non-constant time and it provides the highest inter-quartile width. As no

Algorithm 6: Final Proposed Version of the Constant Time Binary String to Constant Weight Word (Bin2CW).

Input: length n, weight t, L_{min}, binary string C of length L_{min}
Result: a t-tuple $\delta = \{\delta_1, .., \delta_k, .., \delta_t\}$

```
index=0, delta=0, ReadBits=0, bitstop=0
for (j=0; j<L_min; j++) do
    d=dmin(t,(L_min-ReadBits))
    u_d=log_2(d)
    bitread=read(C,1)
    ReadBits++
    flag_zb=IsZero(bitstop)
    u=flab_zb·u_d+(1-flab_zb)(u-1)
    delta+= flag_zb·bitread·d+
            (1-flag_zb)·(1<<u)·bitread
    flag_u=IsZero(u)
    flag_rb=IsZero(bitread)
    δ[index]=delta·[flag_u·((1-flag_zb)·
        bitread+flag_rb)+bitread·flag_zb]
    delta-=flag_u·delta·[(1-flag_zb)·
                         bitread+flag_rb]
    index+=flag_u·((1-flag_zb)·bitread+flag_rb)
    t-=flag_u·((1-flag_zb)·bitread+flag_rb)
    flag_f=IsZero(flag_u)
    bitstop=((1-flag_rb)·flag_f·(1-flag_zb))
                         or (flag_rb·flag_f)
    n--
return (δ_1,δ_2,.., δ_t)
```

floating-point unit is included as part of the microprocessor, a fixed-point version of such algorithm was implemented. Even so, it is the slowest and takes about 20499 clock cycles in average. Algorithm 3, that makes the assessment of parameter d using (5), is also non-constant time but it is the fastest one. This is due to the fact that the calculation of (5) is very simple, being its result always a power of two. As mentioned earlier, Algorithm 5 is an improved version based on the use of a *for-do* loop that ensures a fixed number of iterations. As Fig. 5 shows, the length of the inter-quartile box is reduced to 8 cycles. This low value is produced as a consequence of the branch logic included in the algorithm through the use of *if-then* statements. The right-hand box of Fig. 5 corresponds to the results for our proposed constant time algorithm. Note as the execution time is always constant for any of the 8192 possible combinations of the input string. The boxplot becomes a straight line, being the values for both quartiles 3441 clock cycles. Additionally, in average it is only 12% slower than Algorithm 5, so the introduction of the equivalences presented in Fig. 3 to avoid the use of *if-then* statements is justified.

Finally, we checked as the number of bits read from the any input string C is always constant and equal to $L_{\text{min}} = 13$ bits. To achieve this aim, we used Algorithm 2 to reverse the process. The output of function Bin2CW, represented by the t-tuple δ, is used as input of the function CW2Bin, verifying that the same input string C with $L = L_{\text{min}}$ is obtained.

B. Results Using a Modern CPU

In order to validate our proposal in a modern CPU, all algorithms are additionally run on an Intel Core i7-8700 CPU, clocked at 3.2 GHz and 16 GB of RAM memory. The implementation was performed writing the algorithms in ANSI C. All benchmarks were run and compiled on a Windows 11 with MinGW-W64-builds-4.3.5, and compilation options *–march = native –O3*. Timing results were obtained adapting the method proposed by Intel in [16], but for C codes executed in a Windows environment. Basically, the method in based on reading the timestamp counter through the RDTSC and RDTSCP assembly instructions. Additionally, the serializing CPUID instruction is used as well, in order to avoid the out of order execution that might lead to distorting measurements.

However, we observe as the load of the CPU, the lasts of measures, and particularly data and instructions cache memory, have a huge influence on the timing measurements. In order to avoid misleading statistics influenced by such factors, we modify the method proposed in [17] as follows:

1) For each input string, the proposed constant time algorithm is executed 20 times.
2) A K-best scheme is adopted, and the average timing is calculated by considering only the 3 lowest measures. In this way, most outliers and measurements partially affected by the previous factors are discarded.
3) The statistics are obtained collecting data for a total of N random input strings.
4) Mean (μ) and standard deviation (σ) of the collected N strings are calculated, and those measurements not included in the range $\mu \pm 3\sigma$ are eliminated. Thus, we are considering that such removed values are outliers non-detected by the K-best scheme. This way, the statistics are obtained using the 99,9% of the collected measures, leading to more consistent and reliable results.

We obtain measurements for Classic McEliece, using different parameters for n and t that provide several levels of post-quantum security. Substituting L_{min} and t in (10) the corresponding value for n is assessed.

Fig. 6 shows the execution time when collecting 10^5 random measurements. These pictures are obtained for the particular case in which the value of $n = 8192$ and $t = 128$, that provide maximum post-quantum security (Category 5). The upper picture shows as the method is quite efficient, appearing just a few outliers that are removed once the step 4 is applied (bottom picture).

Once the measurements are collected, and to verify that our proposed algorithm is constant time, a Welch t-student test was performed [18]. Such test allows to check if the distribution of two sets of data are statistically different. First, we collect 10^5 measurements using random strings of length L_{min}, and one additional set using an all one input string. Note that such string generates the t-tuple $\delta = \{\delta_{\max}, 0, 0, \ldots, 0\}$, with only one non-zero element, potentially creating the highest difference between timing measurements. Afterwards, the Welch t-student test is applied on the two sets of data. If the t-value obtained is less than $|t| \leq 4.5$ then the test fails to reject the null hypothesis (the two timing distributions are equal). Figs. 7 and 8 shows the t-statistic (absolute value) as the number of measurements increases, for Algorithm 3 (non-constant time) and Algorithm 6

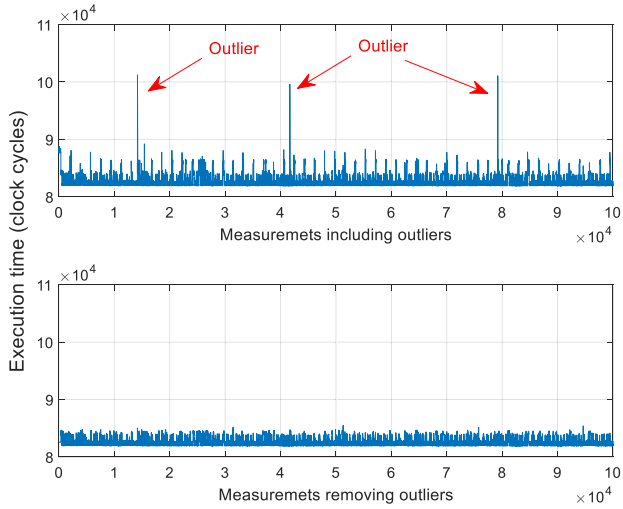


Fig. 6. Execution time (in clock cycles) for the constant time algorithm (Algorithm 6). Upper figure: including outliers; Bottom figure: removing outliers after applying step 4 of the proposed method.

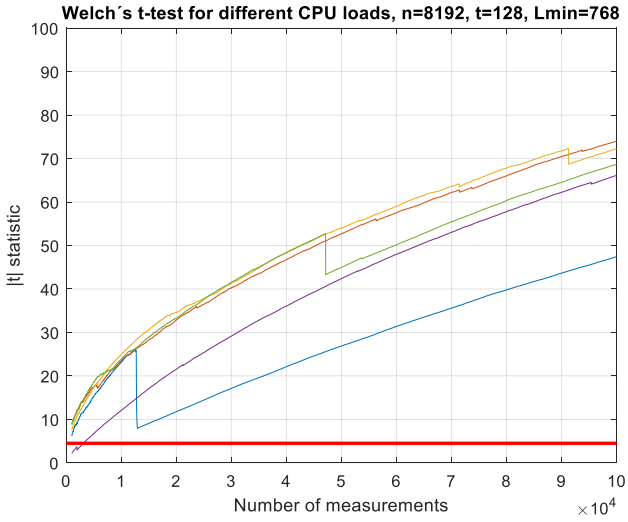


Fig. 7. Welch's t-test for different CPU loads using the non-constant time algorithm (Algorithm 3). The value of |t| is always higher than 4.5. Timing leakage is detected.

(constant time), respectively. Each figure presents 5 simulations that correspond to different CPU load conditions. Fig. 7 shows as in all cases the lines surpass the 4.5 threshold, so there is a strong statistical evidence that the distributions are different, so that it is clear that Algorithm 3 is non-constant time (timing leakage is detected). Yet, in Fig. 8, which shows the results for our proposed constant time algorithm, the t-value is always less than the threshold 4.5. Thus, once can conclude that the algorithm is executed in constant time since both distributions are indistinguishable, corroborating the results related to the proof of concept previously presented in Figs. 4 and 5.

C. Comparison With Other Proposals

Table III shows the timing results for Classic McEliece when using parameters that provide different security levels. Again,

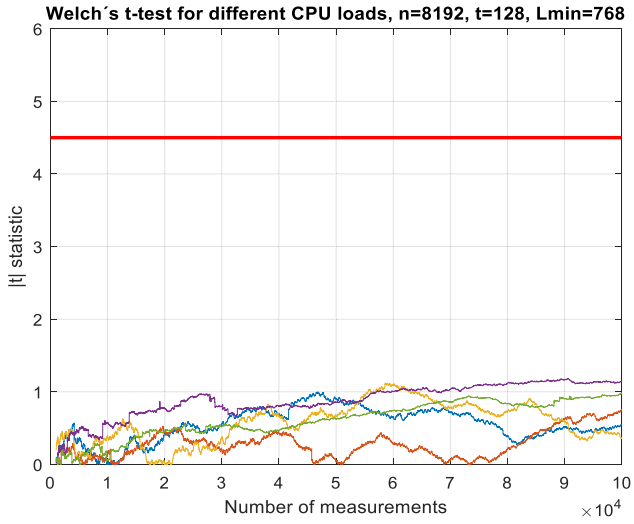


Fig. 8. Welch's t-test for different CPU loads using our proposal for constant time (Algorithm 6). The value of |t| is always less than 4.5. No timing leakage is detected.

Algorithm 3 (non-constant time) is the fastest version. It is observed as Algorithm 6 (constant time version) is between 6% and 15% slower than Algorithm 5, corroborating again that *if-then* statements can be removed with a low penalty on the execution time. The efficiency of our proposal is constant for any input string: about 81% for category 5; around 79% for category 3; and 80% for category 1.

Table IV shows a comparison between the execution time of our algorithm and the methods proposed in [5], [9] and [8] for different post-quantum algorithms. Reported benchmarks for the first method were obtained on an Intel i7-7700 clocked at 3.6 GHz. Clearly, the combinational approach proposed by Cover is the slowest one for all tested values of n and t. Algorithms proposed in [8] are public and available in [19]. We run these public algorithms using the same values that were proposed in [5] for n and t, but using the same CPU and compiler options employed to run our proposal. The value of $L_{\min}$ was chosen to comply (3). However, when executing our constant time algorithm, the value of $L_{\min}$ was selected to ensure (10) along with providing maximum efficiency. Note as, when using another operating system or compiler, results may differ substantially.

The method proposed in [5] generates directly a random constant weight binary string, that is not linked with any input sequence. Thus, a fair comparison is difficult to carry out, since no constraints are included in the input string, along with another CPU with different performances was used. Then, extracting quantitative results may lead to misleading conclusions. However, as it is shown in Table IV, it seems that results provided by our constant time algorithm depend on the ratio (n/t), and they improve as such ratio decreases. In fact, when (n/t) is lower than 0.2 our algorithm is the fastest one. This conclusion can be demonstrated as follows. Note as, the execution time of our proposal depends on the number of iterations fixed by the *for-loop* (i.e., $L_{\min}$). If $t \cdot \lceil \frac{L_{\min}}{t} \rceil$ is approximated by $L_{\min}$, then

TABLE III
CLOCK CYCLES REQUIRED TO COMPUTE THE NON-CONSTANT (ALGORITHM 3 AND 5) AND CONSTANT TIME ALGORITHMS (ALGORITHM 6)

Bin2CW	**Category 5** n=8192, t=128, L_{min}=768	**Category 3** n=4608, t=96, L_{min}=528	**Category 1** n=3488, t=64, L_{min}=365
Non-constant time version (algorithm 3)	26,184 cycles (8.183 µs)	18,349 cycles (5.734 µs)	11,285 cycles (3.527 µs)
Non-constant time version (algorithm 5)	77,307 cycles (24.158 µs)	50,796 cycles (15.874 µs)	32,093 cycles (10.029 µs)
Constant time version (algorithm 6)	82,365 cycles (25.739 µs)	56,592 cycles (17.685 µs)	37,500 cycles (11.719 µs)
Efficiency=$\frac{L_{min}}{log_2\binom{n}{t}}$	0.8118	0.7904	0.7999

TABLE IV
COMPARISON AGAINST OTHER PROPOSALS USING DIFFERENT POST-QUANTUM ALGORITHMS. RESULTS GIVEN IN CLOCK CYCLES

Alg.	**Cat.**	**n (bits) t (bits)**	**Ratio (n/t)**	**Rep. AND method Drucker et al. [5]**	**Cover's method [9] (Comb. approach)**	**Method proposed by Berenghi et al [8]**	**Our method (constant time)**
BIKE1	1	n=20,326 t=134	0.006	137,130 (cycles)	79,157,610 (cycles)	656,510 (cycles) (L_{min}=670, d=16)	**114,418 (cycles) (L_{min}=963)**
BIKE1	5	n=32,749 t=137	0.004	189,791 (cycles)	144,586,930 (cycles)	728,590 (cycles) (L_{min}=822, d=32)	**129,105 (cycles) (L_{min}=1078)**
BIKE1	5	n=65,498 t=261	0.004	426,333 (cycles)	736,269,120 (cycles)	1,277,620 (cycles) (L_{min}=1566, d=32)	**264,792 (cycles) (L_{min}=2078)**
Lizard	1	n=536 t=140	0.261	**2,377 (cycles)**	841,290 (cycles)	650,560 (cycles) (L_{min}=276, d=1)	27,102 (cycles) (L_{min}=268)
Lizard	5	n=1,024 t=200	0.195	**5,072 (cycles)**	2,269,620 (cycles)	878,880 (cycles) (L_{min}=400, d=2)	49,605 (cycles) (L_{min}=456)
NTRUEn	1	n=443 t=143	0.322	**2,006 (cycles)**	653,090 (cycles)	616,590 (cycles) (L_{min}=286, d=1)	22,690 (cycles) (L_{min}=222)
NTRUEn	5	n=743 t=247	0.322	**3,414 (cycles)**	1,537,350 (cycles)	1,065,380 (cycles) (L_{min}=494, d=1)	40,395 (cycles) (L_{min}=372)
NTRUPr	5	n=761 t=261	0.328	**3,086 (cycles)**	1,622,030 (cycles)	1,127,420 (cycles) (L_{min}=499, d=1)	41,793 (cycles) (L_{min}=381)

(10) becomes

$$n_{\min} = \left(L_{\min} + 2t - t\left\lceil \frac{L_{\min}}{t} \right\rceil\right) 2^{\left\lceil \frac{L_{\min}}{t} \right\rceil - 1} \approx t \cdot 2^{\left\lceil \frac{L_{\min}}{t} \right\rceil} \tag{19}$$

and then

$$L_{\min} \approx t \cdot \log_2\left(\frac{n}{t}\right) \tag{20}$$

Therefore, the expected execution time is $\mathcal{O}(t \cdot \log_2(\frac{n}{t}))$. Hence, when comparing two measurements that shares the same ratio (*n/t*), the relation between both timings is given by the quotient between the values of *t*. This conclusion is corroborated by the results obtained for BIKE1 (Category 5). Note as the execution time of the second one is twice, that basically is the relation between the values of *t* used in each case (261/137 = 1.9).

D. Discussion About Pros and Cons of the Proposal

The generation of constant weight words is currently part of the implementation of the post-quantum algorithms BIKE and HQC. As it was recently revealed in [24], a timing attack can be performed because timing depends on the seed used to initialize the pseudorandom generator. The same author, proposed a variant of the Fisher-Yates shuffle algorithm to fix this problem and to achieve secure implementations against timing and cache attacks. On the other hand, after the recommendation made to the authors of HQC, a modification was included in the fourth-round version submitted to the NIST PQC [25], [26]. This variant is based on the algorithm presented in [24], fixing the non-constant time behavior of the earlier fixed-weight versions.

In terms of security, the main advantage of our proposal is that is constant time for any input string of length $L_{\min}$, so that timing attacks cannot be performed successfully. However, its main shortcoming is the dependency between $L_{\min}$, n and t as stated in (10), i.e., not all combinations of these values are accepted. For instance, if Classic McEliece is used to encrypt a facial biometric feature coded with $L_{\min} = 820$ bits, then if $t = 128$ it leads to a value $n \geq 11520$. Although, such code length is completely valid and can be employed, in this particular case the system would be somewhat oversized, since the maximum post-quantum security can be achieved using only $n = 8192$.

E. Future Research Lines

Fault injections attacks are intended to alter the correct functioning of a computing device by means of variations in the power supply or clock frequency, overheating or exposing devices to intense light, etc. These attacks, and their corresponding countermeasures like error detection, are not studied in this paper. Additionally, the proposed algorithm only works for binary

alphabets. The extension to non-binary alphabets, or considering injections attacks, could be a future research line to be explored. The round 2 candidates announced in 2024 for Post-Quantum digital signature, include two algorithms based on code-based signatures [28], [29]. These schemes use algorithms to create fixed-weight challenges. An additional potential research line, focused on the application of our proposal to generate these challenges, could be an interesting option to be considered.

V. Conclusion

We introduced a novel proposal for obtaining constant weight words by encoding binary strings. Our main contribution is that the input binary sequence has a fixed-length, is fully-decodable (when the only constraint is met), it offers a high efficiency and it runs in constant-time, preventing timing leakage. Our algorithm is valuable for various post-quantum cryptosystems or other applications in which a constant weight word is needed.

The Classic McEliece cryptosystem was chosen to validate our findings, using several values of n and t to achieve different levels of security. A method to obtain confidence measures of the execution time was proposed. Finally, by applying the Welch t-student test to the collected data, we demonstrated that the proposed implementation is constant time.

References

[1] National Institute of Standards and Technology (NIST), "Post quantum project," Nov. 2017. [Online]. Available: https://csrc.nist.gov/Projects/post-quantum-cryptography/post-quantum-cryptography-standardization/Call-for-Proposals

[2] T. Bernstein et al., "Classic McEliece," 2017. [Online]. Available: https://classic.mceliece.org/peole.html

[3] R. J. McEliece, "A public cryptosystem based on algebraic coding theory," The Deep Space Network Progress Report, DSN PR 42-44, 1978. [Online]. Available: https://tda.jpl.nasa.gov/progress_report/42-44/44N.PDF

[4] H. Niederreiter, "Knapsack-type cryptosystems and algebraic coding theory," *Problems Control Inf. Theory*, vol. 15, pp. 159–166, 1986.

[5] N. Drucker and S. Gueron, "Generating a random string with a fixed weight," in *Proc. Int. Symp. Cyber Secur. Cryptogr. Mach. Learn.*, 2019, pp. 141–155, doi: 10.1007/978-3-030-20951-3_13.

[6] K. Kobara and H. Imai, "Semantically secure McEliece public-key cryptosystems-conversions for McEliece PKC," in *Proc. Public Key Cryptography*, Berlin, Germany, vol. 1992, 2001, pp. 19–35, doi: 10.1007/3-540-44586-2_2.

[7] M. Baldi et al., "LEDAcrypt specification v3.0," 2020. [Online]. Available: https://www.ledacrypt.org/LEDAcrypt/

[8] A. Barenghi and G. Pelosi, "Constant weight strings in constant time: A building block for code-based post-quantum cryptosystems," in *Proc. 17th ACM Int. Conf. Comput. Frontier*, 2020, pp. 132–141, doi: 10.1145/3387902.3382630.

[9] T. Cover, "Enumerative source encoding," *IEEE Trans. Inf. Theory*, vol. 19, no. 1, pp. 73–77, Jan. 1973, doi: 10.1109/TIT.1973.1054929.

[10] N. Sendrier, "Encoding information into constant weight words," in *Proc. IEEE Int. Symp. Inf. Theory*, 2005, pp. 435–438, doi: 10.1109/ISIT.2005.1523371.

[11] S. Golomb, "Run-length enconding," *IEEE Trans. Inf. Theory*, vol. 12, no. 3, pp. 399–401, Jul. 1966, doi: 10.1109/TIT.1996.1053907.

[12] J. Hu, R. C. C. Cheung, and T. Güneysu, "Compact constant weight coding engines for the code-based cryptography," *IEEE Trans. Circuits Syst.*, vol. 64, no. 9, pp. 1092–1096, Sep. 2017, doi: 10.1109/TCSII.2017.2675449.

[13] S. Heyse, "Low-Reiter: Niederreiter encryption scheme for embedded microcontrollers," in *Proc. Post-Quantum Cryptography*, 2010, pp. 165–181, doi: 10.1007/978-3-642-12929-2_13.

[14] S. Heyse and T. Güneysu, "Towards one cycle per bit asymmetric encryption: Code-based cryptography on reconfigurable cardware," in *Proc. Cryptographic Hardware Embedded Syst. 2012*, 2012, pp. 340–355, doi: 10.1007/978-3-642-33027-8_20.

[15] J. Gilcher, "Constant-time implementation of NTS-KEM," M.S. thesis, Computer Science, Swiss Federal Institute of Technology Zurich, Sweden, Mar. 2020.

[16] G. Paoloni, "How to benchmark code execution times on Intel IA-32 and IA-64 instruction set architecture," White paper Intel corporation, Sep. 2010.

[17] R. Bryant and D. O'Hallaron, *Computer Systems: A Programmer's Perspective*, 2nd ed. Reading, MA, USA: Addison-Wesley, Feb. 2010.

[18] O. Reparaz, J. Balasch, and I. Verbauwhede, "Dude, is my code constant time?," in *Proc. Conf. Des., Automat. Test Europe*, Mar. 2017, pp. 1697–1702.

[19] A. Barenghi and G. Pelosi, "Code for constant weight strings in constant time," (n.d.). 2020. [Online]. Available: https://zenodo.org/records/3747546

[20] J. Hu, Y. Liu, R. C. C. Cheung, S. Bhasin, S. Ling, and H. Wang, "Compact code-based signature for reconfigurable devices with side channel resilience," *IEEE Trans. Circuits Syst.*, vol. 67, no. 7, pp. 2305–2316, Jul. 2020, doi: 10.1109/TCSI.2020.2984026.

[21] Y. Song et al., "A code-based signature scheme from the Lyubashevsky framework," *Theor. Comput. Sci.*, vol. 835, Oct. 2020, pp. 15–30, doi: 10.1016/j.tcs.2020.05.011.

[22] S. Alahmadi, K. Kasem, and B. Magdy, "Enhancing arbiter PUF against modeling attacks using constant weight encoding," in *Proc. IEEE 67th Int. Midwest Symp. Circuits Syst.*, Aug. 2024, pp. 173–177.

[23] H. Wang, Y. Li, and L. P. Wang, "Post-quantum secure password-authenticated key exchange based on Ouroboros," *Secur. Commun. Netw.*, Jul. 2022, pp. 1–11, doi: 10.1155/2022/9257443.

[24] N. Sendrier, "Secure sampling of constant-weight words – Application to BIKE," Cryptology ePrint Archive. 2021. [Online]. Available: https://eprint.iacr.org/2021/1631

[25] S. Deshpande et al., "Fast and efficient hardware implementation of HQC," Cryptology ePrint Archive, Paper 2022/1183, 2022. [Online]. Available: https://eprint.iacr.org/2022/1183

[26] A. Budroni, I. A. Canales-Martínez, and L. Padolfo, "Methods for sampling random permutations in post-quantum cryptography," Cryptology ePrint Archive, Paper 2024/008, 2024. [Online]. Available: https://eprint.iacr.org/2024/008

[27] R. Arjona et al., "Post-quantum biometric authentication based on homomorphic encryption and classic McEliece," *Appl. Sci.*, vol. 13, no. 2, 2023, Art. no. 757, doi: 10.3390/app13020757.

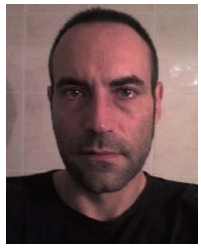
[28] M. Baldi et al., "CROSS codes and restricted objects signature scheme," 2024. [Online]. Available: https://cross-crypto.com/

[29] M. Baldi et al., "LESS linear equivalence signature scheme," 2024. [Online]. Available: https://www.less-project.com

[30] A. Borisenko et al., "Description and applications of binomial numeral systems," *Int. J. Innov. Comput., Inf. Control*, vol. 10, no. 1, pp. 57–66, 2014.

Mariano López-García received the M.S. and Ph. D. degrees in telecommunication engineering from the Universitat Politècnica de Catalunya (UPC), Barcelona, Spain, in 1996 and 1999, respectively. Since 2001, he has been an Associate Professor with the Department of Electronic Engineering (UPC). He currently teaches courses in electronics and advance digital design. He also taught during several years Power Electronics, Microelectronics and Analog Electronics at undergraduate level. He spent one year with Cambridge University, Faculty of Computer Sciences, as Visiting Scholar collaborating on the implementation of biometric algorithms on low-cost devices. His research interests include signal processing, biometrics, post-quantum cryptography, embedded systems, hardware-software co-design and FPGAs.

David G. Farouk-Marei received the bachelor's degree in computer science from the October University for Modern Sciences and Arts, Cairo, Egypt and the University of Greenwich, London, U.K. and the master's degree in artificial intelligence and computer security engineering from Universitat Rovira i Virgili, Tarragona, Spain, in 2021. He is currently working toward the Ph.D. degree with Universitat Politècnica de Catalunya, Barcelona, Spain. He is currently a Data Scientist with Universitat Politècnica de Catalunya,. His academic journey includes a he is currently with Universitat Politècnica de Catalunya as a Researcher, where his expertise spans various computer science disciplines, with a particular focus on researching biometrics combined with post-quantum cryptography. This showcases his comprehensive foundation in both academic and practical realms.



Enrique Cantó-Navarro received the M.S. in electronics enginering and the Ph.D degree from the Universitat Politècnica de Catalunya (UPC), Barcelona, Spain, in 1995 and 2001, repectively. Since 1996, he has been Associate Professor with the UPC, and Assistant Professor with the Universitat Rovira i Virgili (URV), Tarragona, Spain, since 2003. He has participated in several National and International research projects related to smart-cards, FPGAs, hardware accelerators and biometrics. He has published more than 60 research papers in journals and conferences. His research interests include hardware accelerators for biometrics algorithms, cryptography, and run-time reconfigurable embedded systems.