A Concurrent Signature Scheme from Coding Theory Assumptions

ASSIDI Hafsa

***Abstract*—Concurrent signature is considered as a type of fair exchange protocols. This signature allows two entities to exchange their signature in a fair manner. In that case, the two signatures are ambiguous until an extra piece of information, namely the keystone, is revealed. Consequently, the two signatures become binding to their real signers. Concurrent signatures find applications in different real life scenarios such as auction protocol, fair tendering of contracts, electronic transactions, etc. In this paper, we propose a concurrent signature scheme based on coding theory assumptions as a promising alternative to classical cryptography in the era of quantum computers. Our construction fulfils the security requirements of concurrent signature including correctness, unforgeability, ambiguity and fairness. In addition, our construction presents a practical results for signature size and public key length. For instance, we achieve a signature of size equal to $4.2KB$ and a public key of length $3200KB$ for $128 - bits$ security level, we find an average 96% reduction of the signature public key size. Our construction fulfills all the security requierement and achieves optimal results in terms of public key size.**

***Index Terms*—Concurrent Signature, Code-based Cryptography, Fairness, Correctness, Unforgeability, Provable Security, Ambiguity, Post-quantum.**

I. Introduction

The concept of concurrent signature was introduced for the first time by Chen *et al.* [1] in 2004. Concurrent signature allows entities to exchange their digital signatures efficiently and fairly. The two entities, namely an initial signer A and a matching signer B sign messages without the need of a third entity. The signers exchange their signature in a fair way, that is, either allowing all the entities to get each other's signatures simultaneously or letting none of them get any counterpart's signature. This point of time is exactly when one of the two signers computes a set of information called a keystone. Consequently, the two signatures are ambiguous in a third entity point of view until the keystone is revealed. Then, the two signatures become binding to their real signers.

Since its introduction, concurrent signature has been considered as a type of fair exchange protocols. The advantage of concurrent signature is that, it does not rely on any trusted or semi-trusted third entity for disputing resolution or assume computational balance between the entities compared to other fair exchange solutions.
Concurrent signature find applications in different real life scenarios such as auction protocol, fair tendering of contracts, electronic transactions and so on. In what follows, we illustrate one of the use cases of concurrent signature namely in electronic transactions. We consider the situation where a customer C aims to purchase an article from a vendor V. To achieve this, the customer and the vendor exchange their respective signatures. The initial signer A chooses a keystone, then signs his payment instruction ambiguously to pay the vendor V as the matching signer. When receiving the client's signature, the vendor V agrees this order by signing a receipt ambiguously that authorizes C to pick up the article from the shop. In order to get the article from the shop, the client C has to present both vendor's signature and the keystone, because vendor's ambiguous signature alone can be forged by C. When the keystone is released, both of the two ambiguous signatures become binding concurrently to V and C respectively. Therefore, V can present the client's signature with the keystone to get money from bank.

A secure concurrent signature must verify some specified security requirements including correctness, unforgeability, fairness and ambiguity. The correctness means that, if a signature is generated by a honest signer, then the verification process will succeed. The unforgeability property means that the entity that receives an ambiguous signature, for example the matching signer (without loss of generality), is sure that the signature is produced by the initial signer. For the fairness property, it means that the signatures of the two entities (the initial and the matching signer) are bounded concurrently *i.e* until a set of information is revealed, namely the keystone. Concerning the ambiguity, we say that a signature is ambiguous from a third entity point of view, when he cannot distinguish whether the signature is generated by the initial or the matching signer.

In the literature, Chen *et al.* [1] presented for the first time the concept of concurrent signatures. They also explain how concurrent signatures can provide a solution to the problem of fair exchange of signatures. Therefore, they present a security model and a concrete construction of such kind of signature. Later, many constructions and variants of concurrent signature have been proposed. For example in [2], the authors extend the concurrent signature by using a new notion called perfect concurrent signature, they instantiate their proposal by a bilinear pairing construction. In [3], the authors provide the first generic construction of identity-based perfect concurrent signature schemes from ring signature schemes. In [4], the authors propose a new concurrent signature scheme which is independent of the ring signature concept. Their concurrent signatures are anonymous. The ordinary signatures obtained from their concurrent signature protocol are unlinkable and do not reveal which concurrent signature transaction has occurred. In the multi-users setting, the proposal of [5] is the first construction of multi-users concurrent signatures using techniques of ring signatures and

Manuscript received July 26, 2022; revised February 6, 2023.
ASSIDI Hafsa is a PhD candidate of University Mohammed V, Rabat, Morocco, BP 1014, (e-mail: assidihafsa@gmail.com).

bilinear pairings. In 2017, BaoHong *et al.* [6] present an attribute concurrent signature which is an extension of concurrent signature in the attribute-based setting. The proposed scheme of [6] allows two entities to fairly exchange their signatures only if each of them has convinced the opposite entity that he or she possesses some attributes that satisfy a given signing policy.

Concerning post-quantum cryptography, in 2013, Wang *et al.* [7] presented a lattice-based multi-parties concurrent signature scheme where a new formal model of multi-parties concurrent signatures is proposed. Their construction is based on constant-size ring signatures and a lattice-based multi-parties concurrent signatures scheme. Later in 2016, Xiang *et al.* in [8] proposed an efficient multi-parties concurrent signature from lattice assumptions. The security of their scheme is based on the hardness of the small integer solution (SIS) problem. In 2017, Asaar *et al.* [9] proposed a code based concurrent signature. Their proposal is derived from the modified CFS signature [10] and it is proved secure under the hardness of code-based assumptions in the random oracle model.
In the literature, theire are some recent works related to encryption like [11] where the authors present an encryption system involving matrix associated with semigraphs. Also in terms of coding theory, the authors in [12] proposed a weight enumerator of some families of linear error-block codes and in [13] they prensent a tensor product and linear error block codes.

Given that the number theoretic based cryptography will not resist to the quantum computer as shown by Shor in his paper [14], the research for alternative solutions is of special interest. Code based cryptography is considered as an attractive and prominent alternative to classical cryptography in the era of quantum computers as well as lattice based cryptography, multivariate cryptography and isogeny based cryptography. In the literature, many cryptographic primitives were derived from coding theory assumptions such as group signature [15], ring and threshold ring signature In 2020, a strong designated verifier signature was proposed [16] using rank metric code-based cryptography.

The contribution of this paper consists in proposing a concurrent signature scheme that is based on error correcting code assumptions. Our construction is based on the ternary generalized $(U|U+V)$ codes to design a "hash-and-sign" signature [17] . Our construction fulfils the security requirements including the correctness, the fairness, the unforgeability and the ambiguity. The practical results show that our proposal is practical, for instance, we achieve a signature of size equal to $4.2KB$ and a public key of length $3200KB$ for $128-bits$ security level.

This paper is organized as follows: in Section II, we recall some definitions from error correcting codes. We recall also the algorithms that compose a concurrent signature and we give formal definitions of the security requirements in Section III. In Section IV, we present our proposed construction of code-based concurrent signature. Section V is devoted to the security analysis of our proposal. In Section VI, we analyse the performance of the proposed concurrent signature scheme in terms of public key size and signature size, then we make a comparison with some known concurrent signature schemes. We conclude in Section VII.

II. Preliminaries

In this section, we first provide the notations that will be used all along this paper. Secondly, we give some background related to code-based cryptography. We present also the wave signature scheme based on ternary generalized $(U|U+V)$ codes.

A. Error Correcting Codes

Let F_q denotes the finite field of cardinality q. A linear $[n,k]$-code, denoted $\mathcal{C}$, is a linear subspace of dimension k of the vector space F_q^n , where k and $n \in N$ with $n \geq k \geq 1$. The elements of F_q^n and of an $[n,k]$-code are named words and codewords, respectively. We use coding theory notations, where G and H respectively denote generator and parity check matrices of a code. We consider a vector $x \in F_q^n$ and a matrix H in the space of $k \times n$ matrices over F_q, $H \in \mathcal{M}_{k\times n}(F_q)$. The product $H \cdot x^T$ is called the syndrome of x where x^T is the transpose of a vector x and $w_H(x)$ refers to the Hamming weight of x.

A generalized $(U|U+V)$ code $\mathcal{C}$ of length n over F_q is built from two codes U and V of length $\frac{n}{2}$ and 4 vectors a, b, c and d in $F_q^{\frac{n}{2}}$ as the following "mixture" of U and V:

$$\mathcal{C} = \{(a \odot u + b \odot v, c \odot u + d \odot v) : u \in U, v \in V\} \quad (1)$$

where $x \odot y$ stands for the component-wise product.

We recall hereafter two difficult problems based on coding theory assumptions that were proven to be NP hard. These problems are the syndrome decoding problem and the problem of distinguishing generalised $(U|U+V)$ code from random codes.

Problem 1 (Syndrome decoding (SD) [18]): The $SD(n,k,\omega)$ problem is formulated as follows: let n, k and ω be non negative integers, given a uniformly random matrix $H \in \mathcal{M}_{k\times n}(F_q)$ and a uniformly random syndrome $y \in F_q^k$, find a vector $s \in F_q^n$ such that $w_H(s) \leq \omega$ and $H \cdot s^\top = y^\top$.

Problem 2 (Distinguishing generalised $(U|U+V)$ codes): Distinguishing a permuted generalised $(U|U+V)$ code from a random code of the same length and dimension and this even when U and V are themselves random codes.

B. The generalised $(U|U+V)$ Signature Scheme

In what follows, we describe the generalised $(U|U+V)$ signature scheme as it is presented in [17]. Let H be a parity check matrix of a q-ary linear code $\mathcal{C}$ of length n and dimension k. We consider a one way function $f_{w,H}$ defined as follows: $f_{w,H} : S_\omega \to F_q^{n-k}$ such that:

$$f_{w,H}(e) = eH^T \quad (2)$$

Where ω is a non negative integer and S_ω is the subset of F_q^n of words of weight equal to ω.

A Trapdoor One-way Preimage Sampleable on Average Code-based Functions is a pair of probabilistic polynomial-time algorithms $(Trapdoor, InvAlg)$ together with a triple

of functions $(n(\lambda), k(\lambda), w(\lambda))$ growing polynomially with the security parameter λ and giving the length and dimension of the codes and the weights we consider for the syndrome decoding problem, such that:

- $Trapdoor$: when given λ, outputs (H, T) where H is an $(n-k) \times n$ matrix over F_q and T the trapdoor corresponding to H.
- $InvAlg$: is a probabilistic algorithm which takes as input T and an element $s \in F_q^{n-k}$ and outputs an $e \in S_{w,n}$ such that $eH^T = s$.

In what follows, we describe concretely the algorithms $Trapdoor$ and $InvAlg$:
We consider λ as a security parameter and n, k, w are the system parameters. We split $k = k_U + k_V$, The tuple $T = sk = (\phi, H_U, H_V, S, P)$ is the secret key where ϕ is a UV-normalized mapping (The definition of UV-normalized mapping is in [17] Proposition 1), $H_U \in F_q^{(\frac{n}{2}-k_U)\times\frac{n}{2}}$, $H_V \in F_q^{(\frac{n}{2}-k_V)\times\frac{n}{2}}$, $S \in F_q^{(n-k)\times(n-k)}$ is a non singular matrix and $P \in F_q^{n\times n}$. All the elements of the secret key sk are chosen randomly and uniformly in their domain.
From (ϕ, H_U, H_V) we derive the parity check matrix $H_{sk} = \mathcal{H}(\phi, H_U, H_V)$ (For the definition of $\mathcal{H}$, see [17] Proposition 1). The public key is $H = SH_{sk}P$. We consider D_{ϕ,H_U,H_V} as an algorithm which inverts $f_{w.H_{sk}}$. To conclude we have:

- $(pk, sk) = Trapdoor(\lambda)$ where $sk = (\phi, H_U, H_V, S, P)$ and $pk = H$.
- $InvAlg(s, sk)$ returns eP where $e = D_{\phi,H_U,H_V}(s.(S^{-1})^T)$.

Given a one-way preimage sampleable code-based function $(Trapdoor, InvAlg)$ we easily define a code-based FDH signature scheme as follows. We generate the public/secret key as $(pk, sk) = (H, T) \leftarrow Trapdoor(\lambda)$. We also select a cryptographic hash function $h : \{0,1\}^* \to F_q^{n-k}$ and a salt r of size λ_0 . The signature and verification algorithms are defined as follows:

Algorithm 1 The generalised $(U|U+V)$ signature

Input: A message m.
Output: The signature σ of m.

- To sign a message m, the signer chooses randomly $r \in \{0,1\}^{\lambda_0}$ (where λ_0 is a positive integer as a security parameter),
- Then, he computes $s = h(m, r)$ and $e = InvAlg(s, T)$ where $h : \{0,1\}^* \to F_q^{n-k}$ is a one way hash function,
- The signature is $\sigma = (e', r)$ where $e' = eP$.

Algorithm 2 The generalised $(U|U+V)$ verification

Input: A message m and its signature $\sigma = (e', r)$.
Output: 1 or 0.

- The verifier computes $s = h(m, r)$ and $\omega_0 = w_H(e')$ (where w_H denotes the Hamming weight of a vector),
- If $e'H^T = s$ and $\omega_0 = \omega$ then return 1,
- else return 0 .

III. Concurrent Signature

A. Definition

A concurrent signature scheme [1] is a digital signature scheme comprised of the following algorithms:

- **Setup:** a probabilistic algorithm that takes as input a security parameter λ. It returns a set of users participants $\mathcal{U}$, the message space $\mathcal{M}$, the signature space $\mathcal{S}$, the keystone space $\mathcal{K}$, the keystone fix space $\mathcal{F}$, and a function $KGen : \mathcal{K} \to \mathcal{F}$. The algorithm also returns the public keys $p_k^{(i)}$ of all the participants, each participant retaining its private key $s_k^{(i)}$, $s_k^{(j)}$ and the public parameters $\mathcal{P} = \{\mathcal{U}, \mathcal{M}, \mathcal{K}, \mathcal{F}, \mathcal{S}\}$.
- **KGen:** this algorithm takes as input the public parameters $\mathcal{P}$, a random keystone $\kappa \in \mathcal{K}$ and outputs a keystone fix $x \in \mathcal{F}$ such that $x = KGen(\kappa)$.
- **ASign:** a probabilistic algorithm that takes as inputs $s_k^{(i)}$, $p_k^{(i)}$, $p_k^{(j)}$, a keystone fix $x \in \mathcal{F}$ and a message $m \in \mathcal{M}$ and outputs a signature $\sigma = (s, h_1, h_2)$ where $s \in \mathcal{S}$ and h_1, $h_2 \in \mathcal{F}$.
- **AVerify:** a deterministic algorithm that takes as inputs participants public keys $p_k^{(i)}$, $p_k^{(j)}$, the public parameters $\mathcal{P}$ and a signature $\sigma = (s, h_1, h_2)$ where $s \in \mathcal{S}$ and h_1, $h_2 \in \mathcal{F}$. The algorithm returns 1 if σ is a valid signature and 0 otherwise.
- **Verify:** a deterministic algorithm that takes as inputs participants public keys $p_k^{(i)}$, $p_k^{(j)}$, the public parameters $\mathcal{P}$, $\kappa \in \mathcal{K}$ and a signature $\sigma = (s, h_1, h_2)$ where $s \in \mathcal{S}$ and h_1, $h_2 \in \mathcal{F}$. It outputs 1 if $AVerify(\mathcal{P}, p_k^{(i)}, p_k^{(j)}, \sigma, m) = 1$ and $x = KGen(\kappa)$. Otherwise, it outputs 0.

B. Concurrent Protocol

In what follows, we define what is a concurrent protocol. The first entity A, who creates the keystone and sends the first ambiguous signature is called the initial signer. The second entity B, who responds to the initial signature by creating another ambiguous signature with the same keystone fix is called a matching signer. The two participants run the **Setup** algorithm to generate public parameters $\mathcal{P}$, A's public and secret keys $p_k^{(A)}$ and $s_k^{(A)}$ and B's public key $p_k^{(B)}$. A concurrent protocol consists of the following steps:

1) A picks a random keystone $\kappa \in \mathcal{K}$, and computes $x = KGen(\kappa)$. Then, A takes his/her own public key $p_k^{(A)}$ and B's public key $p_k^{(B)}$ and picks a message $m_A \in \mathcal{M}$ to sign. Finally, A computes his ambiguous signature to be:
$$\sigma_A = (s_A, h_A, x) = ASign(p_k^{(A)}, p_k^{(B)}, s_k^{(A)}, x, m_A) \quad (3)$$
Then A sends σ_A to B.
2) Once the ambiguous signature σ_A of participant A is received, B verifies the signature by checking that $AVerify(s_A, h_A, x, p_k^{(A)}, p_k^{(B)}, m_A) = 1$. If not, B aborts and then chooses a message $m_B \in \mathcal{M}$ to sign and computes his ambiguous signature:
$$\sigma_B = (s_B, h_B, x) = ASign(p_k^{(B)}, p_k^{(A)}, s_k^{(B)}, x, m_B) \quad (4)$$
Then, B sends σ_B back to A. Note that B uses the same value x in his signature as A did to produce σ_A.

3) The initial signer A verifies that $AVerify(s_B, h_B, x, p_k^{(B)}, p_k^{(A)}, m_B) = 1$, where x is the same keystone fix as A used in Step 1. If not, A aborts, otherwise A sends keystone κ to B. Note that $Verify$ algorithm outputs 1 on inputs (σ_A, κ) and (σ_B, κ).

C. Security model of concurrent signature

In what follows, we present the security requirements that must be verified by a concurrent signature namely, Correctness, Unforgeability, Ambiguity and fairness.

Correctness: A concurrent signature is correct if the following conditions hold: If $\sigma = ASign(p_k^{(i)}, p_k^{(j)}, s_k^{(i)}, x, m)$, then $AVerify(\sigma, p_k^{(i)}, p_k^{(j)}, m) = 1$. In addition, if $x = KGen(\kappa)$ where $\kappa \in \mathcal{K}$ then $Verify(\kappa, p_k^{(i)}, p_k^{(j)}, s_k^{(i)}, x, m) = 1$.

Unforgeability: We consider the following security game between the adversary $\mathcal{A}$ and a challenger *Ch*.

1) The challenger *Ch* runs the **Setup** algorithm on a security parameter λ, it obtains public parameters $\mathcal{P}$ participant's key pair (p_k, s_k) and sends $(\mathcal{P}, p_k)$ to adversary $\mathcal{A}$.
2) The adversary $\mathcal{A}$ makes the following queries to the challenger *Ch* as follows:
   - Private Key Extract: Adversary $\mathcal{A}$ can ask for the secret key of each user with public key p_k, and the challenger *Ch* returns the corresponding secret key, s_k.
   - $KGen$: Adversary $\mathcal{A}$ can ask *Ch* to choose a keystone $\kappa \in \mathcal{K}$ and returns its corresponding keystone fix $x = KGen(\kappa)$ to $\mathcal{A}$. Note that the corresponding keystone fix x is computed using $KGen$ algorithm.
   - $KReveal$: Adversary $\mathcal{A}$ can ask for the keystone $\kappa \in \mathcal{K}$ of each keystone fix $x \in \mathcal{F}$ which was returned by $KGen$. In response, *Ch* returns κ if x is a previous $KGen$ output; otherwise, it returns 0.
   - $ASign$: Adversary $\mathcal{A}$ can ask for an ambiguous signature on the tuple $(p_k^{(i)}, p_k^{(j)}, x, m)$, where $m \in \mathcal{M}$ is the message, $x \in \mathcal{F}$ is the keystone fix and $p_k^{(i)}$ and $p_k^{(j)}$ are users public keys. Then, *Ch* returns $\sigma = (s, h_1, h_2) = ASign(\mathcal{P}, s_k^{(i)}, p_k^{(i)}, p_k^{(j)}, x, m)$, where h_1, $h_2 \in \mathcal{F}$ and $s \in \mathcal{S}$.
3) The adversary $\mathcal{A}$ returns a forged signature σ^* of a message m^* where $\sigma^* = (s^*, h_1^*, h_2^*)$ using users public keys $p_k^{(i^*)}$ and $p_k^{(j^*)}$ such that $AVerify(\mathcal{P}, p_k^{(i^*)}, p_k^{(j^*)}, \sigma^*, m^*) = 1$, $\mathcal{A}$ wins the unforgeability game if one of the two conditions holds.
   a) No $ASign$ query with input either of the tuples $(p_k^{(i^*)}, p_k^{(j^*)}, x^*, m^*)$ or $(p_k^{(j^*)}, p_k^{(i^*)}, x^*, m^*)$ was made by $\mathcal{A}$ and no Private Key Extract query was made by $\mathcal{A}$ on either $p_k^{(i^*)}$ or $p_k^{(j^*)}$.
   b) No $ASign$ query with input $(p_k^{(i^*)}, p_k^{(j^*)}, x^*, m^*)$ was made by $\mathcal{A}$ for any $p_k^{(i^*)} \neq p_k^{(j^*)}$, no Private Key Extract query with input $p_k^{(i^*)}$ was made by $\mathcal{A}$, and either x^* was a previous output from a $KGen$ query or $\mathcal{A}$ produces a keystone κ such that $x^* = KGen(\kappa)$.

Definition 1: A concurrent signature scheme is existentially unforgeable under chosen message attack if the probability of success in the previous game is negligible.

Ambiguity: To define what is ambiguity, we consider the following game between an adversary $\mathcal{A}$ and a challenger *Ch*:

1) **Setup:** It is the same as the unforgeability game.
2) The adversary $\mathcal{A}$ makes a sequence of $KGen$, $KReveal$, $ASign$ and $Private\ Key\ Extract$ queries like the unforgeability game.
3) The adversary $\mathcal{A}$ request an ambiguous signature on inputs $(p_k^{(i)}, p_k^{(j)}, x, m)$, the challenger *Ch* chooses randomly $b \in \{i, j\}$ and returns either

$$\sigma_i = (s_1, h_1, f) = ASign(\mathcal{P}, s_k^{(i)}, p_k^{(i)}, p_k^{(j)}, x, m)$$

or

$$\sigma_j = (s_2, h_2, f) = ASign(\mathcal{P}, s_k^{(j)}, p_k^{(j)}, p_k^{(i)}, x, m)$$

4) The adversary $\mathcal{A}$ outputs $b' \in \{i, j\}$ and wins the ambiguity game if $b' = b$ and $\mathcal{A}$ has not made a $KReveal$ query on any values of f, h_1 or h_2.

Definition 2: A concurrent signature scheme is ambiguous if no polynomially bounded adversary has advantage that is non-negligibly greater than $\frac{1}{2}$ of winning in the above game.

Fairness: The concept of fairness is defined in the following game between the adversary $\mathcal{A}$ and the challenger *Ch*:

1) **Setup**: It is the same as the unforgeability game.
2) The adversary $\mathcal{A}$ makes a sequence of $KGen$, $KReveal$, $ASign$ and $Private\ Key\ Extract$ queries like the unforgeability game.
3) Finally $\mathcal{A}$ chooses the challenge public keys $p_k^{(i^*)}$ and $p_k^{(j^*)}$ outputs a keystone $\kappa^* \in \mathcal{K}$, and a signature $\sigma^* = (s^*, h_1^*, f^*), s^* \in \mathcal{S}, h_1^*, f^* \in \mathcal{F}$, and $m^* \in \mathcal{M}$ where $AVerify(\mathcal{P}, p_k^{(i^*)}, p_k^{(j^*)}, \sigma^*, m^*) = 1$. The adversary wins the game if either of the following cases hold:
   - Case 1: If f was a previous output from a $KGen$ query, no KReveal query on input f was made and also $Verify(\mathcal{P}, p_k^{(i^*)}, p_k^{(j^*)}, \sigma^*, \kappa^*, m^*) = 1$.
   - Case 2: If $\mathcal{A}$ also produces $\sigma' = (s', h_1', f), s' \in \mathcal{S}, h_1', f \in \mathcal{F}$, message $m' \in \mathcal{M}$, where $AVerify(\mathcal{P}, p_k^{(i^*)}, p_k^{(j^*)}, \sigma', m') = 1$ and (κ^*, σ^*) we have $Verify(\mathcal{P}, p_k^{(i^*)}, p_k^{(j^*)}, \sigma^*, \kappa^*, m^*) = 1$ but $Verify(\mathcal{P}, p_k^{(i^*)}, p_k^{(j^*)}, \sigma'^*, \kappa^*, m') = 0$.

Definition 3: A concurrent signature scheme is fair if a polynomially bounded adversary's probability of success in the above game is negligible.

IV. The Proposed Concurrent Signature

In this section, we present our proposal according to concurrent signature scheme from error correcting codes. We explain in details the components of our construction namely Algorithms 3, 4, 5, 6 and 7.

Setup: Given the security parameter λ, this algorithm outputs a set of public parameters $\mathcal{P}$, the secret key s_k and it's corresponding public key p_k as detailed in Algorithm 3.

Algorithm 3 $Setup(1^\lambda)$

Input: a security parameter λ.
Output: a set of public parameters $\mathcal{P}$, secret key s_k and public key p_k.

- Let $h : \{0,1\}^* \to F_q^{n-k}$ be a hash function and $g : F_q^{n-k} \to F_q^{n-k}$ is a random oracle where $w_H(g(.)) \leq t$ (where t is an integer).
- We consider $f_{w,H} : S_\omega \to F_q^{n-k}$ such that: $f_{w,H}(e) = eH^T$ where H is an $(n-k) \times n$ parity check matrix of a code over F_q
- The public key is $p_k = H$ and the secret key is $s_k = T$ where $(H, T) = Trapdoor(\lambda)$.
- The output is s_k, p_k and $\mathcal{P} = (h, g, f_{w,H}, n, k, t, w)$.

$KGen$: Given the public parameters $\mathcal{P}$ and a random keystone κ, it outputs a fix keystone $x = KGen(\kappa)$. Algorithm 4 describes more details on $KGen$ algorithm.

Algorithm 4 $KGen(\kappa)$

Input: a keystone κ.
Output: a keystone fix x.

- Let $\kappa \in F_q^n$ where $w_H(\kappa) \leq t$ (t is an integer).
- Let $x = g(H^{(i)}.\kappa^T)$ where $w_H(x) \leq t$.
- Return x.

To avoid any ambiguity through the upcoming sections we mention that each user has his own pair of keys (public and secret keys) that differs for all users:

- $s_k^{(i)} = (\phi_{(i)}, H_U^{(i)}, H_V^{(i)}, S^{(i)}, P^{(i)})$ refers to the secret key of user i.
- $s_k^{(j)} = (\phi_{(j)}, H_U^{(j)}, H_V^{(j)}, S^{(j)}, P^{(j)})$ refers to the secret key of user j.
- $p_k^{(i)} = H^{(i)} = S^{(i)} H_{sk}^{(i)} P^{(i)}$ is the public key of user i where $H_{sk}^{(i)} = \mathcal{H}(\phi_{(i)}, H_U^{(i)}, H_V^{(i)})$.
- $p_k^{(j)} = H^{(j)} = S^{(j)} H_{sk}^{(j)} P^{(j)}$ is the public key of user j where $H_{sk}^{(j)} = \mathcal{H}(\phi_{(j)}, H_U^{(j)}, H_V^{(j)})$.

$ASign$: The signature algorithm takes as input the public parameters $\mathcal{P}$, the signer's pair keys $(s_k^{(i)}, p_k^{(i)})$, the matching signer's public key $p_k^{(j)}$, a fix keystone x and the message m. Algorithm 5 explain the signature process.

Algorithm 5 $ASign(\mathcal{P}, p_k^{(i)}, s_k^{(i)}, p_k^{(j)}, x, m)$

Input: public parameter $\mathcal{P}$, users' keys $s_k^{(i)}, p_k^{(i)}, p_k^{(j)}$, a keystone κ and a message m.
Output: an ambiguous signature σ.

- Given a message $m \in F_2^*$, the signer chooses randomly r in $(F_2^n)^{\lambda_0}$.
- The signer computes $y = h(m, r, H^{(i)}, H^{(j)}) + g(H^{(j)}.x^T)$ and $e = InvAlg(y, sk_i)$.
- Output a signature $\sigma = (e', r, x)$ where $e' = eP$.

$AVerify$: This algorithm takes as input the public parameters $\mathcal{P}$, the signers public keys $p_k^{(i)}$, $p_k^{(j)}$ and a signature σ of a message m. The algorithm outputs 1 if the verification succeed and 0 otherwise.

Algorithm 6 $AVerify(\mathcal{P}, p_k^{(i)}, p_k^{(j)}, \sigma, m)$

Input: public parameter $\mathcal{P}$, users' public keys $p_k^{(i)}, p_k^{(j)}$ and signature σ of a message m.
Output: 1 or 0.

- **If** $e'(H^{(i)})^T = y$ and $w_H(e') = w$ where $y = h(m, r, H^{(i)}, H^{(j)}) + g(H^{(j)}.x^T)$
  Then return 1.
- **Else**
  return 0.
  End If

Given the same inputs as in $AVerify$ in addition to the keystone κ, this algorithm outputs 1 if the $AVerify$ returns 1 and the verification of the fix keystone succeed and 0 otherwise.

Algorithm 7 $Verify(\mathcal{P}, p_k^{(i)}, p_k^{(j)}, \sigma, \kappa, m)$

Input: public parameter $\mathcal{P}$, users' public keys $p_k^{(i)}, p_k^{(j)}$, a signature σ of a message m and the keystone κ.
Output: 1 or 0.

- **If** $e'(H^{(i)})^T = y$ and $w_H(e') = w$ where $y = h(m, r, H^{(i)}, H^{(j)}) + g(H^{(j)}.x^T)$ and $g(H^{(i)}.\kappa^T) = x$ and $w_H(x) \leq t$ and $w_H(\kappa) \leq t$.
  Then return 1
- **Else**
  return 0.
  End If

V. Security analysis

This section is devoted to analyse the security requirements using the Random Oracle (RO) model of the proposed concurrent signature scheme by proving, respectively, the correctness, the unforgeability, the ambiguity and the fairness.

Correctness

The correctness property means that if σ is a signature produced by $Asign$ algorithm. The process of its verification will succeed *i.e.* $Verify$ Algorithm 7 returns 1.

We consider $\sigma = (e', r, x)$, a signature obtained by $ASign$ Algorithm 5, we have to prove that $Verify(p_k^{(i)}, p_k^{(j)}, \sigma, \kappa, m) = 1$. For this reason, we have to prove in a first level that $AVerify(\mathcal{P}, p_k^{(i)}, p_k^{(j)}, \sigma, m) = 1$ i.e,

the conditions $e'.(H^{(i)})^T = y$ and $w_H(e') = w$ hold where $y = h(m, r, H^{(i)}, H^{(j)}) + g(H^{(j)}.x^T)$. Since $\sigma = (e', r, x)$ is a signature generated by $ASign$ Algorithm 5, we have by definition of $InvAlg$ algorithm

$$\begin{aligned} e &= D_\phi(y.(S^{-1})^T) \\ f_{w,H_{sk}}(e) &= y.(S^{-1})^T \\ e.(H_{sk})^T &= y.(S^{-1})^T \\ e.(H_{sk})^T.S^T &= y.(S^{-1})^T.S^T \\ e.(SH_{sk})^T &= y \\ e'.P^T.(SH_{sk})^T &= y \\ e'.(S.H_{sk}.P)^T &= y \\ e'.H^T = y & \end{aligned}$$

Furthermore, for the signer i we have $e'.(H^{(i)})^T = y$ and $w_H(e') = w$. In addition, since $\sigma = (e', r, x)$ is a valid signature, then $x = KGen(\kappa) = g(H^{(i)}.\kappa^T)$ where $w_H(x) \leq t$ and $w_H(\kappa) \leq t$.
We conclude that if $\sigma = (e', r, x)$ on a message m then $Verify(p_k^{(i)}, p_k^{(j)}, \sigma, \kappa, m) = 1$.

Unforgeability

The unforgeability property means that it is infeasible for an adversary $\mathcal{A}$ to produce a valid concurrent signature without the knowledge of signers secret keys.

We consider in one side an adversary $\mathcal{A}$ witch is able to break the unforgeability of the concurrent signature scheme with probability equal to ϵ_1, we consider on the other side a challenger *Ch* that is able to solve an instance of Problem 1 with probability ϵ_2 *i.e*. The adversary *Ch* returns a vector $s^* \in F_q^n$ such that $w_H(s^*) \leq t$ and $H^* s^{*T} = y^*$ where H^* is a random matrix in $F_q^{n-k \times n}$.
The challenger *Ch* runs the **Setup** Algorithm 3 on the security parameter λ and gets the public parameters $\mathcal{P} = (h, g, f_{w,H}, w, n, k, t)$, then the challenger *Ch* provides the adversary $\mathcal{A}$ with $\mathcal{P}$ and $H^{(i)} = H^*$, $\mathcal{A}$ makes adaptively a sequence of queries to the oracles that are constructed as follows:

- $g(.)$ queries: If $\mathcal{T}_g[.]$ is defined for query $(H(\kappa)^T)$, then *Ch* returns its value, else $\mathcal{T}_g[H(\kappa)^T]\$\leftarrow F_q^{n-k}$ where $w_H(g(H(\kappa)^T)) \leq t$ and returns $g(H(\kappa)^T)$ to $\mathcal{A}$ (where $\$\leftarrow$ means, it is chosen randomly).
- $h(.)$ queries: If $\mathcal{T}_h[.]$ is defined for query $(m, r, H^{(i)}, H^{(j)})$ the challenger *Ch* returns its value, else the challenger *Ch* chooses $\mathcal{T}_h[m, r, H^{(i)}, H^{(j)}]\$\leftarrow F_q^{n-k}$ and return it to $\mathcal{A}$.
- $KGen$ queries: The challenger *Ch* chooses randomly $\kappa \in F_q^n$ where $w_H(\kappa) \leq t$, then it computes $H.(\kappa)^T$. We distinguish two cases: If $\mathcal{T}_g[.]$ is already defined for query $H(\kappa)^T$, then *Ch* returns its value. Elsewhere, *Ch* chooses a random value $\mathcal{T}_g[H(\kappa)^T]\$\leftarrow F_q^{n-k}$ where $w_H(g(H.(\kappa)) \leq t$ and returns $x = g(H.(\kappa)^T)$ to $\mathcal{A}$, the tuple (κ, x) is joined to what we call K-list.
- $KReveal$ queries: The challenger *Ch* searches for the tuple (κ, x) in the K-list where x is produced by $KGen$ query. If (κ, x) is in the K-list returns κ else returns invalid.
- Private Key extract queries: The challenger *Ch* returns the secret key that corresponds to the public key.
- $ASign$ queries: For a query $(\mathcal{P}, H^{(i)}, H^{(j)}, x, m)$, the challenger *Ch* generates a signature using $ASign$ Algorithm 5.
- In the last step, the adversary $\mathcal{A}$ returns a signature $\sigma^* = (e^{'*}, r^*, x^*)$ of the message m^* where the public keys are H^* and $H^{(j)}$ with success probability equal to ϵ_1.

The adversary $\mathcal{A}$ wins the unforgeability game with probability equal to

$$Pr[Event1] \times Pr[Event2 Event1]$$

where:

- $Event1$: The adversary *Ch* does not abort as a result of $ASign$ oracle.
- $Event2$: The adversary $\mathcal{A}$ breaks the unforgeability of the scheme.

In order to compute the probability of $Event1$, we distinguish two cases:

- Case 1: If $(r^*, H^*, H^{(j)}, m)$ produced in one $ASign$ oracle has occurred in a previous query to $h(.)$ oracle, in that case we have $bad = true$ (Where event bad refers to the adversary *Ch* abort in $ASign$ Algorithm 5). Given that there exists $(q_h + q_s)$ (Whre q_h and q_s are the number of queries asked to h and $ASign$ oracles respectively) in $\mathcal{T}_h[.]$ and r is chosen uniformly at random in $F_2^{\lambda_0}$, we have the probability of this event for q_s query is at most $\frac{q_s(q_h+q_s)}{2^{\lambda_0}}$.
- Case 2: If the adversary *Ch* used the same randomness $r \in F_2^{\lambda_0}$ in one $ASign$ oracle, we have $bad = true$ and *Ch* makes at most q_s queries to $ASign$ oracle. As a consequence, the probability of this event is at most $\frac{q_s^2}{2^{\lambda_0}}$

$$Pr[Event1] \geq 1 - \frac{q_s(2q_s + q_h)}{2^{\lambda_0}}$$

Consequently, the adversary $\mathcal{A}$ returns the tuple $(e^{'*}, r^*, x^*)$ with probability at least equal to $\epsilon_1 - \frac{q_s(2q_s+q_h)}{2^{\lambda_0}}$.
The challenger employs $\mathcal{A}$, guesses the indices $1 \leq i_1 \leq q_g$ and $1 \leq i_2 \leq q_h$ and expect that i_1 be the index of the query $H^*(\kappa^*)^T$ to oracle g and i_2 be the index of the query (m^*, r^*, H^*, H^{j^*}) to oracle h. The adversary *Ch* outputs x^* as a response to the query (m^*, r^*, H^*, H^{j^*}) and the probability of this event is $\frac{1}{q_g q_h}$.

Ambiguity

The ambiguity means that, if we have two potential signers i and j, for any third entity it is infeasible to distinguish if a signature σ is either produced by the user i or j.

Let *Ch* be a challenger and $\mathcal{A}$ an adversary against the ambiguity of the concurrent signature scheme, the adversary $\mathcal{A}$ makes a sequence of $KGen$, $KReveal$, $ASign$, and $KeyExtract$ queries as it is described in the unforgeability game, the challenger *Ch* chooses random public keys $H^{(i)}$ and $H^{(j)}$ and makes $ASign$ query on $(H^{(i)}, H^{(j)}, x, M)$, then *Ch* chooses randomly $b \in \{i, j\}$ and returns a signature $\sigma_b = ASign(\mathcal{P}, p_k^{(b)}, s_k^{(b)}, p_k^{(a)}, x, m)$.
The probability of a signer to produce a signature σ_i given $(H^{(i)}, H^{(j)}, x, M)$ is equal to $\frac{1}{2^{\lambda_0}}$ which is equal to the probability to choose randomly $r_i \in \{0,1\}^{\lambda_0}$ where a signature $\sigma_i = (e', r, x)$. Consequently, the probability of a signer to produce an ambiguous signature is equal to $\frac{1}{2^{\lambda_0}}$ which is the same probability for the j^{th} signer. Thus, we conclude that a signature generated by the i^{th} signer is undistinguishable from one produced by the j^{th} signer. The conclusion is that the adversary $\mathcal{A}$ returns $b' = b$ with probability equal to $\frac{1}{2}$.

Fairness

By fairness, we mean that no one of the signers (either the initial or the matching signer) is able to return a malicious keystone where the ambiguous signature and this forged keystone passes the verification Algorithm 7 $Verify$.

We consider a challenger *Ch* that runs the **Setup** Algorithm 3 and sends the public parameters $\mathcal{P}$ and users public keys to the adversary $\mathcal{A}$ who makes q_g and q_h queries to random oracles $g(.)$ and $h(.)$, q_{kg} queries to the $KGen$ oracle and also a sequence of queries to $KReveal$, Private key Extract and $ASign$ oracles, the challenger *Ch* sends the answers to the adversary $\mathcal{A}$ as in the unforgeability game. We suppose that the adversary $\mathcal{A}$ wins the fairness property with non negligible probability, chooses the challenge public keys $p_k^{(i)^*}$ and $p_k^{(j)^*}$ outputs a keystone $\kappa^* \in \mathcal{K}$, and a signature $\sigma^* = (e'^*, r^*, x^*)$ where $AVerify(\mathcal{P}, p_k^{i*}, p_k^{j*}, \sigma^*, m^*) = 1$. The adversary wins the game if either of the following cases hold:

- Case 1: If x^* is a previous output from a $KGen$ query, no KReveal query on input x^* was made, and if $Verify(\mathcal{P}, p_k^{i*}, p_k^{j*}, \sigma^*, \kappa^*, m^*) = 1$.
- Case 2: If $\mathcal{A}$ also produces $\sigma^* = (e'^*, r^*, x^*)$, message $m' \in \mathcal{M}$, where $AVerify(\mathcal{P}, p_k^{i*}, p_k^{j*}, \sigma', m') = 1$ and (κ^*, σ^*) we have $Verify(\mathcal{P}, p_k^{(i)^*}, p_k^{(j)^*}, \sigma^*, \kappa^*, m^*) = 1$ but $Verify(\mathcal{P}, p_k^{i*}, p_k^{j*}, \sigma', \kappa^*, m') = 0$

Hereafter, we analyse in details the two cases:

- Analysis of case 1: In this case, the adversary $\mathcal{A}$ is assumed to return, with non negligible probability, a valid signature $\sigma^* = (e'^*, r^*, x^*)$ and a keystone κ^* where for (κ^*, σ^*) we have $Verify(\mathcal{P}, pk_i^*, pk_j^*, \sigma^*, \kappa^*, m^*) = 1$. This means that $\mathcal{C}$ returns the tuple (κ^*, x^*) with non negligible probability such that $g(H^{(i)}.\kappa^{*T}) = x^*$ where κ^{*T} is a keystone and x^* is the output of $KGen$ Algorithm 4 without making $KReveal$ query on x^* . The probability of returning the tuple (κ^*, x^*) by $\mathcal{A}$ is negligible.
- Analysis of case 2: We suppose that $\mathcal{A}$ returns a signature $\sigma' = (e'', r', x^*)$ as another valid ambiguous signature for public keys H_i^* and H_j^* on the message m' such that for σ', $AVerify(\mathcal{P}, p_k^{i*}, p_k^{j*}, \sigma', m') = 1$ but $Verify(\mathcal{P}, p_k^{i*}, p_k^{j*}, \sigma', \kappa^*, m') = 0$ on the input (σ', κ^*). On the other hand, σ^* is a valid ambiguous signature, which implies that $AVerify$ for that returns 1, and by assumption $Verify$ algorithm on input(κ^*, σ^*) outputs 1, then $x^* = g(H^{(i)}.\kappa^{*T})$ is fulfilled. Since σ^* and σ' shares the same value x^* and also $AVerify$ on input σ' returns 1, as a consequence we have $Verify(\mathcal{P}, p_k^{i*}, p_k^{j*}, \sigma', \kappa^*, m') = 1$ and this is a contradiction with the assumption which states that $Verify(\mathcal{P}, pk_i^*, pk_j^*, \sigma', \kappa^*, m') = 0$.

VI. Parameters and Results

In this section, we present the parameters of our concurrent signature scheme namely for signature size and public key length for a large scale of security levels. We also evaluate our construction by computing the computational cost of different lgorithms including $KGen$, $ASign$, $AVerify$ and $Verify$ (Algorithms 4, 5, 6, 7 respectively).

- The signer's public key size: The public key is a parity-check matrix, which requires $n \times (n-k)$ bits

$$\begin{aligned} size(p_k) &= size_{(H)} \\ &= (n \times (n-k))bits \end{aligned}$$

- The signature size is computed as follows:

$$\begin{aligned} size(\sigma) &= size(e') + size(r) + size(x) \\ &= (2 \times (q-1) \times n + \lambda_0)bits \end{aligned}$$

For a securty level λ we have: $n = 66.34\lambda$, $w = 0.9396n$,$k_U = 0.8379n$, $k_V = 0.4821n$, $size(p_k) = 1565\lambda^2$

In Table I, we give the parameters used for different security levels, we present the signature size and the public key length. In table II, we compare the security requierements with some related works and then in table III, we compare the public key and signature size of our proposal and the scheme in [9]

The practical results presented in Tables I and III show clearly that our schemes produce signature of smaller public key size than the schemes presented in [9]. For example, we reach a public key equal to $3200KB$ insted of $99000KB$ for 128 bit security level witch is equivalent to a reduction of 96% of the public key size.

TABLE I
Parameters for different security levels.

Security level	n	k_U	k_V	ω	λ_0	Public key size	Signature size
80 bits	5308	2224	1280	4987	240	1250 KB	2.6 KB
128 bits	8492	3558	2047	7980	384	3200 KB	4.2 KB
256 bits	16984	7116	3738	15957	768	12800 KB	8.5KB
512 bits	33971	14028	7477	31916	1536	51282 KB	16.7KB
1024 bits	67942	28056	14954	63831	3072	205127 KB	33.3KB

TABLE II
Comparison in terms of security properties with some related works.

Scheme	Correctness	Unforgeability	Ambiguity	Fairness
Scheme of [9]	Yes	Yes	Yes	Yes
Scheme of [8]	Yes	Yes	Yes	Yes
Our scheme	Yes	Yes	Yes	Yes

TABLE III
Comparison in terms of public key and signature sizes.

Scheme	Hard problem	Security model	Signature size	Public key size
Scheme of [9]	Syndrome Decoding	Random oracle	0.07 KB	99000 KB
Our scheme	Rank Syndrome Decoding	Random oracle	4.2 KB	3200 KB

VII. Conclusion

Since in the literrature there is just one conccurent signature scheme that is based on coding theory, our scheme is the second one and reduces significantly the public key size which makes our proposal very efficient and prartical in different real life applications. Throughout this article, a

concurrent signature scheme is proposed using coding theory hard problems. Our construction relies on error correcting codes assumptions based on the Hamming metric as a promising alternative to classical cryptography in the era of quantum computers. In addition, the proposed approach uses the generalized $(U|U+V)$ codes and makes a trade-off between efficiency and security requirements since we achieve efficient results particularly in terms of public key size if we compare it with some post-quantum constructions. Our scheme maintain the security requirements including correcteness, unforgeability, ambiguity and fairness.

REFERENCES

[1] L. Chen, C. Kudla, and K. Paterson, "Concurrent signatures," in *International Conference on the Theory and Applications of Cryptographic Techniques*, ser. Lecture Notes in Computer Science, vol. 3027. Springer, 2004, pp. 287–305.

[2] W. Susilo, Y. Mu, and F. Zhang, "Perfect concurrent signature schemes," in *International Conference on Information and Communications Security*, ser. Lecture Notes in Computer Science, vol. 3269. Springer, 2004, pp. 14–26.

[3] S. Chow and W. Susilo, "Generic construction of (identity-based) perfect concurrent signatures," in *International Conference on Information and Communications Security*, ser. Lecture Notes in Computer Science, vol. 3783. Springer, 2005, pp. 194–206.

[4] K. Nguyen, "Asymmetric concurrent signatures," in *International Conference on Information and Communications Security*, ser. Lecture Notes in Computer Science, vol. 3783. Springer, 2005, pp. 181–193.

[5] D. Tonien, W. Susilo, and R. Safavi-Naini, "Multi-party concurrent signatures," in *International Conference on Information Security*, ser. Lecture Notes in Computer Science, vol. 4176. Springer, 2006, pp. 131–145.

[6] B. Li, G. Xu, and Y. Zhao, "Attribute-based concurrent signatures," in *Proceedings of the 6th International Conference on Information Engineering*, 2017, pp. 1–7.

[7] S. Wang, L. Liu, J. Chen, J. Sun, X. Zhang, and Y. Zhang, "Lattice-based multi-party concurrent signatures scheme," in *2013 5th International Conference on Intelligent Networking and Collaborative Systems*. IEEE, 2013, pp. 568–572.

[8] X. Xiang, H. Li, M. Wang, and X. Zhao, "Efficient multi-party concurrent signature from lattices," *Information Processing Letters*, vol. 116, no. 8, pp. 497–502, 2016.

[9] M. Asaar, M. Ameri, M. Salmasizadeh, and M. Aref, "A provably secure code-based concurrent signature scheme," *IET Information Security*, vol. 12, no. 1, pp. 34–41, 2017.

[10] L. Dallot, "Towards a concrete security proof of courtois, finiasz and sendrier signature scheme," in *Research in Cryptology: Second Western European Workshop, WEWoRC 2007, Bochum, Germany, July 4-6, 2007, Revised Selected Papers 2*, ser. Lecture Notes in Computer Science, vol. 4945. Springer, 2007, pp. 65–77.

[11] J. Shetty, G. Sudhakara, and V. Madhusudanan, "Encryption system involving matrix associated with semigraphs," *IAENG International Journal of Applied Mathematics*, vol. 52, no. 2, pp. 458–465, 2022.

[12] S. Belabssir, "The weight enumerator of some families of linear error-block codes," *IAENG International Journal of Computer Science*, vol. 49, no. 3, pp. 728–735, 2022.

[13] S. Belabssir and N. Sahlal, "Tensor product and linear error block codes," *IAENG International Journal of Applied Mathematics*, vol. 51, no. 2, pp. 279–283, 2021.

[14] P. Shor, "Algorithms for quantum computation: Discrete logarithms and factoring," in *35th Annual Symposium on Foundations of Computer Science, Santa Fe,New Mexico, USA, 20-22 November 1994*, 1994, pp. 124–134.

[15] H. Assidi, E. Ayebie, and E. Souidi, "A code-based group signature scheme with shorter public key length," in *Proceedings of the 13th International Joint Conference on e-Business and Telecommunications (ICETE 2016)*, vol. 4, 2016, pp. 432–439.

[16] H. Assidi and E. Souidi, "Strong designated verifier signature based on the rank metric," in *IFIP International Conference on Information Security Theory and Practice*, ser. Lecture Notes in Computer Science, vol. 12024. Springer, 2019, pp. 85–102.

[17] T. Debris-Alazard, N. Sendrier, and J. Tillich, "Wave: A new family of trapdoor one-way preimage sampleable functions based on codes," in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2019, pp. 21–51.

[18] E. Berlekamp, R. McEliece, and H. Van Tilborg, "On the inherent intractability of certain coding problems (corresp.)," *IEEE Transactions on Information Theory*, vol. 24, no. 3, pp. 384–386, 1978.