High-Memory Masked Convolutional Codes for Post-Quantum Cryptography

Meir Ariel

Faculty of Engineering, Tel Aviv University, Tel Aviv 6997801, Israel
meirariel@tauex.tau.ac.il

Abstract. This paper introduces a novel post-quantum encryption scheme based on high-memory masked convolutional codes, offering flexible and robust security compared to conventional code-based methods that rely on block codes with fixed parameters and limited error correction. Our approach supports arbitrary plaintext lengths and adapts to diverse code families with varying complexity and security levels. The scheme scales efficiently with linear decryption complexity, maintaining consistent computational costs regardless of plaintext size. Security is strengthened through the high-rate injection of random errors, with additional noise introduced via polynomial division. Semi-invertible transformations further enhance cryptographic strength by generating dense, random-like matrices. Our method significantly increases cryptanalytic resistance, surpassing the security of 'Classic McEliece' by factors over 2^{100}. Moreover, integrating the Viterbi algorithm supports efficient hardware implementation, making the scheme practical for real-world quantum-resistant encryption.

Keywords: Code-based cryptography · Post-quantum cryptography · Convolutional codes.

1 Introduction

Code-based cryptography, introduced by Robert McEliece [1] in 1978, employs binary Goppa codes to build a public key encryption scheme. The core concept disguises an error-correcting code using invertible linear transformations. The sender deliberately introduces errors into the ciphertext (i.e., the codeword), making decryption challenging for attackers while allowing the recipient, who possesses the private key, to correct the errors and retrieve the plaintext.

Since its inception, numerous modifications to the McEliece cryptosystem have emerged, mostly using different codes. However, most have failed to maintain comparable security [2-3]. Consequently, Classic McEliece reached Round 4 of the NIST standardization process for Post-Quantum Cryptography (PQC) [4] but was not selected for final adoption. Despite its strong theoretical basis, Classic McEliece has practical limitations, including inflexible code selection with limited error correction capabilities (essentially the $(N, K, t) = (1024, 524, 50)$

and $(4096, 3556, 45)$ Goppa codes, where $K \times N$ are the generator matrix dimensions and t is the error correction capacity). These constraints limit adaptability to varying security requirements.

Our work introduces a novel PQC approach using high-memory masked convolutional codes, offering several advantages over traditional code-based cryptosystems:

Diverse Code Selection: A wide range of convolutional codes can form the public and private keys, allowing customization to meet specific performance and security needs.

Stronger Public Key Security: Our method's high-density generating matrix with a random-like structure significantly improves security against cryptanalysis compared to the low-density matrices of other code-based systems [5]. The flexibility to select convolutional codes with strong error-correcting capabilities enables the introduction of a higher (but unknown) number of random errors, boosting cryptanalysis resistance by factors exceeding 2^{100}, depending on key length, compared to Classic McEliece.

Scalable Key Length and Decoding Complexity: Unlike block codes with fixed parameters, our scheme supports plaintexts of arbitrary length, meeting diverse security needs. The recipient's decoding complexity scales linearly with key length.

Efficient Hardware Implementation: The decoding process utilizes the Viterbi algorithm, based on directed graph methods, facilitating efficient and straightforward hardware implementation.

The paper is structured as follows: Section 2 describes the construction of public and private keys, forming our cryptographic framework's core. Section 3 covers the sender's encryption process, while Section 4 details the recipient's decryption method. Section 5 examines potential eavesdropping attacks and their impact. Section 6 discusses the rationale behind our polynomial selection strategy, crucial for security. Section 7 provides a worked example demonstrating practical implementation. Finally, Section 8 analyzes resistance to cryptanalysis and complexity metrics, highlighting security benefits over traditional block-code-based systems, and Section 9 concludes with key findings.

2 Public and Private Key Construction

In this paper, we denote binary vectors using bold letters (e.g., $\boldsymbol{a}, \boldsymbol{b}$), while their corresponding polynomial representations are written as $\boldsymbol{a}(x), \boldsymbol{b}(x)$. The notations $\boldsymbol{a}$ and $\boldsymbol{a}(x)$ are used interchangeably depending on the context. We follow standard conventions from error-correcting code theory: $\boldsymbol{m}$ denotes an information sequence, $\boldsymbol{c}$ and $\boldsymbol{d}$ represent codewords (from different codes), $\boldsymbol{s}$ denotes a syndrome, and $\boldsymbol{r}$ represents a CRC polynomial. Sets of binary vectors are denoted using calligraphic letters, such as $\mathcal{D}$ and $\mathcal{L}$. Polynomial generator matrices associated with convolutional codes are denoted by uppercase letters (e.g., $A(x), B(x)$), with their corresponding scalar representations written as A and B.

Given n vectors:

$$\boldsymbol{v} = (v_0, v_1, v_2, \ldots), \boldsymbol{u} = (u_0, u_1, u_2, \ldots), \boldsymbol{w} = (w_0, w_1, w_2, \ldots), \ldots$$

the interleaving operation, denoted $\boldsymbol{u} \wedge \boldsymbol{v} \wedge \boldsymbol{w}, \ldots$ produces a new vector by alternating the elements of the input vectors:

$$(\boldsymbol{v} \wedge \boldsymbol{u} \wedge \boldsymbol{w} \ldots) = (v_0, u_0, w_0, \ldots v_1, u_1, w_1, \ldots v_2, u_2, w_2, \ldots) \tag{1}$$

To Deinterleave an interleaved vector and extract one of its constituent components, we define the following operation:

$$(\boldsymbol{v} \wedge \boldsymbol{u} \wedge \boldsymbol{w} \ldots)_i \tag{2}$$

This operation extracts the elements located at positions $i, i+n, i+2n, \ldots$ Specifically $(\boldsymbol{v} \wedge \boldsymbol{u} \wedge \boldsymbol{w} \ldots)_0 = \boldsymbol{v}$.

Denote by $\boldsymbol{p}_i(x)$ a binary polynomial of memory up to p

$$\boldsymbol{p}_i(x) = \sum_{j=0}^{p} a_j x^j, \quad \text{where} \quad a_j \in \mathbb{F}_2 \tag{3}$$

A Convolutional Code (CC) is defined by a set of n constituent polynomials, customarily structured into the form of a polynomial generator matrix, denoted as

$$G_P(x) = [\boldsymbol{p}_0(x), \boldsymbol{p}_1(x), \ldots, \boldsymbol{p}_{n-1}(x)] \tag{4}$$

The matrix $G_P(x)$ determines both the rate and the error-correction capability of the CC, which is typically characterized by its free distance, d_{free}. The parameter p determines the number of states, 2^p, in the trellis diagram—a directed graph representing the code structure—and thus influences the complexity of decoding using the Viterbi algorithm. For simplicity, we consider good CCs of rate $1/n$, although the construction presented herein is applicable to any CC, including punctured codes. The matrix $G_P(x)$ has a scalar representation G_P given by:

$$G_P = \begin{bmatrix} \boldsymbol{g}_0 & \boldsymbol{g}_1 & \boldsymbol{g}_2 & \ldots & \boldsymbol{g}_p & \mathbf{0} & \mathbf{0} & \mathbf{0} \ldots \\ \mathbf{0} & \boldsymbol{g}_0 & \boldsymbol{g}_1 & \ldots & \boldsymbol{g}_{p-1} & \boldsymbol{g}_p & \mathbf{0} & \mathbf{0} \ldots \\ \mathbf{0} & \mathbf{0} & \boldsymbol{g}_0 & \ldots & \boldsymbol{g}_{p-2} & \boldsymbol{g}_{p-1} & \boldsymbol{g}_p & \mathbf{0} \ldots \\ \vdots & & & & & & & \end{bmatrix} \tag{5}$$

where $\boldsymbol{g}_i$ is the $1 \times n$ matrix of coefficients of x^i (in the general case $\boldsymbol{g}_i$ is a $k \times n$ scalar matrix). For example, for $G_P(x) = [1 + x^2, 1 + x + x^2]$ we have $\boldsymbol{g}_0 = [11], \boldsymbol{g}_1 = [01], \boldsymbol{g}_2 = [11]$ and $\mathbf{0} = [00]$. To each polynomial $\boldsymbol{p}_i(x)$ we associate a high-memory polynomial $\boldsymbol{q}_i(x)$ with degree up to q,

$$\boldsymbol{q}_i(x) = \sum_{j=0}^{q} b_j x^j, \quad \text{where} \quad b_j \in \mathbb{F}_2 \tag{6}$$

The set of these n high-memory polynomials forms the polynomial matrix:

$$G_Q(x) = [\boldsymbol{q}_0(x), \boldsymbol{q}_1(x), \ldots, \boldsymbol{q}_{n-1}(x)] \tag{7}$$

The proposed algorithm imposes no restriction on the choice of $G_P(x)$ and $G_Q(x)$, and their polynomials may be reducible or irreducible. However, in practice the degrees p and q are chosen such that $p+q > 200$ and $p \ll q$. Furthermore, the selection of $G_Q(x)$ significantly affects the propagation of the error during decryption (as discussed in Section 6), which requires an appropriate choice to maintain a tolerable error rate in the decoder. The high-memory polynomial generator matrix $G_{PQ}(x)$ is defined as:

$$G_{PQ}(x) = [\boldsymbol{p}_0(x)\boldsymbol{q}_0(x), \boldsymbol{p}_1(x)\boldsymbol{q}_1(x), \ldots, \boldsymbol{p}_{n-1}(x)\boldsymbol{q}_{n-1}(x)] \tag{8}$$

A corresponding finite-dimensional high-memory scalar generator matrix is given by

$$G_P = \begin{bmatrix} \boldsymbol{g}_0 & \boldsymbol{g}_1 & \boldsymbol{g}_2 & \cdots & \boldsymbol{g}_{p+q} & \mathbf{0} & \cdots & & & \mathbf{0} \\ \mathbf{0} & \boldsymbol{g}_0 & \boldsymbol{g}_1 & \cdots & \boldsymbol{g}_{p+q-1} & \boldsymbol{g}_{p+q} & \mathbf{0} & \cdots & & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \boldsymbol{g}_0 & \cdots & \boldsymbol{g}_{p+q-2} & \boldsymbol{g}_{p+q-1} & \boldsymbol{g}_{p+q} & \mathbf{0} & \cdots & \mathbf{0} \\ \vdots & & & & & & & & & \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & \boldsymbol{g}_0 & \boldsymbol{g}_1 & \cdots & \boldsymbol{g}_{p+q-2} & \boldsymbol{g}_{p+q-1} & \boldsymbol{g}_{p+q} \end{bmatrix} \tag{9}$$

The number K of rows of G_{PQ} and the corresponding number of columns, given by $N = n(K+p+q)$, can be determined by the owner of the key. Multiplying each polynomial of $G_P(x)$ by a high-memory polynomial substantially increases the run-length of each row in G_{PQ}, i.e., the length of a sequence within the row that begins and ends with "1" compared to G_P. When employed for error correction, the $K \times N$ matrix G_{PQ} corresponds to a CC with memory length $p+q$, that can be described using a trellis diagram comprising N/n segments and up to 2^{p+q} states. A conventional trellis diagram starts from a single state and expands to 2^{p+q} states over $p+q$ segments. Additionally, $p+q$ zeroes need be appended at end of the information sequence to drive the trellis to a single state. In principle, a Viterbi decoder can be used for maximum-likelihood hard-decision decoding (based on a Hamming distance metric) of the CC. However, this approach becomes impractical for large values of 2^{p+q}.

Since G_{PQ} has finite dimensions, the corresponding CC exhibits block code properties, defined by a generator matrix with a structured diagonal pattern. Each row is obtained by a right shift of n bits relative to the previous row. Although the code description is intricate, permuting the columns of G_{PQ} does not obscure its structure, as the position of zero sequences in each column still reveal discernible patterns.

To obfuscate G_{PQ}, disrupting its diagonal structure while maximizing the run length, we introduce a $K \times N$ masking matrix $\tilde{G}$, where each row is randomly selected from a predefined set $\mathcal{L}$ of l random binary vectors of length N. Notably, the set $\mathcal{L}$ can be constructed from the same n-tuple of Hamming weight approximately $n/2$, repeated N/n times. This structured construction is primarily chosen to simplify notation and clarify the decryption process, without significantly compromising resistance to cryptanalysis. Furthermore, it maintains the same computational complexity as using fully random vectors. For example, when $n = 2$, there are only two such vectors:

$$\mathcal{L} = \{(101010\ldots10), (010101\ldots01)\} \tag{10}$$

For $n = 4$, the set $\mathcal{L}$ could be taken as:

$$\begin{aligned}\mathcal{L} = \{&(11001100\ldots1100), (10101010\ldots1010), (10011001\ldots1001),\\ &(01100110\ldots0110), (01010101\ldots0101), (00110011\ldots0011)\}\end{aligned} \tag{11}$$

The masked generator matrix is then constructed as $G_{PQ} + \tilde{G}$, where addition is performed over $\mathbb{F}_2$. Finally, the encryption matrix, or public key, is obtained by applying row and column transformations:

$$G = S(G_{PQ} + \tilde{G})R \tag{12}$$

where S is a random non-singular $K \times K$ binary matrix used to hide the encoding—that is correspondence between plaintexts (information words) and ciphertexts (codewords). The matrix R is a random $N \times N$ permutation matrix. We refer to the code described by G, a high-memory masked Convolutional Code (MCC).The transformation from G_P to G involves several steps, some reversible and others semi-reversible, ultimately producing a fully dense, random-like matrix structure.

At the decoder, in addition to applying the inverse permutation, the effect of the masking matrix $\tilde{G}$ must be reversed. However, this masking operation is not fully reversible. Although the recipient (the owner of the public key) knows the exact value of $\tilde{G}$, the plaintext remains unknown. Consequently, the specific linear combination of the rows of $\tilde{G}$ added to the ciphertext by the sender cannot be unambiguously determined by the recipient's decoder. The approach to addressing this challenge is explained in Section 4. Moreover, the masking introduced by the high-memory polynomials must also be accounted for prior to decoding. A trellis decoder with feasible complexity can operate only on G_P and not on its high-memory variant G_{PQ}. Fully inverting the multiplication by the high-memory polynomials is impractical in the presence of errors and may even result in error propagation at the decoder, increasing the likelihood of decryption failure for the recipient. However, this additional error propagation also raises the complexity of any potential attack. An appropriate choice of $G_Q(x)$ can help moderate this error propagation.

The final step in constructing the public key is to define an error-detection code, such as a cyclic redundancy check (CRC), determined by a polynomial $\boldsymbol{r}(x)$ of degree r. This additional layer of encoding, applied to the plaintext, allows for the detection of possible decoding failures at the recipient's end. This step is necessary because, unlike block codes, successful decoding of a CC is not guaranteed, even when the number of errors is known. We are now ready to define the public key as:

$$\{G, e, r\} \tag{13}$$

and the private key as:

$$\{S, R, G_P(x), G_Q(x), \tilde{G}\} \tag{14}$$

3 Encryption by Sender

Assume that the sender possesses the public key. The following steps are performed by the sender to generate the ciphertext:

Step1: Generation of Plaintext
A random plaintext $\boldsymbol{m}$ of length $K - r$ bits is generated. Its polynomial representation is denoted by $\boldsymbol{m}(x)$.

Step 2: Appending CRC
To append r CRC bits to $\boldsymbol{m}$, the following procedure is used: multiply $\boldsymbol{m}(x)$ by x^r (effectively shifting it); divide $x^r\boldsymbol{m}(x)$ by the polynomial $\boldsymbol{r}(x)$ and compute the remainder; append the remainder to $\boldsymbol{m}$ to form a binary vector of length K, denoted as $\boldsymbol{m}_r$.

Step 3: Codeword Generation
The codeword $\boldsymbol{c}$ of length N is calculated as:

$$\boldsymbol{c} = \boldsymbol{m}_r G. \tag{15}$$

Step 4: Error Introduction
Random errors are introduced to $\boldsymbol{c}$ by flipping each bit with probability e. The actual number of errors injected by the sender is unknown. Denote the resulting ciphertext as:

$$\boldsymbol{c}_e = \boldsymbol{c} + \boldsymbol{e} \tag{16}$$

where $\boldsymbol{e}$ is the random error vector generated by the sender. Due to polynomial division at the decoder, the effective error weight may increase further, contributing to additional obfuscation.

Step 5: Transmission
The ciphertext $\boldsymbol{c}_e$ is then transmitted by the sender to the recipient.

Typically, e will be chosen such that the Hamming weight of $\boldsymbol{e}$, denoted $\mathrm{wt}(\boldsymbol{e})$, is sufficiently large to significantly increase resistance to attacks. Since $\boldsymbol{e}$ is randomly generated, it may contain more than eN errors or clusters of errors that could cause a decoding failure. However, thanks to the superior error correction capabilities of the CC defined by G_P, K and N, the probability of decoding failure remains very low even for relatively large values of e. In a practical system, any (rare) decoding failures will be detected by the CRC, enabling the recipient either to select the next most likely candidate plaintext or to request a retransmission.

4 Decryption by the Recipient

The recipient possesses the public and private keys along with the received ciphertext, i.e.,

$$\{S, R, G_P(x), G_Q(x), \tilde{G}, G, e, \boldsymbol{r}, \boldsymbol{c}_e\} \tag{17}$$

The following steps are performed by the recipient to decrypt the ciphertext:

Step 1: Inverse Permutation
Apply the inverse permutation to $\boldsymbol{c}_e$ to obtain $\tilde{\boldsymbol{c}}$. (Note that if R is a permutation matrix, then $R^{-1} = R^T$). Thus,

$$\tilde{\boldsymbol{c}} = \boldsymbol{c}_e R^T \tag{18}$$

Since $\boldsymbol{e}$ is a random error vector with an unknown Hamming weight,

$$\boldsymbol{e}R^T = \boldsymbol{c}_e R^T - \boldsymbol{c}R^T \tag{19}$$

is simply another random error vector with the same weight, i.e.,

$$\mathrm{wt}(\boldsymbol{e}R^T) = \mathrm{wt}(\boldsymbol{e}) \tag{20}$$

Step 2: unmasking
Reverse the masking introduced by the summation of $\tilde{G}$ and G_{PQ}. Denote by $\mathrm{LS}(\mathcal{L})$ the linear span of the set $\mathcal{L}$. Since $\mathcal{L}$ contains l vectors, we have $\mathrm{LS}(\mathcal{L}) \leq 2^l$. The vector $\tilde{\boldsymbol{c}}$ can be expressed as:

$$\tilde{\boldsymbol{c}} = \boldsymbol{m}_r S G_{PQ} + \boldsymbol{m}_r S \tilde{G} = \boldsymbol{m}_r S G_{PQ} + \boldsymbol{l}_i, \quad \text{with} \quad \boldsymbol{l}_i \in \mathrm{LS}(\mathcal{L}) \tag{21}$$

However, the value of $\boldsymbol{l}_i$ is unknown to the recipient and may be any member of $\mathrm{LS}(\mathcal{L})$. Therefore, decoding must be attempted for all possible candidates in the set $\mathcal{M}$ of unmasked vectors:

$$\mathcal{M} = \{\tilde{\boldsymbol{c}} - \boldsymbol{l}_i \mid \boldsymbol{l}_i \in \mathrm{LS}(\mathcal{L})\} \tag{22}$$

For example, if the CC has rate 1/4 (i.e., $n = 4$) and the set $\mathcal{L}$ is chosen as in Equation (11), then

$$\mathrm{LS}(\mathcal{L}) = \{(0000\ldots0000), (1000\ldots1000), (0100\ldots0100), \ldots, (1111\ldots1111)\} \tag{23}$$

so that $|\mathrm{LS}(\mathcal{L})| = 16$. Consequently, decoding must be applied in parallel to all 16 members of $\mathcal{M}$.

Step 3: Inverting the High-Memory Polynomial Multiplication
The next step involves reversing the multiplication by the high-memory polynomials $G_Q(x)$. Using polynomial representations, each member of $\mathcal{M}$ can be regarded as the result of interleaving the following n polynomials, each of length N/n:

$$(\tilde{\boldsymbol{c}} - \boldsymbol{l}_i)_0 \wedge (\tilde{\boldsymbol{c}} - \boldsymbol{l}_i)_1 \wedge \ldots \wedge (\tilde{\boldsymbol{c}} - \boldsymbol{l}_i)_{n-1} \tag{24}$$

where the elements of $(\tilde{\boldsymbol{c}} - \boldsymbol{l}_i)_j$ appear at the jth, $(n + j)$th, $(2n + j)$th, ... and $\left(\frac{N}{n} - 1 + j\right)$th positions of the interleaved vector $\tilde{\boldsymbol{c}} - \boldsymbol{l}_i$. Therefore, each such polynomial, denoted $(\tilde{\boldsymbol{c}}(x) - \boldsymbol{l}_i(x))_j$, shall be divided by the corresponding polynomial $\boldsymbol{q}_i(x)$ to obtain a quotient polynomial, denoted by $(\boldsymbol{d}_i(x))_j$. The value of $(\boldsymbol{d}_i(x))_j$ naturally depends on the value of $\boldsymbol{l}_i(x)$. For simplicity, assume that $\mathcal{L}$ was selected as in Equation (10). Under this condition, each polynomial

$(\tilde{\boldsymbol{c}}(x) - \boldsymbol{l}_i(x))_j$ arises from masking with either an all-zero masking polynomial $\mathbf{0}(x)$, where $\boldsymbol{l}_i(x) = \mathbf{0}(x)$ or an all-one masking polynomial $\mathbf{1}(x)$, where $\boldsymbol{l}_i(x) = \mathbf{1}(x)$. This results with two possible quotient values. When $\boldsymbol{l}_i(x) = \mathbf{0}(x)$

$$(\boldsymbol{d}_0(x))_j = \frac{(\tilde{\boldsymbol{c}}(x) - \mathbf{0}(x))_j}{\boldsymbol{q}_i(x)} \tag{25}$$

When $\boldsymbol{l}_i(x) = \mathbf{1}(x)$

$$(\boldsymbol{d}_1(x))_j = \frac{(\tilde{\boldsymbol{c}}(x) - \mathbf{1}(x))_j}{\boldsymbol{q}_i(x)} \tag{26}$$

At this stage of decryption, the recipient cannot determine which quotient, $(\boldsymbol{d}_0(x))_j$ or $(\boldsymbol{d}_1(x))_j$, is the correct one for index j. Furthermore, since $(\tilde{\boldsymbol{c}}(x) - \boldsymbol{l}_i(x))_j$ may contain errors, the division might yield non-zero remainders. These remainders can either be ignored or be detected and subtracted if $\boldsymbol{q}_i(x)$ is carefully chosen to function as an error-detecting code. When ignored, we assume that errors left in the quotient will be corrected by the Viterbi decoder in Step 5 below.

Step 4: Quotient Interleaving
The quotients are re-interleaved to form a vector $\boldsymbol{d}_i$:

$$\boldsymbol{d}_i = (\boldsymbol{d}_i)_0 \wedge = (\boldsymbol{d}_i)_1 \wedge ... \tag{27}$$

All candidates need be considered. In the general case, there are no more than 2^l variants in the set of interleaved quotients, denoted as

$$\mathcal{D} = \{\boldsymbol{d}_i\}_{i=0}^{2^l-1} \tag{28}$$

Step 5: Parallel Viterbi Decoding
We now aim to determine $\hat{\boldsymbol{m}}_r$, the most likely value of $\boldsymbol{m}_r$, (i.e., the plaintext with the appended CRC), by applying Viterbi decoding to all possible members of the set $\mathcal{D}$. Since $\mathcal{D}$ contains 2^l candidate vectors, these can be decoded in parallel for improved efficiency. Notably, there exists a key distinction among the 2^l members of $\mathcal{D}$. One specific variant, $\boldsymbol{d}_i$, will be decoded such that the most likely codeword is found at a Hamming distance of approximately $eN + \alpha$ from $\boldsymbol{d}_i$, where α represents the number of additional errors introduced by the polynomial division process. The value of α can be estimated by simulations, as described in Section 6. For all other candidate vectors in $\mathcal{D}$ the most likely decoded codeword will typically exhibit a significantly larger Hamming distance than $eN + \alpha$. This distinction arises because subtracting an incorrect masking vector from $\tilde{\boldsymbol{c}}$ effectively introduces a substantial number of errors into $\boldsymbol{d}_i$. The most likely codeword $\hat{\boldsymbol{d}}$ is determined by

$$\hat{\boldsymbol{d}} = \boldsymbol{d}_i - \hat{\boldsymbol{e}} \tag{29}$$

where $\hat{\boldsymbol{e}}$ is the error vector with the minimum Hamming weight among the outcomes of all 2^l parallel Viterbi decoders. The index i corresponds to the decoder that processes the correct $\boldsymbol{d}_i$. The same Viterbi decoder also reveals

$\hat{\boldsymbol{m}}_r S$, the transformed plaintext with appended CRC that generated $\hat{\boldsymbol{d}}$.

Step 6: Plaintext Recovery
To recover the original plaintext, we invert the transformation induced by the matrix S:

$$\hat{\boldsymbol{m}}_r = \hat{\boldsymbol{m}}_r S S^{-1} \tag{30}$$

Next, the remainder obtained from dividing $\hat{\boldsymbol{m}}_r(x)$ by $\boldsymbol{r}(x)$ must be computed. If this remainder is zero, we declare that

$$\boldsymbol{m}_r = \hat{\boldsymbol{m}}_r \tag{31}$$

and recover the plaintext $\boldsymbol{m}$ by discarding the r CRC bits from $\boldsymbol{m}_r$. If the remainder is non-zero, the selected codeword $\hat{\boldsymbol{d}}$ is rejected, and the process continues iteratively with the next most likely candidate. This process is repeated until a valid plaintext is identified or until all candidates are exhausted. If no valid plaintext is found, a retransmission of another ciphertext is requested.

We remark that for a CC with memory length p, the information sequence is padded with p zero bits at the end to force the trellis to converge to a single termination state. Consequently, to obtain the most likely plaintext, these appended zeroes must also be discarded from the recovered sequence. This ensures that the final recovered message accurately represents the original plaintext. The encryption and decryption algorithms are illustrated in the block diagram of Fig. 1.

5 Eavesdropper Attack

An eavesdropper has access to the public key G and the intercepted ciphertext $\boldsymbol{c}_e$. However, since $\boldsymbol{c}_e$ contains errors, the eavesdropper cannot employ G^{-1} to recover $\boldsymbol{m}$. Instead, the adversary would need to perform an impractically large number of decryption iterations—each corresponding to a possible error vector—or attempt to reconstruct G_{PQ} by attacking the random-like structure of G. Such an attack must overcome multiple layers of obfuscation, including a random permutation of the columns of G, a random selection of a $K \times K$ non-singular binary matrix, and the substantial masking introduced by the product $S\tilde{G}R$. Moreover, knowing $G_{PQ}(x)$ does not necessarily reveal $G_P(x)$, since the high-memory products $\boldsymbol{p}_i(x)\boldsymbol{q}_i(x)$ can be chosen to have multiple factorizations, adding yet another layer of complexity.

Attack from a Known Convolutional Code: Even if the adversary possesses complete knowledge of the CC—that is, if they know the matrix $G_P(x)$—they would still face significant obstacles in decrypting the ciphertext. Since the Viterbi decoder requires the appropriately transformed received word as input, the adversary cannot simply apply the trellis corresponding to $G_P(x)$ to the intercepted cyphertext. Instead, they must first disentangle the masked ciphertext's structure. To achieve this, the attacker must identify the inverse permutation applied to the received ciphertext and recover the quotients obtained

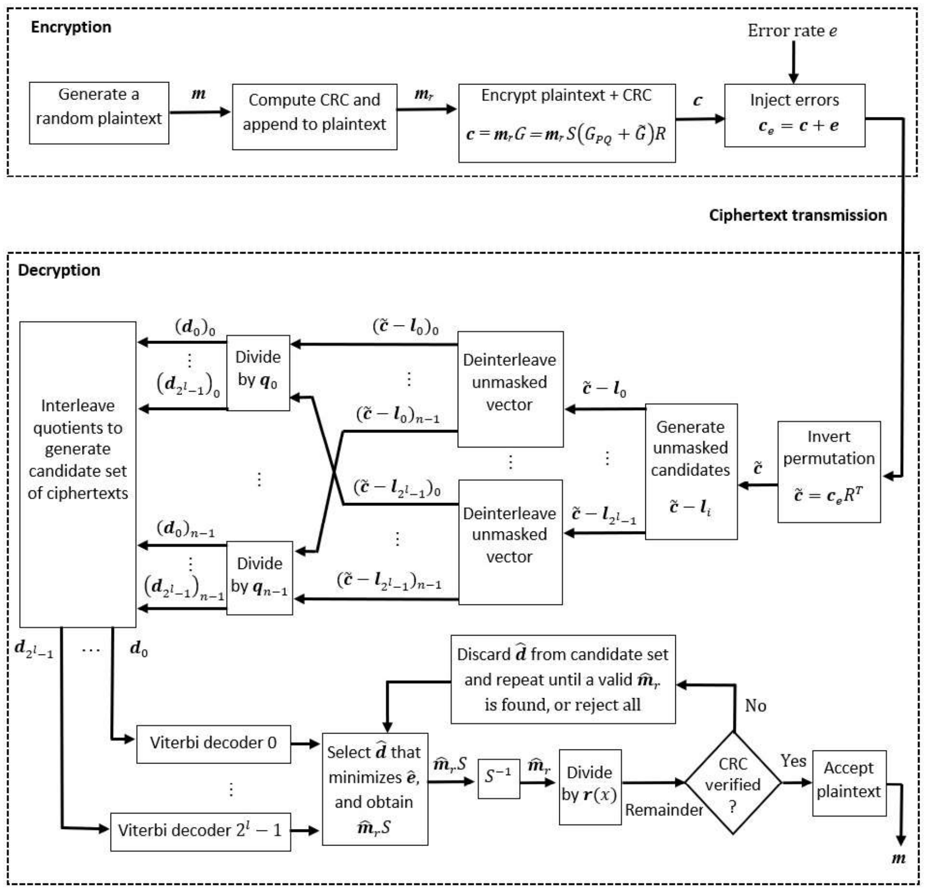
Encryption
Error rate e
Generate a random plaintext
$\boldsymbol{m}$
Compute CRC and append to plaintext
$\boldsymbol{m}_r$
Encrypt plaintext + CRC
$\boldsymbol{c} = \boldsymbol{m}_r G = \boldsymbol{m}_r S(G_{PQ} + \tilde{G})R$
$\boldsymbol{c}$
Inject errors
$\boldsymbol{c}_e = \boldsymbol{c} + \boldsymbol{e}$
Ciphertext transmission
Decryption
Invert permutation
$\tilde{\boldsymbol{c}} = \boldsymbol{c}_e R^T$
$\tilde{\boldsymbol{c}}$
Generate unmasked candidates
$\tilde{\boldsymbol{c}} - \boldsymbol{l}_i$
$\tilde{\boldsymbol{c}} - \boldsymbol{l}_0$
$\tilde{\boldsymbol{c}} - \boldsymbol{l}_{2^l-1}$
Deinterleave unmasked vector
Deinterleave unmasked vector
$(\tilde{\boldsymbol{c}} - \boldsymbol{l}_0)_0$
$(\tilde{\boldsymbol{c}} - \boldsymbol{l}_0)_{n-1}$
$(\tilde{\boldsymbol{c}} - \boldsymbol{l}_{2^l-1})_0$
$(\tilde{\boldsymbol{c}} - \boldsymbol{l}_{2^l-1})_{n-1}$
Divide by $\boldsymbol{q}_0$
Divide by $\boldsymbol{q}_{n-1}$
$(\boldsymbol{d}_0)_0$
$(\boldsymbol{d}_{2^l-1})_0$
$(\boldsymbol{d}_0)_{n-1}$
$(\boldsymbol{d}_{2^l-1})_{n-1}$
Interleave quotients to generate candidate set of ciphertexts
$\boldsymbol{d}_{2^l-1}$
$\boldsymbol{d}_0$
Viterbi decoder 0
Viterbi decoder $2^l - 1$
Select $\hat{\boldsymbol{d}}$ that minimizes $\hat{\boldsymbol{e}}$, and obtain $\hat{\boldsymbol{m}}_r S$
$\hat{\boldsymbol{m}}_r S$
S^{-1}
$\hat{\boldsymbol{m}}_r$
Divide by $\boldsymbol{r}(x)$
Remainder
CRC verified ?
Yes
No
Accept plaintext
$\boldsymbol{m}$
Discard $\hat{\boldsymbol{d}}$ from candidate set and repeat until a valid $\hat{\boldsymbol{m}}_r$ is found, or reject all

Fig. 1. Encryption and Decryption Block Diagram.

by polynomial division with error propagation. Each of these steps introduces substantial complexity. In essence, an attack from this "bottom-up" perspective must still overcome significant obfuscation, making the system highly resistant to both direct and indirect forms of cryptanalysis.

6 Polynomial Selection

The decryption algorithm involves polynomial divisions that may cause error propagation; a single error in the polynomials $(\tilde{\boldsymbol{c}}(x) - \boldsymbol{l}_i(x))_j$ can lead to multiple errors in the quotient. To reduce this risk, the polynomial matrix $G_P(x)$ is selected to ensure that the corresponding CC has strong error-correction capabilities (e.g., free distance $d_{free} > 20$), outperforming Goppa codes or similar linear block codes of comparable size. This results in rare decoding failures, even at relatively high error rates e (a larger e also implies improved security).

To further limit error propagation, the polynomials $G_Q(x)$ should be chosen to minimize the spread of isolated errors in $(\tilde{\boldsymbol{c}}(x)-\boldsymbol{l}_i(x))_j$, confining them to just a few errors in the quotient. Given an error rate e, simulations can test $G_Q(x)$ by measuring the number of quotient errors for various random error vectors with the same e, allowing for informed selection. Let $\boldsymbol{e}_j$ be a random error vector of length N/n with error rate e. The total number of additional errors introduced by polynomial division, denoted α, is given by:

$$\alpha = \sum_{j=0}^{n-1}[\mathrm{wt}(\frac{\boldsymbol{e}_j}{\boldsymbol{q}_j}) - \mathrm{wt}(\boldsymbol{e}_j)] \tag{32}$$

For the trivial choice $\boldsymbol{q}_j(x) = x^y$ (with $y \leq q$), division by $\boldsymbol{q}_j(x)$ avoids error propagation. A practical way to limit the spread of isolated errors during polynomial division is to select $\boldsymbol{q}_j$ as a sparse polynomial, increasing the gaps between the exponents of its nonzero terms. An effective choice is a two-term polynomial that extends the run-length of each row while minimizing error spread:

$$\boldsymbol{q}_j(x) = 1 + x^q \tag{33}$$

Increasing the number of nonzero elements in $\boldsymbol{q}_j(x)$ can dramatically amplify the difference:

$$\mathrm{wt}(\frac{\boldsymbol{e}_j}{\boldsymbol{q}_j}) - \mathrm{wt}(\boldsymbol{e}_j) \tag{34}$$

Thus, polynomials of $G_Q(x)$ should be chosen to ensure the estimated error rate:

$$\frac{eN+\alpha}{N} \tag{35}$$

remains within the CC's decoding capacity.

7 Worked Example

Define the following CC, $G_P(x) = [\boldsymbol{p}_0(x), \boldsymbol{p}_1(x)] = [1+x^2, 1+x+x^2]$. Suppose that a randomly selected plaintext of length 6 is given by $\boldsymbol{m} = [111001]$, in polynomial form $\boldsymbol{m}(x) = 1+x+x^2+x^5$. In this example we skip the trivial step of CRC construction, assuming it is already contained within $\boldsymbol{m}(x)$. A codeword $\boldsymbol{d}(x)$ of the CC is obtained by either interleaving $\boldsymbol{m}(x)\boldsymbol{p}_0(x)$ with $\boldsymbol{m}(x)\boldsymbol{p}_1(x)$ or simply by employing a scalar generator matrix with 6 rows (corresponding to the length of $\boldsymbol{m}$) constructed according to Equation (5)

$$G_P = \begin{bmatrix} 1\,1\,0\,1\,1\,1\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0 \\ 0\,0\,1\,1\,0\,1\,1\,1\,0\,0\,0\,0\,0\,0\,0\,0 \\ 0\,0\,0\,0\,1\,1\,0\,1\,1\,1\,0\,0\,0\,0\,0\,0 \\ 0\,0\,0\,0\,0\,0\,1\,1\,0\,1\,1\,1\,0\,0\,0\,0 \\ 0\,0\,0\,0\,0\,0\,0\,0\,1\,1\,0\,1\,1\,1\,0\,0 \\ 0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,1\,1\,0\,1\,1\,1 \end{bmatrix}$$

The codeword $\boldsymbol{d}$ in vector form is given by

$$\boldsymbol{d} = \boldsymbol{m}G_P = [111001]G_P = [1110011011110111]$$

Next, choosing the polynomial matrix $G_Q = [1 + x^7, x^7]$, we obtain the corresponding high-memory generator matrix. $G_{PQ}(x) = [\boldsymbol{p}_0\boldsymbol{q}_0, \boldsymbol{p}_1\boldsymbol{q}_1] = [1 + x^2 + x^7 + x^9, x^7 + x^8 + x^9]$. Using $G_{PQ}(x)$, we construct the high-memory scalar generator matrix G_{PQ} as in Equation (9) with: $\boldsymbol{g}_0 = [10], \boldsymbol{g}_1 = [00], \boldsymbol{g}_2 = [10], \boldsymbol{g}_3 = [00], \boldsymbol{g}_4 = [00], \boldsymbol{g}_5 = [00], \boldsymbol{g}_6 = [00], \boldsymbol{g}_7 = [11], \boldsymbol{g}_8 = [01], \boldsymbol{g}_9 = [11], \boldsymbol{0} = [00]$. The resulting matrix is:

$$G_P = \begin{bmatrix} 1&0&0&0&1&0&0&0&0&0&0&0&0&0&1&1&0&1&1&1&0&0&0&0&0&0&0&0&0&0 \\ 0&0&1&0&0&0&1&0&0&0&0&0&0&0&0&0&1&1&0&1&1&1&0&0&0&0&0&0&0&0 \\ 0&0&0&0&1&0&0&0&1&0&0&0&0&0&0&0&0&0&1&1&0&1&1&1&0&0&0&0&0&0 \\ 0&0&0&0&0&0&1&0&0&0&1&0&0&0&0&0&0&0&0&0&1&1&0&1&1&1&0&0&0&0 \\ 0&0&0&0&0&0&0&0&1&0&0&0&1&0&0&0&0&0&0&0&0&0&1&1&0&1&1&1&0&0 \\ 0&0&0&0&0&0&0&0&0&0&1&0&0&0&1&0&0&0&0&0&0&0&0&0&1&1&0&1&1&1 \end{bmatrix}$$

Note that the run-length of each row has increased from 6 to 20. Construct $\tilde{G}$ by randomly choosing its rows from the set $\mathcal{L} = \{(101010\ldots01), (010101\ldots01)\}$ (or any other set of l random vectors of length 30):

$$\tilde{G} = \begin{bmatrix} 1&0&1&0&1&0&1&0&1&0&1&0&1&0&1&0&1&0&1&0&1&0&1&0&1&0&1&0&1&0 \\ 0&1&0&1&0&1&0&1&0&1&0&1&0&1&0&1&0&1&0&1&0&1&0&1&0&1&0&1&0&1 \\ 1&0&1&0&1&0&1&0&1&0&1&0&1&0&1&0&1&0&1&0&1&0&1&0&1&0&1&0&1&0 \\ 0&1&0&1&0&1&0&1&0&1&0&1&0&1&0&1&0&1&0&1&0&1&0&1&0&1&0&1&0&1 \\ 0&1&0&1&0&1&0&1&0&1&0&1&0&1&0&1&0&1&0&1&0&1&0&1&0&1&0&1&0&1 \\ 1&0&1&0&1&0&1&0&1&0&1&0&1&0&1&0&1&0&1&0&1&0&1&0&1&0&1&0&1&0 \end{bmatrix}$$

Now sum $\tilde{G}$ and G_{PQ} to obtain

$$G_{PQ} + \tilde{G} = \begin{bmatrix} 0&0&1&0&0&0&1&0&1&0&1&0&1&0&0&1&1&1&0&1&1&0&1&0&1&0&1&0&1&0 \\ 0&1&1&1&0&1&1&1&0&1&0&1&0&1&0&1&1&0&0&0&1&0&0&1&0&1&0&1&0&1 \\ 1&0&1&0&0&0&1&0&0&0&1&0&1&0&1&0&1&0&0&1&1&1&0&1&1&0&1&0&1&0 \\ 0&1&0&1&0&1&1&1&0&1&1&1&0&1&0&1&0&1&0&1&1&0&0&0&1&0&0&1&0&1 \\ 0&1&0&1&0&1&0&1&1&1&0&1&1&1&0&1&0&1&0&1&0&1&1&0&0&0&1&0&0&1 \\ 1&0&1&0&1&0&1&0&1&0&0&0&1&0&0&0&1&0&1&0&1&0&1&0&0&1&1&1&0&1 \end{bmatrix}$$

which is now a high-density matrix with nearly half of the entries being "1". Multiplying $G_{PQ} + \tilde{G}$ on the left by the following non-singular matrix S:

$$S = \begin{bmatrix} 1&0&0&1&0&0 \\ 0&1&0&0&0&1 \\ 0&0&1&0&0&0 \\ 0&0&1&1&1&0 \\ 0&0&0&0&1&0 \\ 0&0&1&0&1&1 \end{bmatrix}$$

we have

$$S(G_{PQ}+\tilde{G}) = \begin{bmatrix} 0&1&1&1&0&1&0&1&1&1&0&1&1&1&0&0&1&0&0&0&0&0&1&0&0&0&1&1&1&1 \\ 1&1&0&1&1&1&0&1&1&1&0&1&1&1&0&1&0&0&1&0&0&0&1&1&0&0&1&0&0&0 \\ 1&0&1&0&0&0&1&0&0&0&1&0&1&0&1&0&1&0&0&1&1&1&0&1&1&0&1&0&1&0 \\ 1&0&1&0&0&0&0&0&1&0&0&0&0&0&1&0&1&0&0&1&0&0&1&1&0&0&0&1&1&0 \\ 0&1&0&1&0&1&0&1&1&1&0&1&1&1&0&1&0&1&0&1&0&1&1&0&0&0&1&0&0&1 \\ 0&1&0&1&1&1&0&1&0&1&1&1&1&1&1&1&0&1&1&0&0&0&0&1&1&1&1&1&1&0 \end{bmatrix}$$

Apply to $S(G_{PQ}+\tilde{G})$ the following column permutation π, defined by the bijection function

$$\pi : \big(14\ 25\ 9\ 18\ 30\ 8\ 21\ 1\ 10\ 29\ 5\ 26\ 3\ 11\ 23\ 28\ 15\ 2\ 7\ 12\ 20\ 6\ 17\ 4\ 27\ 16\ 24\ 13\ 22\ 19\big)$$
$$\mapsto \big(1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9\ 10\ 11\ 12\ 13\ 14\ 15\ 16\ 17\ 18\ 19\ 20\ 21\ 22\ 23\ 24\ 25\ 26\ 27\ 28\ 29\ 30\big)$$

This permutation can also be performed by multiplying $S(G_{PQ}+\tilde{G})$ on the right by an equivalent 30×30 permutation matrix R

$$S(G_{PQ}+\tilde{G})R = \begin{bmatrix} 1&0&1&0&1&1&0&0&1&1&0&0&1&0&1&1&0&1&0&1&0&1&1&1&1&0&0&1&0&0 \\ 1&0&1&0&0&1&0&1&1&0&1&0&0&0&1&0&0&1&0&1&0&1&0&1&1&1&1&1&0&1 \\ 0&1&0&0&0&0&1&1&0&1&0&0&1&1&0&0&1&0&1&0&1&0&1&0&1&0&1&1&1&0 \\ 0&0&1&0&0&0&0&1&0&1&0&0&1&0&1&1&1&0&0&0&1&0&1&0&0&0&1&0&0&0 \\ 1&0&1&1&1&1&0&0&1&0&0&0&0&0&1&0&0&1&0&1&1&1&0&1&1&1&0&1&1&0 \\ 1&1&0&1&0&1&0&0&1&1&1&1&0&1&0&1&1&1&0&1&0&1&0&1&1&1&1&1&0&1 \end{bmatrix}$$

The codeword $\boldsymbol{c}$ of the MCC corresponding to G is given by

$$\boldsymbol{c} = \boldsymbol{m}G = [1110001]G = [100111101101000001111101001010]$$

If we assume a randomly generated error vector $\boldsymbol{e}$ with three errors:

$$\boldsymbol{e} = [000100000000000010100000000000]$$

then the received vector is:

$$\boldsymbol{c}_e = \boldsymbol{c} + \boldsymbol{e} = [100011101101000011011101001010]$$

<u>**Decryption**</u>

Step1: Inverse Permutation:
Apply the inverse permutation π^{-1} on the bits of $\boldsymbol{c}_e$:

$$\pi^{-1} : \big(2\ 18\ 13\ 24\ 11\ 22\ 19\ 6\ 3\ 9\ 14\ 20\ 28\ 1\ 17\ 26\ 23\ 4\ 30\ 21\ 7\ 29\ 15\ 27\ 2\ 12\ 25\ 16\ 10\ 5\big)$$
$$\mapsto \big(1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9\ 10\ 11\ 12\ 13\ 14\ 15\ 16\ 17\ 18\ 19\ 20\ 21\ 22\ 23\ 24\ 25\ 26\ 27\ 28\ 29\ 30\big)$$

This operation can also be described by multiplying $\boldsymbol{c}_e$ on the right by an equivalent 30×30 permutation matrix R^{-1}:

$$\tilde{\boldsymbol{c}} = \boldsymbol{c}_e R^{-1} = [010101010101011000011101010011]$$

Step2: Unmasking
To unmask $\tilde{\boldsymbol{c}}$, we first need to deinterleave it to its polynomial constitutes, then compute the four possible unmasked variants:

$$(\tilde{c}(x) - \mathbf{0}(x))_0 = x^7 + x^{10} + x^{14}$$
$$(\tilde{c}(x) - \mathbf{1}(x))_0 = 1 + x + x^2 + x^3 + x^4 + x^5 + x^6 + x^8 + x^9 + x^{11} + x^{12} + x^{13}$$
$$(\tilde{c}(x) - \mathbf{0}(x))_1 = 1 + x + x^2 + x^3 + x^4 + x^5 + x^6 + x^9 + x^{10} + x^{11} + x^{12} + x^{14}$$
$$(\tilde{c}(x) - \mathbf{1}(x))_1 = x^7 + x^8 + x^{13}$$

Step 3: Inverting the High-Memory Polynomial Multiplication
Dividing the first two outcomes of Step 2 by $\boldsymbol{q}_0(x)$ and the last two by $\boldsymbol{q}_1(x)$:

$$(\boldsymbol{d}_0(x))_0 = (x^7 + x^{10} + x^{14})/(1 + x^7) = x^3 + x^7 \quad \text{remainder: } x^3$$
$$(\boldsymbol{d}_1(x))_0 = (1 + x + x^2 + x^3 + x^4 + x^5 + x^6 + x^8 + x^9 + x^{11} + x^{12} + x^{13})/(1 + x^7)$$
$$= x + x^2 + x^4 + x^5 + x^6 \quad \text{remainder: } 1 + x^3$$
$$(\boldsymbol{d}_0(x))_1 = (1 + x + x^2 + x^3 + x^4 + x^5 + x^6 + x^9 + x^{10} + x^{11} + x^{12} + x^{14})/x^7$$
$$= x^2 + x^3 + x^4 + x^5 + x^7 \quad \text{remainder: } 1 + x + x^2 + x^3 + x^4 + x^5 + x^6$$
$$(\boldsymbol{d}_1(x))_1 = (x^7 + x^8 + x^{13})/x^7 = 1 + x + x^6 \quad \text{remainder: } 0.$$

Step 4: Quotient Interleaving
The four interleaved values of $\boldsymbol{d}_i$ are given by:

$$\boldsymbol{d}_0 = (\boldsymbol{d}_0)_0 \wedge (\boldsymbol{d}_0)_1 = [0\,0\,0\,0\,0\,1\,1\,1\,0\,1\,0\,1\,0\,0\,1\,1]$$
$$\boldsymbol{d}_1 = (\boldsymbol{d}_0)_0 \wedge (\boldsymbol{d}_1)_1 = [0\,1\,0\,1\,0\,0\,1\,0\,0\,0\,0\,0\,0\,1\,1\,0]$$
$$\boldsymbol{d}_2 = (\boldsymbol{d}_1)_0 \wedge (\boldsymbol{d}_0)_1 = [0\,0\,1\,0\,1\,1\,0\,1\,1\,1\,1\,1\,1\,0\,0\,1]$$
$$\boldsymbol{d}_3 = (\boldsymbol{d}_1)_0 \wedge (\boldsymbol{d}_1)_1 = [0\,1\,1\,1\,1\,0\,0\,0\,1\,0\,1\,0\,1\,1\,0\,0]$$

Step 5: Parallel Viterbi Decoding
Decoding the four interleaved quotients in parallel yields the most likely codeword $\hat{\boldsymbol{d}}$ according to Equation (29), with the corresponding transformed plaintext $\hat{\boldsymbol{m}}_r S$. In this example, the CC has a free distance of $d_{free} = 5$, which implies that the code can reliably correct up to two errors within a sliding window of six bits. If three or more errors occur within this window, the parallel Viterbi decoding step is likely to fail in recovering the correct plaintext. Nevertheless, such decoding failure, can be detected by the CRC. The trellis for the case $\boldsymbol{d}_3 = (\boldsymbol{d}_1)_0 \wedge (\boldsymbol{d}_1)_1$ is depicted in Fig. 2. The highlighted path in the trellis corresponding to the least-weight path has an accumulated distance of 2 from $\boldsymbol{d}_3$, which corresponds to the following information word: 11011000. For all other Viterbi decoders, the most likely path through their respective trellises maintain a minimum distance greater than two from the corresponding $\boldsymbol{d}_i$. Note that for a CC with memory length 2, two zeroes are appended to the end of the information sequence to force the trellis to converge to a single termination state. By discarding these appended zeroes, the most likely transformed plaintext is obtained as:

$$\hat{\boldsymbol{m}} S = 110110$$

Step 6: Plaintext Recovery
Using the inverse matrix S^{-1}:

$$S^{-1} = \begin{bmatrix} 1\,0\,1\,1\,1\,0 \\ 0\,1\,1\,0\,1\,1 \\ 0\,0\,1\,0\,0\,0 \\ 0\,0\,1\,1\,1\,0 \\ 0\,0\,0\,0\,1\,0 \\ 0\,0\,1\,0\,1\,1 \end{bmatrix}$$

we can recover the original plaintext $\boldsymbol{m}$ as follows:

$$\boldsymbol{m} = \hat{\boldsymbol{m}}SS^{-1} = [110110]S^{-1} = [111001]$$

Thus, the recovered plaintext is 111001, confirming correctness. In this example, the recovery of $\boldsymbol{m}$ required only a single iteration. In the general case, the remainder resulting from the division of $\hat{\boldsymbol{m}}_r(x)$ by $\boldsymbol{r}(x)$ must be computed. If this remainder in non-zero, the selected codeword $\hat{\boldsymbol{d}}$ should be discarded. The process must then be repeated iteratively with the next most likely candidate until a valid $\boldsymbol{d}_i$ is identified or all possible candidates have been exhausted.

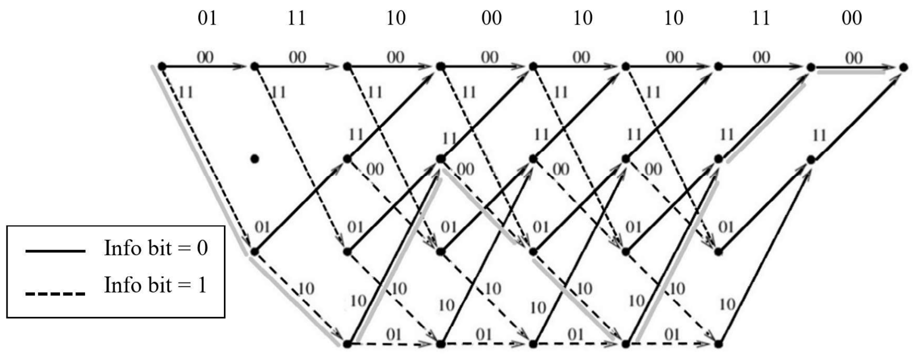

Fig. 2. Trellis and least-weight path corresponding to $\boldsymbol{d}_3 = [0111100010101100]$.

8 Cryptanalysis Resistance and Performance

In code-based cryptography, the complexity of cryptanalysis is primarily determined by the Information-Set Decoding (ISD) algorithm, which targets the problem of decoding random linear block codes in the presence of errors. This task is considered computationally hard for large code dimensions and high error weights. However, if the public code deviates from a random code due to hidden structures, ISD may not reflect the true attack complexity. The underlying mathematical problem is the syndrome decoding problem. In our context, the finite-dimensional matrix G, which serves as the generator matrix for the MCC (allowing it to be treated as a linear block code), has an associated parity-check matrix H satisfying:

$$GH^T = \mathbf{0} \tag{36}$$

Given a syndrome $\boldsymbol{s}$, and assuming that exactly t errors have occurred, the challenge is to find an error vector $\boldsymbol{e}$ such that

$$H\boldsymbol{e}^T = \boldsymbol{s} \quad \text{and} \quad \mathrm{wt}(\boldsymbol{e}) = t \tag{37}$$

The cryptanalysis complexity is estimated using the best-known ISD algorithms (e.g., Prange, Lee-Brickell, BJMM) [6-8], which assess security as bit operations, often ignoring memory usage or parallelization. These estimates assume a fixed weight $\mathrm{wt}(\boldsymbol{e})$. However, in our method, the exact error count is unknown—only the error probability e is known—introducing additional complexity compared to other code-based schemes. While structured error patterns may affect complexity, the MCC's error vector $\boldsymbol{e}$ is truly random and, by definition, unstructured. In the context of quantum computing, ISD complexity is typically estimated based on classical assumptions. While quantum algorithms like Grover's [9] can offer quadratic speedup, no exponential ISD improvement is known. Therefore, ISD-based complexity assessments in code-based cryptography assume ISD remains the most efficient attack. To compare MCC complexity with other code-based methods, we use the measure from [9]. For a classical computer attacking an (N, K, t) code, the ISD complexity, denoted C_{ISD}, is proportional to the number of iterations executed by the ISD algorithm. In our notation, we have

$$C_{ISD} \sim \frac{\binom{N}{K}}{0.29\binom{N-t}{K}} \tag{38}$$

and for quantum computing, the complexity, denoted C_{QISD}, is given by

$$C_{QISD} \sim \sqrt{\frac{\binom{N}{K}}{0.29\binom{N-t}{K}}} \tag{39}$$

where each quantum iteration involves a function evaluation requiring $O(N^3)$ qubit operations.

To compare MCC decoding complexity with the most efficient Goppa-code variant, specifically the $(4096, 3556, 45)$ code with similar key lengths, we observe that MCC offers an additional layer of security. In MCC, only the probability e is known, not the actual number of errors t. Additionally, polynomial division in the decoder introduces extra, unpredictable errors, further obscuring the error count. As a result, while ISD assumes that t is fixed, decoding in MCC requires iterating over all possible values of t. For comparison purposes, however, we assume that in MCC, t is fixed at $eN + \alpha$, where α represents the additional errors introduced by the polynomial division. Substituting the parameters of the Goppa $(4096, 3556, 45)$ code we obtain:

$$C_{ISD}(\text{Goppa}) \approx \frac{\binom{4096}{3556}}{0.29\binom{4096-45}{3556}} \approx 7.06 \times 10^{40} \tag{40}$$

For an MCC with a comparable key length, we can choose parameters $N = 5600$ and $K = 2400$. For instance, by selecting polynomials $\boldsymbol{p}_0(x)$ and $\boldsymbol{p}_1(x)$ of degree 14 and, $\boldsymbol{q}_0(x)$ and $\boldsymbol{q}_1(x)$ of degree 386 (resulting in a rate 1/2 MCC with memory length 400 and 2400 input bits) the corresponding 5600-bit codeword achieves a public key length comparable to Goppa code, approximately 2 megabytes in both cases. For these parameters, selecting $\boldsymbol{q_0}(x) = x^{193}$ and $\boldsymbol{q_1}(x) = 1 + x^{386}$ with $e = 0.02$ yield an error probability at the Viterbi decoder input of $\frac{eN+\alpha}{N} \approx 0.07$. This corresponds to approximately 392 errors on average in a 5600-bit ciphertext. This error rate comfortably falls within the error-correction capability of a rate 1/2 CC with memory length 14, making decoding failure highly unlikely. Under these assumptions, the ISD complexity of the MCC is:

$$C_{ISD}(\text{MCC}) \sim \frac{\binom{5600}{2400}}{0.29\binom{5600-392}{2400}} \approx 3.75 \times 10^{100} \tag{41}$$

This result reflects a security improvement by a factor of approximately $0.53 \times 10^{60} \approx 2^{198}$ compared to the classic McEliece scheme in Equation (40). For quantum ISD (QISD), the improvement is roughly 2^{99}, consistent with the square-root speedup provided by Grover's algorithm. Increasing the public key length further enhances resistance to cryptanalysis, as indicated by Equation (38). Given that the primary role of the MCC cryptosystem is to securely distribute the key (with subsequent reuse), a key length of several megabytes is not considered a significant limitation.

To estimate the probability of a decoding failure (and thus the need for retransmission), consider the following example. Suppose we employ a rate 1/4 CC with memory length 10, defined by $G_P(x) = [2327, 2313, 2671, 3175]$, where the polynomials are expressed in octal form. This code offers strong error correction capabilities, featuring a free distance $d_{free} = 29$. Consequently, the Hamming distance between any two paths through the 1024-state trellis that diverge from the same state and merge after 11 segments (i.e., over 44 bits) is at least 29. This implies that the CC can correct up to 14 errors in a 44-bit window. Next consider the following high-memory sparse polynomials

$$G_Q(x) = [1 + x^{495} + x^{990}, x^{247}, x^{743}, 1 + x^{990}]$$

For an error rate $e = 0.04$, simulations yield

$$\frac{\alpha}{N} \approx \frac{0.18 + 0 + 0 + 0.13}{4} \approx 0.0775$$

Thus

$$e + \frac{\alpha}{N} = 0.04 + 0.0775 = 0.1175$$

The Viterbi decoder fails if more than 14 errors occur in a 44-bit window. At an effective error rate of 0.1175, the probability of such an event is approximately 8.998×10^{-5}. This extremely low failure probability ensures reliable decryption for large ciphertexts. For example, over 500 consecutive windows, the probability of successful decryption is $(1-8.998\times10^{-5})^{500} \approx 0.956$. This result demonstrates that the MCC cryptosystem maintains a high probability of successful decryption, even for ciphertexts spanning tens of thousands of bits.

Finally, we discuss the computational complexity of decryption. The dominant factor in the decryption complexity of the MCC cryptosystem arises from the Viterbi decoding step. While other operations—such as polynomial divisions, unmasking, interleaving, and deinterleaving—are performed over the entire received ciphertext, their computational impact is negligible compared to the per-bit complexity of the Viterbi algorithm. To quantify this complexity, we evaluate the number of Add-Compare-Select (ACS) modules employed per decrypted plaintext bit. These ACS modules are efficiently implemented in both software and hardware, making them suitable for practical deployment. Suppose the MCC employs a CC with memory p and a set $\mathcal{L}$ consisting of l random masking vectors. In this case the decryption process requires 2^l parallel Viterbi decoders, each utilizing 2^p ACS modules per bit. Consequently, the total number of ACS modules per plaintext bit is 2^{l+p}. This complexity measure scales linearly with the plaintext length K ensuring predictable and manageable performance as the data size grows. For example, using the rate 1/4 CC with memory $p = 10$ and $l = 5$, the total ACS operations per bit become $2^{l+p} = 2^{15} = 32,768$, a computational load well within the capabilities of commercial processors, including those found in modern mobile devices.

9 Conclusion

This work presents a novel post-quantum encryption scheme based on high-memory masked convolutional codes, addressing the limitations of traditional block code-based methods. Security is reinforced through semi-invertible transformations that generate fully dense, random-like matrices, mitigating vulnerabilities related to low-weight or structured generator matrices. The scheme further strengthens security by incorporating two layers of error injection: deliberate random error insertion at high error rates and additional random errors inherently introduced by polynomial division. The dual-layered error strategy significantly complicates cryptanalysis, even in scenarios where an adversary has complete knowledge of the convolutional code. The interplay between masking and polynomial division ensures that decrypting the ciphertext remains computationally formidable. Beyond its security enhancements, the scheme offers improved scalability, flexibility, and efficient hardware implementation via the Viterbi algorithm. These characteristics make the proposed method a strong candidate for both classical and post-quantum cryptographic applications.

References

1. McEliece, R.J.: A Public-Key Cryptosystem Based on Algebraic Coding Theory. DSN Progress Report (1978)
2. Bernstein, D.J., Lange, T., Peters, C.: Attacking and defending the McEliece cryptosystem. In: Post-Quantum Cryptography, pp. 31–46 (2008)
3. Misoczki, R., Tillich, J.-P., Sendrier, N., Barreto, P.S.L.M.: MDPC-McEliece: New McEliece variants from Moderate Density Parity-Check codes. IEEE Transactions on Information Theory (2013)
4. NIST Post-Quantum Cryptography Standardization (2023). `https://csrc.nist.gov`
5. Baldi, M., Chiaraluce, F., Garello, R., Mininni, F.: Quasi-cyclic low-density parity-check codes in the McEliece cryptosystem. In: IEEE International Conference on Communications (ICC), Glasgow, Scotland, pp. 951–956. IEEE (2007). `https://doi.org/10.1109/ICC.2007.161`
6. Becker, A., Joux, A., May, A., Meurer, A.: Decoding random binary linear codes in $2^{n/20}$. In: Advances in Cryptology - EUROCRYPT (2012)
7. Prange, E.: The use of information sets in decoding cyclic codes. IRE Transactions on Information Theory, 8(5), 5–9 (1962)
8. Lee, P.J., Brickell, E.F.: An observation on the security of McEliece's public-key cryptosystem. In: Advances in Cryptology - EUROCRYPT (1988)
9. Bernstein, D.J.: Grover vs. McEliece. In: Sendrier, N. (ed.) Post-Quantum Cryptography, PQCrypto 2010. Lecture Notes in Computer Science, vol. 6061, pp. 73–80. Springer, Heidelberg (2010). `https://cr.yp.to/papers.html#grovercode`