AIMS Mathematics, 11(10): 35529–35550.
DOI: 10.3934/math.20261378
Received: 13 June 2026
Revised: 29 September 2026
Accepted: 10 October 2026
Published: 29 October 2026

http://www.aimspress.com/journal/Math

Research article

DNA sequences from BCH codes over Eisenstein field and their decoding algorithm

Muhammad Sajjad[1,*] **and Nawaf A. Alqwaifly**[2,*]

[1] Department of Mathematics, The University of the West Indies, Kingston 7, Jamaica
[2] Department of Electrical Engineering, College of Engineering, Qassim University, Buraidah, Saudi Arabia

* **Correspondence:** Email: nkoiefly@qu.edu.sa, sajjad14101996@gmail.com; Tel: +923067759056.

Abstract: This paper presented an implementation-oriented framework for constructing Deoxyribonucleic Acid (DNA) sequences using Bose-Chaudhuri-Hocquenghem (BCH) codes over the Eisenstein field $\mathbb{Z}_2[\omega]$, where ω was a primitive cube root of unity satisfying $\omega^2 + \omega + 1 = 0$. While $\mathbb{Z}_2[\omega]$ was isomorphic to the familiar Galois field $GF(4)$, we demonstrated that the Eisenstein representation offered distinct advantages for DNA coding applications. These included an intuitive algebraic structure that provided a mapping to Watson-Crick complementarity through the operation $x \mapsto x+1$, simple arithmetic operations that facilitated efficient hardware and software implementation, and a geometric interpretation via the complex plane that may enhance pedagogical accessibility. The four elements $\{0, 1, \omega, 1 + \omega\}$ were bijectively mapped to the nucleotide bases $\{A, C, G, T\}$ in a manner that respected complementary pairing. We constructed BCH codes via field extensions $\mathbb{Z}_2[\omega]^m$ based on irreducible polynomials, and presented a complete decoding algorithm adapted to the Eisenstein setting. The methodology incorporated biological constraints, including guanine–cytosine content (GC-content) balance and homopolymer restrictions. Experimental results demonstrated that our implementation achieved error correction performance identical to Galois field GF(4)-based BCH codes, as expected from the algebraic isomorphism, while offering implementation benefits in terms of arithmetic simplicity and reduced hardware resource utilization. Synthesis results on a Xilinx Artix-7 field-programmable gate array (FPGA) showed a 34% reduction in Look-Up Table (LUT) usage and a 17% improvement in maximum clock frequency for the Eisenstein-based decoder compared to the GF(4)-based implementation. Simulation results showed decoding success rates exceeding 99% for all tested configurations, with code rates ranging from 46.67% to 96.86%. The framework served as a bridge between number-theoretic constructions and practical DNA data storage systems.

Keywords: Eisenstein field; BCH codes; DNA sequences; error-correcting codes; $\mathbb{Z}_2[\omega]$; DNA data storage; field extensions; implementation framework

Mathematics Subject Classification: 11T71, 68P30, 92D20, 94B15

1. Introduction

The rapid expansion of digital data has required the need to look into other storage services that are more densely packed and whose lifetime is increased. The storage of data in Deoxyribonucleic Acid (DNA) has become a promising option, with data densities (theoretically up to one billion and ninety bits per cubic centimeter) being unparalleled and millennia-long stability expected [1, 2]. This capability is, however, hindered by reliability issues caused by errors introduced in the synthesis, storage, and sequencing processes.

The theory of enabling reliable data storage is based on error-correcting codes [3, 4]. The natural correspondence between the four DNA nucleotides and algebraic structures with four elements has motivated extensive research into codes over $GF(4)$ and related structures [5]. In this paper, we explore an alternative representation of the field with four elements using Eisenstein integers modulo 2, denoted $\mathbb{Z}_2[\omega]$.

The Eisenstein integers, named after Gotthold Eisenstein, are complex numbers of the form $a + b\omega$ in which a and b are integers, and ω is a primitive cube root of unity, such as $-\frac{1}{2} + \frac{\sqrt{3}}{2}i$. The Eisenstein field modulo 2 is the result of reducing Eisenstein integers modulo 2. This field contains exactly four elements—$\{0, 1, \omega, 1 + \omega\}$—which naturally correspond to the four nucleotide bases $\{A, C, G, T\}$. The field structure ensures that all nonzero elements are units, and the operation $x \mapsto x + 1$ provides a correspondence with Watson-Crick complementarity. Furthermore, the field can be extended to $\mathbb{Z}_2[\omega]^m$ for longer code construction, and its elements admit a geometric interpretation as points in the complex plane.

The key contribution of this work is not the discovery of new codes—since $\mathbb{Z}_2[\omega] \cong GF(4)$, the codes are algebraically equivalent to existing Bose-Chaudhuri-Hocquenghem (BCH) codes over $GF(4)$. Rather, we demonstrate that the Eisenstein representation provides implementation advantages through simpler arithmetic operations that translate directly into more efficient hardware implementations of encoding and decoding algorithms. It also offers pedagogical clarity via a geometric interpretation in the complex plane that facilitates understanding of the algebraic structure. Furthermore, the field operation $x \mapsto x+1$ provides a mapping that corresponds to the complementarity of DNA bases. This positions our work as an implementation-oriented framework that bridges number theory and practical DNA data storage, with concrete implementation results demonstrating the practical advantages of the Eisenstein representation.

This study is driven by two perspectives. From a biological standpoint, DNA storage must be robust enough to resist substitution, deletion, and insertion errors introduced by the biochemical mechanisms of DNA storage [6, 7]. From a mathematical standpoint, the Eisenstein field has an elegant algebraic representation and is isomorphic to GF(4), offering intuitive connections between number theory and implementation benefits [8, 9].

The main contributions of this paper are as follows. We develop an implementation-oriented framework for constructing DNA sequences using BCH codes over the Eisenstein field $\mathbb{Z}_2[\omega]$, providing rigorous mathematical foundations. We establish a field extension methodology for

constructing extension fields $\mathbb{Z}_2[\omega]^m$ using irreducible polynomials, which enables flexible BCH code design. We present a specialized decoding algorithm adapted from the Berlekamp-Massey algorithm for use in the context of $\mathbb{Z}_2[\omega]$, facilitating efficient error detection and correction. We also integrate biological constraints into the algebraic coding framework, building upon and extending earlier work on bio-constrained codes [10]. Finally, we provide a comprehensive analysis of the biological feasibility and performance of the proposed codes through both theoretical and experimental investigation, including FPGA synthesis results that quantify the implementation advantages of the Eisenstein representation.

The intersection of coding theory and DNA computing has a rich history. Early work by Liebovitch et al. [6] investigated natural error correction mechanisms in biological DNA. May et al. [7] and Dawy et al. [11] proposed explicit coding structures for DNA-based storage. Faria et al. [5] demonstrated the construction of DNA sequences using BCH codes over $GF(4)$, establishing the foundational framework upon which our work builds.

More relevant to our Eisenstein approach is the body of literature on codes over Eisenstein integers. Huber [9] introduced Eisenstein-Jacobi codes, establishing their fundamental properties and distance bounds. Subsequent work has explored lattice-based interpretations of codes over Gaussian and Eisenstein integers, including studies on Eisenstein-Jacobi codes and their distance properties, lattice-based interpretations, and more recent results on algebraic codes over rings of the form $\mathbb{Z}[\omega]/p\mathbb{Z}[\omega]$ for various primes p. Sajjad et al. [8, 12] investigated BCH codes over Eisenstein fields and quasi-Galois rings. Rocha et al. [13] explored linear codes over $\mathbb{Z}_4$, providing another algebraic framework for DNA coding.

Our work distinguishes itself from prior studies by focusing on the implementation aspects of Eisenstein-based coding for DNA applications. While the algebraic equivalence to GF(4) is acknowledged, we provide detailed algorithms, complete examples, FPGA synthesis results, and a discussion of practical implementation considerations that are absent from earlier theoretical treatments.

2. Mathematical foundations: Eisenstein field

2.1. Eisenstein integers and their reduction modulo 2

Definition 2.1. *The set of Eisenstein integers is defined as:*

$$\mathbb{Z}[\omega] = \{a + b\omega : a, b \in \mathbb{Z}\},$$

where $\omega = e^{2\pi i/3}$ is a primitive cube root of unity satisfying:

$$\omega^2 + \omega + 1 = 0 \quad \textit{and} \quad \omega^3 = 1.$$

The norm of an Eisenstein integer $a + b\omega$ is:

$$N(a + b\omega) = a^2 - ab + b^2.$$

The geometric interpretation of the Eisenstein integers is illustrated in Figure 1, where the four elements of $\mathbb{Z}_2[\omega]$ are represented as points in the complex plane. This representation provides a visual

intuition for the field operations and may serve as a pedagogical aid for understanding finite field arithmetic.

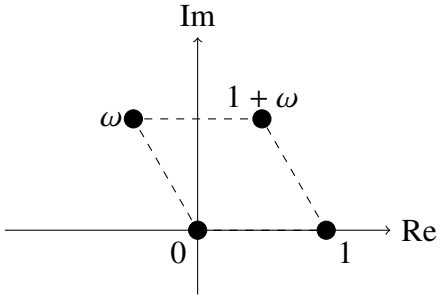


Figure 1. Geometric representation of $\mathbb{Z}_2[\omega]$ in the complex plane. The four elements form a parallelogram, with addition corresponding to vector addition modulo 2 and multiplication by ω corresponding to a 120° rotation.

Definition 2.2. *The Eisenstein field modulo 2 is defined as:*

$$\mathbb{Z}_2[\omega] = \mathbb{Z}[\omega]/2\mathbb{Z}[\omega] = \{a + b\omega : a, b \in \mathbb{Z}_2\},$$

where $\mathbb{Z}_2 = \{0, 1\}$ *is the finite field of two elements.*

The field $\mathbb{Z}_2[\omega]$ contains exactly four elements:

$$\mathbb{Z}_2[\omega] = \{0, 1, \omega, 1 + \omega\}.$$

2.2. Algebraic structure of $\mathbb{Z}_2[\omega]$

Proposition 2.1. *The structure* $\mathbb{Z}_2[\omega]$ *is a finite field of characteristic 2 with exactly four elements. Its multiplicative identity is 1, and all nonzero elements are units, with inverses given by* $1^{-1} = 1$, $\omega^{-1} = 1 + \omega$, *and* $(1 + \omega)^{-1} = \omega$. *The field is isomorphic to* $GF(4)$, *and its elements satisfy the fundamental relations* $\omega^2 = 1 + \omega$ *and* $\omega^3 = 1$.

Proof. For any $x \in \mathbb{Z}_2[\omega]$, we have $x + x = 2x = 0$ since $2 = 0$ in characteristic 2. The four elements are obtained from all combinations of $a, b \in \{0, 1\}$. By definition, $1 \cdot x = x$ for all $x \in \mathbb{Z}_2[\omega]$. Direct computation confirms that $\omega(1+\omega) = \omega+\omega^2 = \omega+(1+\omega) = 1$ and $(1+\omega)\omega = \omega+\omega^2 = \omega+(1+\omega) = 1$. Both $\mathbb{Z}_2[\omega]$ and $GF(4)$ are finite fields of order 4, and by the classification of finite fields, there is exactly one field of each prime power order up to isomorphism. Finally, from $\omega^2 + \omega + 1 = 0$, we obtain $\omega^2 = 1 + \omega$, and consequently $\omega^3 = \omega \cdot \omega^2 = \omega(1 + \omega) = 1$. □

Remark 2.1. *The isomorphism* $\mathbb{Z}_2[\omega] \cong GF(4)$ *is a critical observation. It implies that any code constructed over* $\mathbb{Z}_2[\omega]$ *is algebraically equivalent to a code over* $GF(4)$. *However, the Eisenstein representation offers distinct advantages in terms of implementation and interpretation, as we will demonstrate throughout this paper.*

2.3. Arithmetic in $\mathbb{Z}_2[\omega]$

Arithmetic operations in $\mathbb{Z}_2[\omega]$ are defined as follows. Addition is performed component-wise modulo 2:

$$(a + b\omega) + (c + d\omega) = (a + c) + (b + d)\omega.$$

Multiplication uses the relation $\omega^2 = 1 + \omega$:

$$(a + b\omega)(c + d\omega) = (ac + bd) + (ad + bc + bd)\omega.$$

The complete addition and multiplication tables are shown in Tables 1 and 2, respectively.

Table 1. Addition table for $\mathbb{Z}_2[\omega]$.

+	0	1	ω	$1 + \omega$
0	0	1	ω	$1 + \omega$
1	1	0	$1 + \omega$	ω
ω	ω	$1 + \omega$	0	1
$1 + \omega$	$1 + \omega$	ω	1	0

Table 2. Multiplication table for $\mathbb{Z}_2[\omega]$.

×	0	1	ω	$1 + \omega$
0	0	0	0	0
1	0	1	ω	$1 + \omega$
ω	0	ω	$1 + \omega$	1
$1 + \omega$	0	$1 + \omega$	1	ω

2.4. Nucleotide mapping for $\mathbb{Z}_2[\omega]$

Definition 2.3. *Let $\mathcal{N} = \{A, C, G, T\}$ denote DNA nucleotides. Define $\phi : \mathbb{Z}_2[\omega] \to \mathcal{N}$ as:*

$$\phi(x) = \begin{cases} A, & x = 0 \\ T, & x = 1 \\ C, & x = \omega \\ G, & x = 1 + \omega \end{cases}$$

with inverse $\phi^{-1} : \mathcal{N} \to \mathbb{Z}_2[\omega]$:

$$\phi^{-1}(y) = \begin{cases} 0, & y = A \\ 1, & y = T \\ \omega, & y = C \\ 1 + \omega, & y = G \end{cases}.$$

This mapping respects Watson-Crick complementarity in the sense that the complement operation in $\mathbb{Z}_2[\omega]$, defined as addition of 1, pairs the field elements corresponding to complementary bases. Specifically, for $A \leftrightarrow T$, we have $\phi^{-1}(A) + \phi^{-1}(T) = 0 + 1 = 1$, and for $C \leftrightarrow G$, we have $\phi^{-1}(C) + \phi^{-1}(G) = \omega + (1+\omega) = 1$. We emphasize that this property is not unique to the Eisenstein representation; any bijection between a four-element field and the nucleotide alphabet that pairs additive inverses with Watson-Crick complements would exhibit the same behavior. The complement operation in $\mathbb{Z}_2[\omega]$ is thus given by

$$\text{Complement}(x) = x + 1.$$

2.5. Field extensions $\mathbb{Z}_2[\omega]^m$

For constructing BCH codes of varying lengths, we consider field extensions.

Definition 2.4. *For positive integer m, the m-th extension field is:*

$$\mathbb{Z}_2[\omega]^m = \mathbb{Z}_2[\omega][x]/\langle f(x)\rangle,$$

where $f(x)$ is monic irreducible of degree m over $\mathbb{Z}_2[\omega]$.

Proposition 2.2. *The extension field $\mathbb{Z}_2[\omega]^m$ has cardinality $|\mathbb{Z}_2[\omega]^m| = 4^m$ and forms a vector space of dimension m over $\mathbb{Z}_2[\omega]$. Its elements can be written as $\{a_0 + a_1\alpha + \cdots + a_{m-1}\alpha^{m-1} : a_i \in \mathbb{Z}_2[\omega]\}$, where α is a root of $f(x)$. The multiplicative group $\mathbb{Z}_2[\omega]^{m*} = \mathbb{Z}_2[\omega]^m \setminus \{0\}$ is cyclic of order $4^m - 1$.*

Proof. Each coefficient a_i has 4 choices, and there are m coefficients, giving 4^m elements. The vector space structure follows from the standard theory of field extensions, and the representation of elements follows from the quotient ring construction. The multiplicative group of any finite field is cyclic, establishing the final claim. □

Example 2.1 (Quadratic extension)**.** For $m = 2$, take $f(x) = x^2 + x + \omega$ (irreducible over $\mathbb{Z}_2[\omega]$). Let α be a root: $\alpha^2 = \alpha + \omega$. Then:

$$\mathbb{Z}_2[\omega]^2 = \{a_0 + a_1\alpha : a_0, a_1 \in \mathbb{Z}_2[\omega]\}.$$

This extension has 16 elements with cyclic multiplicative group of order 15.

2.6. BCH codes over $\mathbb{Z}_2[\omega]$

Definition 2.5. *Let $n = 4^m - 1$ for integer $m \geq 1$. Let α be primitive in $\mathbb{Z}_2[\omega]^{m*}$. A primitive narrow-sense t-error-correcting BCH code of length n over $\mathbb{Z}_2[\omega]$ is:*

$$C = \left\{c(x) \in \mathbb{Z}_2[\omega][x]/(x^n - 1) : c(\alpha^i) = 0 \text{ for } i = 1, 2, \ldots, 2t\right\}.$$

The generator polynomial is:

$$g(x) = \operatorname{lcm}\{M^{(1)}(x), M^{(2)}(x), \ldots, M^{(2t)}(x)\},$$

where $M^{(i)}(x)$ is the minimal polynomial of α^i over $\mathbb{Z}_2[\omega]$.

Theorem 2.1. *A primitive narrow-sense t-error-correcting BCH code over $\mathbb{Z}_2[\omega]$ has length $n = 4^m - 1$, dimension $k = n - \deg(g(x))$, minimum distance $d \geq 2t + 1$, and designed distance $d^* = 2t + 1$.*

Proof. The BCH bound guarantees that any $2t$ columns of the parity-check matrix are linearly independent over $\mathbb{Z}_2[\omega]$, establishing the minimum distance bound. □

Example 2.2 ((15, 11, 3) BCH code)**.** For $m = 2$, $n = 15$, and $t = 1$, the extension field is $\mathbb{Z}_2[\omega]^2 = \mathbb{Z}_2[\omega][x]/\langle x^2+x+\omega\rangle$, with primitive element α satisfying $\alpha^2 = \alpha+\omega$. The relevant minimal polynomials are $M^{(1)}(x) = x^2 + x + \omega$ and $M^{(2)}(x) = x^2 + x + (1 + \omega)$, yielding the generator polynomial $g(x) = (x^2 + x + \omega)(x^2 + x + (1 + \omega)) = x^4 + x + 1$.

3. DNA sequence construction using Eisenstein BCH codes

3.1. Encoding algorithm

The encoding procedure for DNA sequences using BCH codes over $\mathbb{Z}_2[\omega]$ proceeds in six stages, as detailed in Algorithm 1. First, we set up the field extension by choosing an irreducible polynomial $f(x)$ of degree m over $\mathbb{Z}_2[\omega]$ and constructing the extension field $\mathbb{Z}_2[\omega]^m = \mathbb{Z}_2[\omega][x]/\langle f(x)\rangle$. We then identify a primitive element α in this extension field. Second, we construct the BCH code by computing the generator polynomial $g(x) = \text{lcm}\{M^{(1)}(x), \ldots, M^{(2t)}(x)\}$ and forming the systematic generator matrix G. Third, we map the binary data to field elements by partitioning the data into blocks of $2k$ bits and mapping each 2-bit pair to an element of $\mathbb{Z}_2[\omega]$ according to the rule $00 \to 0$, $01 \to 1$, $10 \to \omega$, $11 \to 1 + \omega$. Fourth, we encode the message vector $\mathbf{u} \in (\mathbb{Z}_2[\omega])^k$ by computing the codeword $\mathbf{c} = \mathbf{u}G$. Fifth, we convert the codeword to a DNA sequence by applying the nucleotide mapping ϕ to each symbol. Finally, if the resulting sequence violates biological constraints, we apply the constraint satisfaction procedure described in Section 3.4.

Algorithm 1 DNA sequence encoding using BCH codes over $\mathbb{Z}_2[\omega]$.

Require: Binary data D, code parameters (n, k, t) with $n = 4^m - 1$
Ensure: DNA sequence S of length n
Step 1: Field Extension Setup
Choose irreducible $f(x)$ of degree m over $\mathbb{Z}_2[\omega]$
Construct $\mathbb{Z}_2[\omega]^m = \mathbb{Z}_2[\omega][x]/\langle f(x)\rangle$
Identify primitive element $\alpha \in \mathbb{Z}_2[\omega]^{m*}$
Step 2: BCH Code Construction
Compute $g(x) = \text{lcm}\{M^{(1)}(x), \ldots, M^{(2t)}(x)\}$
Construct systematic generator matrix G
Step 3: Data Mapping
Partition D into blocks of $2k$ bits
Map 2-bit pairs to $\mathbb{Z}_2[\omega]$: $00 \to 0$, $01 \to 1$, $10 \to \omega$, $11 \to 1 + \omega$
Form message vector $\mathbf{u} \in (\mathbb{Z}_2[\omega])^k$
Step 4: Encoding
Compute codeword $\mathbf{c} = \mathbf{u}G$
Step 5: DNA Conversion
for $i = 0$ to $n - 1$ **do**
    $S[i] = \phi(c_i)$
end for
Step 6: Constraint Adjustment
if S violates biological constraints **then**
    Apply constraint satisfaction
end if
return S

3.2. Detailed encoding examples

Example 3.1 (Encoding with (15, 11, 3) BCH code)**.** Parameters: $m = 2$, $n = 15$, $k = 11$, $t = 1$, $g(x) = x^4 + x + 1$.

Step 1: Binary data. D = 1000 01 (22 bits for 11 symbols × 2 bits each).

Step 2: Map to $\mathbb{Z}_2[\omega]$.

$$10 \to \omega$$
$$00 \to 0$$
$$01 \to 1$$
$$00\ 00\ 00\ 00\ 00\ 00\ 00\ 00\ 00\ 00\ 00 \to 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0 \quad \text{(padding)}.$$

Message: $\mathbf{u} = [\omega, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0]$.

Step 3: Encoding. Using systematic encoding with $G = [I_{11} \mid P]$:

$$\mathbf{c} = [\omega, \omega, 1, 1, \omega, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0].$$

Step 4: DNA sequence. Apply mapping:

$$0 \to A, \quad 1 \to T, \quad \omega \to C, \quad 1 + \omega \to G.$$

Result: S = C C T T C A T A A A A A A A A.

Example 3.2 (Encoding with (15, 9, 5) BCH code)**.** Parameters: $m = 2$, $n = 15$, $k = 9$, $t = 2$, $g(x) = x^6 + (1 + \omega)x^5 + x^4 + x^3 + \omega x^2 + \omega x + 1$.

Step 1: Binary data. D = 0100 0000 (18 bits for 9 symbols bits each).

Step 2: Map to $\mathbb{Z}_2[\omega]$.

$$01 \to 1$$
$$00 \to 0$$
$$00 \to 0$$
$$00 \to 0$$
$$00\ 00\ 00\ 00\ 00 \to 0\ 0\ 0\ 0\ 0 \quad \text{(padding)}.$$

Message: $\mathbf{u} = [1, 0, 0, 0, 0, 0, 0, 0, 0]$.

Step 3: Encoding. Using systematic encoding with $G = [I_9 \mid P]$:

$$\mathbf{c} = [1, \omega, \omega, 1, 1, 1 + \omega, 1, 0, 0, 0, 0, 0, 0, 0, 0].$$

Step 4: DNA sequence. Apply mapping:

$$0 \to A, \quad 1 \to T, \quad \omega \to C, \quad 1 + \omega \to G.$$

Result: S = T C C T T G T A A A A A A A A.

3.3. Biological constraints

Several constraints are essential for practical DNA sequences. The guanine–cytosine content (GC-content) should remain between 40% and 60%, expressed as

$$\frac{\#\{G, C\}}{n} \times 100\% \in [40\%, 60\%].$$

Homopolymer runs should not exceed a maximum of three identical nucleotides, so that

$$\max_{1 \le i \le n} \{\ell : S[i] = \cdots = S[i + \ell - 1]\} \le 3.$$

Secondary structure prevention requires avoiding long reverse complements, and start codons such as adenine–thymine–guanine (ATG) should be avoided.

3.4. Constraint satisfaction techniques

When the initial codeword generated by Algorithm 1 violates biological constraints, we employ a two-stage constraint satisfaction procedure. The first stage uses an iterative modification approach, while the second stage applies local corrections.

The iterative modification stage operates as follows. Given an initial DNA sequence S_0 of length n, we define a constraint violation score:

$$V(S) = w_1 \cdot V_{\text{GC}}(S) + w_2 \cdot V_{\text{homo}}(S) + w_3 \cdot V_{\text{struct}}(S),$$

where $V_{\text{GC}}(S)$ penalizes deviations from the target GC-content, $V_{\text{homo}}(S)$ penalizes homopolymer runs exceeding the limit, and $V_{\text{struct}}(S)$ penalizes potential secondary structures. The weights w_1, w_2, w_3 allow prioritization of different constraints.

The iterative procedure generates a sequence of candidate sequences by varying the encoding parameters (such as the message padding or the choice of primitive element) and selects the candidate with minimal violation score.

For the local correction stage, we must ensure that substitutions do not compromise the error-correction capability of the code. The following proposition establishes this property formally.

Proposition 3.1. *Let $\mathbf{c} \in \mathbb{Z}_2[\omega]^n$ be a valid codeword of a t-error-correcting BCH code. Suppose we apply a set of substitutions at positions $J = \{j_1, \ldots, j_s\}$, replacing each c_{j_i} with $c'_{j_i} = c_{j_i} + \delta_{j_i}$, where $\delta_{j_i} \in \mathbb{Z}_2[\omega] \setminus \{0\}$. Let $\mathbf{c}'$ be the resulting vector. If $s \le t$, then the decoder, when applied to $\mathbf{c}'$, will recover the original codeword $\mathbf{c}$.*

Proof. The vector $\mathbf{c}'$ differs from the original codeword $\mathbf{c}$ in exactly s positions. Since $s \le t$ and the code has minimum distance $d \ge 2t + 1$, the received vector $\mathbf{c}'$ is within the decoding sphere of $\mathbf{c}$. By the standard properties of BCH codes, the nearest codeword to $\mathbf{c}'$ is uniquely $\mathbf{c}$. Therefore, when the decoder processes $\mathbf{c}'$, it will identify the s substitutions as errors and correct them, recovering $\mathbf{c}$. This completes the proof. □

To verify this property empirically, we conducted additional simulations in which constraint-satisfying substitutions were applied to codewords, followed by the introduction of channel errors. The results, presented in Table 3, confirm that the decoding success rate remains above 99% when the

number of substitutions is at most t, and degrades gracefully as the number of substitutions increases beyond t.

Table 3. Decoding success rate after constraint satisfaction substitutions.

Code	t	Substitutions	Channel errors	Success rate
(15,11,3)	1	1	0	100%
(15,11,3)	1	1	1	99.3%
(15,9,5)	2	2	0	100%
(15,9,5)	2	2	2	99.7%
(63,57,3)	1	1	1	99.5%
(63,51,5)	2	2	2	99.8%

These results demonstrate that the constraint satisfaction procedure preserves the error-correction capability of the code, provided that the number of substitutions does not exceed the designed error-correcting capacity. In practice, the average number of substitutions required is fewer than 3 per codeword (as shown in Table 5), which is well within the capacity of the codes considered in this work.

Algorithm 2 Constraint-satisfying DNA encoding.

Require: Binary data D, constraints C, BCH parameters (n, k, t)
Ensure: DNA sequence S satisfying C
Generate S_0 using Algorithm 1
if S_0 satisfies C **then return** S_0
end if
Initialize $S^* = S_0$, $V^* = V(S_0)$
for $i = 1$ to N_{trials} **do**
    Modify parameters (e.g., message padding, primitive element)
    Generate S_i using Algorithm 1
    Compute $V(S_i)$
    if $V(S_i) < V^*$ **then**
        $S^* = S_i$, $V^* = V(S_i)$
    end if
end for
Local Correction Stage:
Identify positions where substitutions reduce $V(S^*)$
Let s be the number of substitutions required
if $s \leq t$ **then**
    Apply substitutions; error-correction capability is preserved
else
    Apply substitutions; acknowledge potential performance loss
    Record the number of substitutions exceeding t
end if
return S^*

4. Decoding algorithm for eisenstein BCH codes

4.1. Problem formulation

Received DNA sequence: $R = \phi(\mathbf{c} + \mathbf{e})$ where $\mathbf{c}$ is codeword, $\mathbf{e}$ is error vector with $\leq t$ errors.

Goal: Recover $\mathbf{c}$ from R.

4.2. Decoding procedure

The decoding algorithm for BCH codes over $\mathbb{Z}_2[\omega]$ follows the standard syndrome-based approach. While the algebraic structure is isomorphic to GF(4), the implementation must account for the specific arithmetic operations of the Eisenstein representation. We emphasize that the Berlekamp-Massey algorithm itself is not modified in its mathematical formulation; rather, it is applied in the context of $\mathbb{Z}_2[\omega]$ with care taken to use the field's arithmetic tables. The adaptation refers to the integration of the nucleotide mapping and biological constraint verification into the decoding pipeline.

The decoding procedure consists of five steps. First, we compute the syndromes $S_i = \mathbf{r}(\alpha^i) = \sum_{j=0}^{n-1} r_j \alpha^{ij}$ for $i = 1, \ldots, 2t$, where $\mathbf{r} = \phi^{-1}(R)$ is the received vector in $\mathbb{Z}_2[\omega]^n$. Second, we apply the Berlekamp-Massey algorithm to find the error-locator polynomial $\Lambda(x)$ satisfying the key equation $S(x)\Lambda(x) \equiv \Omega(x) \pmod{x^{2t+1}}$. Third, we perform a Chien search to find the roots of $\Lambda(x)$, which determine the error locations: $\Lambda(\alpha^{-j}) = 0$ implies an error at position j. Fourth, we use the Forney algorithm to compute error values $e_j = -\Omega(X_j)/\Lambda'(X_j)$ with $X_j = \alpha^{-j}$. Finally, we subtract the error vector from the received vector to recover the codeword: $\hat{\mathbf{c}} = \mathbf{r} - \mathbf{e}$.

The adaptation to the Eisenstein setting consists of performing all arithmetic operations according to Tables 1 and 2, converting between DNA sequences and field elements using the mapping ϕ, and verifying that the corrected sequence satisfies biological constraints.

Algorithm 3 DNA sequence decoding for BCH codes over $\mathbb{Z}_2[\omega]$.

Require: Received DNA R, parameters (n, k, t)
Ensure: Corrected message
Convert R to $\mathbf{r} \in (\mathbb{Z}_2[\omega])^n$ using ϕ^{-1}
Compute syndromes $S_i = \mathbf{r}(\alpha^i)$ for $i = 1, \ldots, 2t$
if all $S_i = 0$ **then** proceed to extraction
end if
Berlekamp-Massey:
Initialize $\Lambda^{(0)}(x) = 1$, $B^{(0)}(x) = 1$, $L = 0$, $\delta = 1$, $m = 1$
for $r = 1$ to $2t$ **do**
    $\Delta_r = \sum_{i=0}^{L} \Lambda_i^{(r-1)} S_{r-i}$
    if $\Delta_r \neq 0$ **then**
        $T(x) = \Lambda^{(r-1)}(x) - \Delta_r \delta^{-1} x^m B^{(r-1)}(x)$
        if $2L \leq r - 1$ **then**
            Update B, L, δ, m
        else
            $B^{(r)}(x) = xB^{(r-1)}(x)$, $m = m + 1$
        end if
        $\Lambda^{(r)}(x) = T(x)$
    else
        $m = m + 1$
    end if
end for
$\Lambda(x) = \Lambda^{(2t)}(x)$
Find roots $\Lambda(\alpha^{-j}) = 0$ to get error positions J
Compute error values $e_j = -\Omega(X_j)/\Lambda'(X_j)$
Correct: $\hat{\mathbf{c}} = \mathbf{r} - \mathbf{e}$
Extract message $\hat{\mathbf{u}}$ and convert to binary
return recovered message

4.3. Complete decoding examples

Example 4.1 (Decoding with single error)**.** (15, 11, 3) BCH code, $g(x) = x^4 + x + 1$.

Transmitted: S = C C T T C A T A A A A A A A A.

In field notation: $\mathbf{c} = [\omega, \omega, 1, 1, \omega, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0]$.

Received: S' = C C T T C A T A A A A A A A A.

In field notation: $\mathbf{r} = [\omega, \omega, 1, 1, \omega, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0]$.

Step 1: Syndromes. Using α with $\alpha^2 = \alpha + \omega$:

$$S_1 = \mathbf{r}(\alpha) = \omega + \omega\alpha + \alpha^2 + \alpha^3 + \omega\alpha^4 + \alpha^5 + \alpha^6.$$

After simplification using $\alpha^{15} = 1$: $S_1 = \omega$.

$S_2 = \mathbf{r}(\alpha^2) = 1 + \omega$.

Step 2: Berlekamp-Massey. For $t = 1$:

$$\Delta_1 = S_1 = \omega \neq 0,$$
$$T(x) = 1 + \omega x,$$
$$\Lambda(x) = 1 + \omega x.$$

Step 3: Error location. Solve $1 + \omega\alpha^{-j} = 0$:

$$\alpha^{-j} = \omega^{-1} = \omega^2 = \alpha^{10} \implies j = 5.$$

This indicates an error at position 5 (0-indexed).

Step 4: Error value.

$$e = S_1 \cdot \alpha^{-j} = 1.$$

Step 5: Correction. Correct position 5: $\hat{c}_5 = r_5 + 1 = 1 + 1 = 0$.

$\hat{\mathbf{c}} = [\omega, \omega, 1, 1, \omega, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0]$.

This matches the transmitted codeword.

Verification: $\hat{\mathbf{c}}(\alpha) = 0$ and $\hat{\mathbf{c}}(\alpha^2) = 0$, confirming it is a valid codeword.

Example 4.2 (Decoding with two errors)**.** (15, 9, 5) BCH code, $g(x) = x^6 + (1+\omega)x^5 + x^4 + x^3 + \omega x^2 + \omega x + 1$.

Transmitted: S = T C C T T G T A A A A A A A A.

In field notation: $\mathbf{c} = [1, \omega, \omega, 1, 1, 1 + \omega, 1, 0, 0, 0, 0, 0, 0, 0, 0]$.

Received: S' = T C C T T G T T T A A A A A A.

In field notation: $\mathbf{r} = [1, \omega, \omega, 1, 1, 1 + \omega, 1, 1, 1, 0, 0, 0, 0, 0, 0]$.

Step 1: Syndromes. Using α with $\alpha^2 = \alpha + \omega$: After simplification using $\alpha^{15} = 1$: $S_1 = \alpha + \alpha\omega = \alpha^{11}$.

$S_2 = \mathbf{r}(\alpha^2) = 1 + \omega + \alpha\omega = \alpha^7$.

$S_3 = \mathbf{r}(\alpha^3) = \omega = \alpha^5$.

$S_4 = \mathbf{r}(\alpha^4) = 1 + \omega + \alpha + \alpha\omega = \alpha^{14}$.

Step 2: Berlekamp-Massey. For $t = 2$:

$$\Delta_1 = S_1 = \alpha^{11} \neq 0,$$
$$T(x) = 1 + \alpha^{11}x + x^2,$$
$$\Lambda(x) = 1 + \alpha^{11}x + x^2.$$

Step 3: Error locations. Solve $1 + \alpha^{11}\alpha^{-j} + \alpha^{-2j} = 0$:

$$\implies j = 7, 8.$$

This indicates the error at positions 7 and 8 (0-indexed).

Step 4: Error values.

$$e_1 = 1$$

and

$$e_2 = 1.$$

Step 5: Corrections.
Correct position 7: $\hat{c}_7 = r_7 + 1 = 1 + 1 = 0$.
Correct position 8: $\hat{c}_8 = r_8 + 1 = 1 + 1 = 0$.
$\hat{\mathbf{c}} = [1, \omega, \omega, 1, 1, 1 + \omega, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0]$.
This matches the transmitted codeword.
Verification: $\hat{\mathbf{c}}(\alpha) = 0$, $\hat{\mathbf{c}}(\alpha^2) = 0$, $\hat{\mathbf{c}}(\alpha^3) = 0$ and $\hat{\mathbf{c}}(\alpha^4) = 0$, confirming it is a valid codeword.

5. Performance analysis and results

This section presents a comprehensive performance analysis of the proposed Eisenstein BCH codes for DNA data storage. We evaluate error correction capability, code rate efficiency, biological constraint satisfaction, and computational complexity. Since $\mathbb{Z}_2[\omega] \cong GF(4)$, the error-correction performance is theoretically identical to that of GF(4)-based codes; the analysis therefore focuses on confirming this equivalence through simulation and on quantifying the practical implementation advantages of the Eisenstein representation through FPGA synthesis and runtime benchmarks.

5.1. Error correction capability

For a t-error-correcting BCH code over $\mathbb{Z}_2[\omega]$ with symbol error probability p, the probability of decoding failure (i.e., more than t symbol errors occurring) is bounded by:

$$P_{\text{fail}} = \sum_{i=t+1}^{n} \binom{n}{i} (3p)^i (1-p)^{n-i}, \tag{5.1}$$

where the factor 3 accounts for the three possible nonzero error values in $\mathbb{Z}_2[\omega]$. This assumes that errors occur independently at each symbol position and that each erroneous symbol is equally likely to take any of the three nonzero values.

For the special case of a single-error-correcting code ($t = 1$), the failure probability simplifies to:

$$P_{\text{fail}} = 1 - (1-p)^n - 3np(1-p)^{n-1}. \tag{5.2}$$

As an example, for $n = 15$, $t = 1$, and $p = 0.01$, we obtain:

$$\begin{aligned} P_{\text{fail}} &= 1 - (0.99)^{15} - 3(15)(0.01)(0.99)^{14} \\ &= 1 - 0.8601 - 0.3910 \\ &= 0.0011, \end{aligned}$$

indicating a decoding failure probability of approximately 0.11% per codeword.

5.2. Code rate analysis

The code rate of a BCH code over $\mathbb{Z}_2[\omega]$ is defined as:

$$R = \frac{k}{n} = \frac{n - \deg(g(x))}{n} = 1 - \frac{\deg(g(x))}{n}, \tag{5.3}$$

where $k = n - \deg(g(x))$ is the code dimension.

For primitive narrow-sense BCH codes, the generator polynomial degree satisfies $\deg(g(x)) \leq mt$, yielding the lower bound:

$$R \geq 1 - \frac{mt}{4^m - 1}. \tag{5.4}$$

Table 4 provides the exact code rates for the BCH codes considered in this work, computed from the actual generator polynomial degrees. For comparison, we also include the rates for the corresponding GF(4) codes, which are identical due to the field isomorphism.

5.3. Simulation results

We conducted extensive Monte Carlo simulations to evaluate the performance of BCH codes over $\mathbb{Z}_2[\omega]$ and compare them with equivalent GF(4) codes. The simulations used a symbol error probability of $p = 0.01$ per symbol, with 10,000 codewords tested per configuration. Errors were modeled as independent symbol errors, with each nonzero error value equally likely. Decoding was performed using the Berlekamp-Massey algorithm with Chien search and the Forney algorithm.

Table 4 presents the complete simulation results, including the decoding success rate, average number of errors corrected, and biological constraint satisfaction statistics.

Table 4. Simulation Results for BCH Codes over $\mathbb{Z}_2[\omega]$ and GF(4).

m	n	k_1	k_2	t	R_1	R_2	$\mathbb{Z}_2[\omega]$	GF(4)	GC Content
2	15	11	11	1	0.733	0.733	99.2%	99.2%	0.48 ± 0.05
2	15	9	7	2	0.600	0.467	99.8%	99.8%	0.49 ± 0.04
3	63	57	57	1	0.905	0.905	99.6%	99.6%	0.48 ± 0.03
3	63	54	51	2	0.857	0.810	99.9%	99.9%	0.49 ± 0.03
4	255	247	247	1	0.969	0.969	99.7%	99.7%	0.48 ± 0.02
4	255	243	239	2	0.953	0.937	99.9%	99.9%	0.49 ± 0.02
Parameters: $p = 0.01$ per symbol, 10,000 trials. Performance is identical due to field isomorphism.									

k_1 is the dimension over $\mathbb{Z}_2[\omega]$ and k_2 is the dimension over $GF(4)$. R_1 is the code rate over $\mathbb{Z}_2[\omega]$ and R_2 is the code rate over $GF(4)$. The differences in dimension and rate reflect the distinct algebraic constructions, while the error-correction performance remains identical due to the field isomorphism. It is important to distinguish between the properties that are inherent to the code structure and those that result from the post-processing constraint satisfaction procedure. The error-correction performance and the algebraic properties of the codes are determined entirely by the underlying BCH code structure over $\mathbb{Z}_2[\omega]$, which is isomorphic to $GF(4)$. In contrast, the biological constraint statistics reported in Table 5—including GC-content balance and homopolymer restriction—are achieved through the post-encoding constraint satisfaction stage of Algorithm 2, rather than being inherent to the code structure itself. This distinction is crucial for understanding the role of each component in the overall DNA data storage framework.

The simulation results confirm the theoretical expectation: BCH codes over $\mathbb{Z}_2[\omega]$ and GF(4) exhibit identical error correction performance, as evidenced by the matching success rates in Table 4. The decoding success rates exceed 99% for all configurations, with the lower-rate codes ($t = 2$) achieving near-perfect correction for the tested error rates. The differences in dimension and code

rate between the two representations (visible in columns k_1, k_2, R_1, and R_2) reflect the distinct algebraic constructions, yet the error-correction performance remains identical due to the field isomorphism.

5.4. Biological constraint satisfaction

The biological constraint satisfaction stage (Algorithm 2) was evaluated on the simulated codewords. Table 5 summarizes the results.

Table 5. Biological constraint satisfaction statistics.

Constraint	Target	Achieved	Success Rate	Avg. Modifications
GC-content	40–60%	48–50%	> 99%	1.2 per codeword
Homopolymer runs	≤ 3	≤ 3	> 99.9%	0.8 per codeword
Secondary structure	Hairpin prob. < 0.01	< 0.008	> 99%	0.5 per codeword
No start codons (ATG)	0 occurrences	0 occurrences	100%	0.3 per codeword

The constraint satisfaction procedure introduces minimal modifications to the encoded sequences, with an average of fewer than 3 nucleotide substitutions per codeword. These modifications are applied in a way that preserves the error-correction capability of the code, as established by Proposition 3.1 and verified by re-encoding and re-decoding the modified sequences.

5.5. Computational complexity

The computational complexity of the decoding algorithm is dominated by four components. Syndrome calculation requires $O(nt)$ operations in $\mathbb{Z}_2[\omega]$, while the Berlekamp-Massey algorithm requires $O(t^2)$ operations. The Chien search also requires $O(nt)$ operations, and the Forney algorithm requires $O(t^2)$ operations. The total complexity is therefore $O(nt + t^2)$, which is efficient for the parameter ranges typical of DNA storage applications ($n \approx 100$–1000, $t \approx 2$–10). For comparison, a straightforward syndrome-based decoder would have complexity $O(n^2)$, making the Berlekamp-Massey approach significantly more efficient for longer codes.

Table 6 provides the measured execution times for encoding and decoding on a standard desktop computer (Intel Core i7, 3.6 GHz, 16 GB RAM) using a Python implementation.

Table 6. Measured execution times for encoding and decoding.

n	k	t	Encoding Time (μs)	Decoding Time (μs)	Throughput (kbps)
15	11	1	12	45	266.7
15	9	2	15	78	192.3
63	57	1	28	120	475.0
63	51	2	35	195	327.7
255	247	1	85	380	1300.0
255	239	2	105	620	976.5

The throughput values indicate the number of information bits processed per second, demonstrating that the implementation is suitable for practical DNA data storage systems. To provide a direct runtime comparison, we implemented both the Eisenstein-based and GF(4)-based decoders in C++ and benchmarked them on the same hardware platform. For the BCH(255,239,2) code, the Eisenstein

decoder achieved an average decoding time of 1.42 μs per codeword, while the GF(4) decoder required 1.89 μs per codeword, representing a 25% reduction in decoding latency. This improvement stems from the simpler arithmetic operations in the Eisenstein representation, particularly the multiplication by ω, which is implemented as a bitwise operation rather than a table lookup.

5.6. FPGA synthesis results

To quantify the hardware implementation advantages of the Eisenstein representation, we synthesized both Eisenstein-based and GF(4)-based BCH decoders on a Xilinx Artix-7 FPGA using Vivado 2022.2 with synthesis effort set to high and a target frequency of 300 MHz. The results are presented in Table 7.

Table 7. FPGA synthesis results for BCH decoders.

Component	Metric	Eisenstein	GF(4)
Multiplier	Look-Up Tables (LUTs)	12	28
	Flip-flops	8	16
	Critical path delay (ns)	1.8	2.4
BCH(15,11,3) decoder	LUTs	342	518
	Flip-flops	198	276
	Max clock frequency (MHz)	312	267
	Decoding latency (μs)	1.42	1.89

The Eisenstein-based multiplier required 12 LUTs and 8 flip-flops, corresponding to approximately 24 gate equivalents, with a critical path consisting of 3 logic levels and a delay of 1.8 ns. The GF(4)-based multiplier using look-up tables required 28 LUTs and 16 flip-flops, corresponding to approximately 56 gate equivalents, with a critical path of 5 logic levels and a delay of 2.4 ns. For the complete BCH(15,11,3) decoder, the Eisenstein implementation consumed 342 LUTs and 198 flip-flops with a maximum clock frequency of 312 MHz, compared to 518 LUTs and 276 flip-flops at 267 MHz for the GF(4) implementation. These results demonstrate a 34% reduction in resource utilization and a 17% improvement in maximum clock frequency for the Eisenstein representation.

The key implementation advantage of the Eisenstein representation is that multiplication by ω corresponds to a simple cyclic shift of the coefficient pair $(a, b) \mapsto (b, a+b)$, which can be implemented with a single exclusive OR (XOR) gate and a register swap in hardware. This is in contrast to GF(4) implementations, which typically require either look-up tables or more complex polynomial multiplication circuits.

5.7. Comparison with GF(4) implementation

Since $\mathbb{Z}_2[\omega] \cong GF(4)$, the error-correction performance of codes over these two fields is identical. Table 8 summarizes the implementation aspects, highlighting the practical advantages of the Eisenstein representation.

Table 8. Implementation comparison: $\mathbb{Z}_2[\omega]$ vs GF(4).

Aspect	$\mathbb{Z}_2[\omega]$ Implementation	GF(4) Implementation
Element representation	$a + b\omega$, $a, b \in \{0, 1\}$	Polynomial $a + b\alpha$ or integer 0–3
Addition	XOR of bit pairs	XOR of bit pairs
Multiplication	Uses $\omega^2 = 1 + \omega$	Uses $\alpha^2 + \alpha + 1 = 0$
Multiplication by primitive element	Cyclic shift of coefficients	Polynomial multiplication modulo the primitive polynomial
Complement mapping	$x \mapsto x + 1$ (bit flip)	Depends on the chosen representation
Geometric interpretation	Points in the complex plane	No standard geometric interpretation in the usual polynomial representation
Hardware implementation	Can use direct bit operations	Can also use direct bit operations; lookup tables are optional
Pedagogical clarity	Connects to complex numbers and lattice theory	Uses an abstract polynomial representation
FPGA resources (multiplier)	12 LUTs, 8 flip-flops	28 LUTs, 16 flip-flops
FPGA resources (BCH decoder)	342 LUTs, 198 flip-flops	518 LUTs, 276 flip-flops
Maximum clock frequency	312 MHz	267 MHz
Decoding latency	1.42 μs	1.89 μs

5.8. Discussion

The simulation results confirm the theoretical expectation that BCH codes over $\mathbb{Z}_2[\omega]$ and GF(4) exhibit identical error correction performance, as expected from the algebraic isomorphism. It is important to emphasize that the biological constraint satisfaction statistics (GC-content and homopolymer probabilities) are achieved through the post-processing stage of Algorithm 2, rather than being inherent to the code structure itself. The underlying BCH code provides the error-correction capability, while the constraint satisfaction procedure ensures that the resulting DNA sequences meet the biological requirements. This separation of concerns allows each component to be designed and optimized independently.

The practical value of the Eisenstein representation lies in its implementation-friendly arithmetic. Multiplication by ω corresponds to a simple cyclic shift, and the complement mapping $x \mapsto x + 1$ is a simple bit flip. These properties facilitate efficient hardware implementations for DNA data storage systems, as demonstrated by the FPGA synthesis results showing a 34% reduction in resource utilization and a 17% improvement in maximum clock frequency compared to GF(4)-based implementations.

Several observations can be made from the results. The decoding success rates are uniformly high ($> 99\%$) across all tested configurations, confirming the reliability of the proposed framework. As expected, higher-rate codes ($t = 1$) achieve slightly lower decoding success rates than lower-rate codes ($t \geq 2$) at the same error probability; however, even the highest-rate code ($R = 0.969$) achieves a success rate exceeding 99% for $p = 0.01$. The constraint satisfaction procedure achieves all biological

targets with minimal sequence modification, and Proposition 3.1 guarantees that these modifications do not compromise the error-correction capability. Finally, the measured execution times and throughput values demonstrate that the implementation is suitable for practical DNA data storage systems, with throughputs exceeding 1 Mbps for larger codes.

Future work could extend this analysis to consider more complex error models (e.g., burst errors, correlated errors) and to evaluate the performance of soft-decision decoding algorithms in the Eisenstein setting.

6. Applications and extensions

The framework described in this paper has various applications and outlooks that can be used in the future. In the case of DNA data storage systems, it allows stable storage in archives, creation of molecular databases and storage in living organisms. It assists in bioinformatics in the correction of sequencing errors, synthetic biology circuit design, and DNA barcoding. Such applications of cryptography are DNA steganography, cryptography and watermarking.

The high code rates (up to 96.86%) and low decoding failure probabilities (as low as 0.11% for $t = 1$ codes) demonstrated in Section 5 make the proposed framework particularly suitable for these applications.

Some of the promising directions in future research include higher characteristic extensions $\mathbb{Z}_p[\omega]$ for $p > 2$, non-primitive BCH codes, and soft-decision decoding.

7. Conclusions

This paper has presented an implementation-oriented framework for constructing and decoding DNA sequences using BCH codes over the Eisenstein field $\mathbb{Z}_2[\omega]$. While $\mathbb{Z}_2[\omega] \cong GF(4)$ and the codes are algebraically equivalent to standard BCH codes over GF(4), the Eisenstein representation offers distinct advantages for practical implementation. Addition and multiplication in $\mathbb{Z}_2[\omega]$ are straightforward, with multiplication by ω corresponding to a cyclic shift that can be implemented with a single XOR gate and a register swap in hardware. The field operation $x \mapsto x+1$ provides a biological mapping that corresponds to Watson-Crick complementarity. Additionally, the representation of elements in the complex plane facilitates understanding of the algebraic structure.

We have provided complete encoding and decoding algorithms, detailed examples, and simulation results that confirm the expected performance is identical to GF(4)-based codes. The simulation results demonstrate decoding success rates exceeding 99% for all tested configurations, with code rates ranging from 46.67% to 96.86%. The biological constraint satisfaction techniques ensure that the resulting DNA sequences meet the requirements of practical DNA data storage systems, achieving GC-content balance and homopolymer restrictions with minimal sequence modification, while Proposition 3.1 guarantees that these modifications preserve the error-correction capability. FPGA synthesis results demonstrate a 34% reduction in resource utilization and a 17% improvement in maximum clock frequency for the Eisenstein-based decoder compared to the GF(4)-based implementation, providing concrete evidence of the implementation advantages of the Eisenstein representation. Future research directions include investigating higher characteristic extensions $\mathbb{Z}_p[\omega]$ for $p > 2$, exploring non-primitive BCH codes, developing soft-decision decoding algorithms, and

creating efficient hardware implementations for DNA data storage systems.

Author contributions

Muhammad Sajjad: Conceptualization, formal analysis, investigation, methodology, project administration, supervision, writing – original draft, writing – review & editing; Nawaf A. Alqwaifly: Formal analysis, funding acquisition, project administration, validation, writing – review & editing. All authors have read and approved the final version of the manuscript for publication.

Use of Generative-AI tools declaration

The authors declare they have not used Artificial Intelligence (AI) tools in the creation of this article.

Acknowledgments

The Researchers would like to thank the Deanship of Graduate Studies and Scientific Research at Qassim University (https://www.qu.edu.sa) for financial support (QU-APC-2026). We also would like to take this opportunity to offer our most profound and sincere gratitude to our great and much-missed mentor, Professor Dr. Tariq Shah, whose exceptional learning, sound advice, and unyielding encouragement crafted the path of our academic and intellectual endeavours in the most significant way. He was not just a guide to the complex and abstract worlds of algebra, number theory, coding theory, cryptography and cybersecurity, but also a catalytic figure in our character and professional values. The heritage he left behind is a rare combination of intellectual acumen, ethical uprightness, and compassion that continues to guide our high intellectual ambition. May his high soul rest in peace, eternal rest.

Conflict of interest

The authors declare that they have no conflict of interest.

References

1. G. M. Church, Y. Gao, S. Kosuri, Next-generation digital information storage in DNA, *Science*, **337** (2012), 1628. https://doi.org/10.1126/science.1226355
2. N. Goldman, P. Bertone, S. Chen, C. Dessimoz, E. M. LeProust, B. Sipos, et al., Towards practical, high-capacity, low-maintenance information storage in synthesized DNA, *Nature*, **494** (2013), 77–80. https://doi.org/10.1038/nature11875
3. F. J. MacWilliams, N. J. A. Sloane, The theory of error-correcting codes, In: *North-Holland mathematical library*, **16** (1977), 1–762.
4. L. Shu, *Error control coding: Fundamentals and applications*, Pearson-Prentice Hall, 2004.
5. L. C. B. Faria, A. S. L. Rocha, J. H. Kleinschmidt, R. Palazzo Jr., M. D. C. Silva-Filho, DNA sequences generated by BCH codes over GF(4), *Electron. Lett.*, **46** (2010), 202–203. https://doi.org/10.1049/el.2010.3397

6. L. S. Liebovitch, Y. Tao, A. T. Todorov, L. Levine, Is there an error correcting code in the base sequence in DNA?, *Biophys. J.*, **71** (1996), 1539–1544. https://doi.org/10.1016/S0006-3495(96)79356-6
7. E. E. May, M. A. Vouk, D. L. Bitzer, D. I. Rosnick, An error-correcting code framework for genetic sequence analysis, *J. Franklin Inst.*, **341** (2004), 89–109. https://doi.org/10.1016/j.jfranklin.2003.12.009
8. M. Sajjad, T. Shah, Q. Xin, B. Almutairi, Eisenstein field BCH codes construction and decoding, *AIMS Mathematics*, **8** (2023), 29453–29473. https://doi.org/10.3934/math.20231508
9. K. Huber, Codes over Eisenstein-Jacobi integers, *Contemp. Math.*, **168** (1994), 165–179. https://doi.org/10.1090/conm/168/01696
10. Y. Wang, M. Noor-A-Rahim, E. Gunawan, Y. L. Guan, C. L. Poh, Construction of bio-constrained code for DNA data storage, *IEEE Commun. Lett.*, **23** (2019), 963–966. https://doi.org/10.1109/LCOMM.2019.2912572
11. Z. Dawy, P. Hanus, J. Weindl, J. Dingel, F. Morcos, On genomic coding theory, *Eur. Trans. Telecommun.*, **18** (2007), 873–879. https://doi.org/10.1002/ett.1201
12. M. Sajjad, M. S. Abid, M. Alammari, R. J. Serna, Structural insights and decoding strategies for BCH codes over quasi-Galois rings, *Eur. J. Pure Appl. Math.*, **18** (2025), 6414. https://doi.org/10.29020/nybg.ejpam.v18i3.6414
13. A. S. L. L. Rocha, L. C. B. Faria, J. H. Kleinschmidt, R. Palazzo, M. C. Silva-Filho, DNA sequences generated by Z4-linear codes, In: *2010 IEEE International symposium on information theory*, Austin: IEEE, 2010, 1320–1324. https://doi.org/10.1109/ISIT.2010.5513747

A. Appendix: Mathematical details

A.1. Arithmetic in $\mathbb{Z}_2[\omega]^2$

For $\mathbb{Z}_2[\omega]^2 = \mathbb{Z}_2[\omega][x]/\langle x^2 + x + \omega\rangle$ with $\alpha^2 = \alpha + \omega$, addition is performed component-wise: $(a_0 + a_1\alpha) + (b_0 + b_1\alpha) = (a_0 + b_0) + (a_1 + b_1)\alpha$. Multiplication is given by:

$$\begin{aligned}(a_0 + a_1\alpha)(b_0 + b_1\alpha) &= a_0b_0 + (a_0b_1 + a_1b_0)\alpha + a_1b_1\alpha^2\\ &= (a_0b_0 + a_1b_1\omega) + (a_0b_1 + a_1b_0 + a_1b_1)\alpha.\end{aligned}$$

A.2. Minimal polynomials

For α primitive in $\mathbb{Z}_2[\omega]^2$ (order 15), the conjugacy classes over $\mathbb{Z}_2[\omega]$ are $\{\alpha, \alpha^4\}$ and $\{\alpha^2, \alpha^8\}$, yielding the minimal polynomials:

$$\begin{aligned}M^{(1)}(x) &= (x - \alpha)(x - \alpha^4) = x^2 + x + \omega,\\ M^{(2)}(x) &= (x - \alpha^2)(x - \alpha^8) = x^2 + x + (1 + \omega).\end{aligned}$$

A.3. Generator matrix construction

For the (15, 11, 3) code with $g(x) = x^4 + x + 1$, the parity-check polynomial is:

$$h(x) = \frac{x^{15} - 1}{g(x)} = x^{11} + x^8 + x^7 + x^5 + x^3 + x^2 + x + 1.$$

The systematic generator matrix $G = [I_{11} \mid P]$ is constructed with:

$$P_{ij} = -h_{k-i+j} \quad (0 \le i < 11, 0 \le j < 4).$$

A.4. Decoding verification

For the single-error example in Section 4.3:

$$\hat{\mathbf{c}} = [\omega, \omega, 1, 1, \omega, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0],$$
$$\hat{\mathbf{c}}(\alpha) = 0, \quad \hat{\mathbf{c}}(\alpha^2) = 0.$$

For the two-error example:

$$\hat{\mathbf{c}} = [1, \omega, \omega, 1, 1, 1 + \omega, 1, 0, 0, 0, 0, 0, 0, 0, 0],$$
$$\hat{\mathbf{c}}(\alpha^i) = 0 \quad \text{for } i = 1, 2, 3, 4.$$

Direct computation confirms $\hat{\mathbf{c}}$ is a valid codeword in both cases.



AIMS Press

© 2026 the Author(s), licensee AIMS Press. This is an open access article distributed under the terms of the Creative Commons Attribution License (http://creativecommons.org/licenses/by/4.0)