Formal Proof Dossier

Article 98

Quaternary Coding Theory of the Biological Ledger over GF(4)

Charles D. Schaper, Ph.D.

Volume 1, Number 4 November 2026

Certification Summary

Lean 4.24.0 freshly compiled all 33 named theorems in the source below. Every theorem was audited with `#print axioms`; all are axiom-free. There are no admitted proofs or user axioms. This is a scoped finite formalization, not a formal proof of every statement in the manuscript.

1 Formal coverage

The script retains the supplied explicit four-element arithmetic and expands its algebraic coverage to associativity and nonzero inverses. It proves the register identity and additive coordinate projection from $\mathbb{F}_2^3$ to $\mathbb{F}_2^2$. No field homomorphism from GF(8) to GF(4) is asserted: such a nonzero unital field homomorphism cannot exist.

The locus and hexamer code lower weight bounds have explicit attaining witnesses. The locus Euclidean self-orthogonality and the dual simplex weight are checked for all inputs. The full five-symbol decoder now corrects every received word to the parity-check kernel with an error of weight at most one, and all five single-error contact positions are covered. A separate certificate shows that Euclidean self-orthogonality does not imply Hermitian self-orthogonality here.

The square-boundary identity and finite CSS dimension arithmetic are included. The product parameters theorem verifies only the parameter arithmetic; it does not formalize the general tensor-product distance theorem. Full product enumeration, large cubical matrix ranks, graph cut/girth computations, and failure-probability benchmarks are independently checked by Python. The general mathematical proofs remain in the manuscript. Chemical realization and physical decoding are not formalized.

2 Inherited proof audit and reproduction

All eleven theorems extracted from the supplied Lean listing compiled under Lean 4.24.0. The earlier displayed decoder theorem covered only contact zero, and the projection proved an additive identity, not a field map. This dossier corrects those coverage descriptions and adds the other contacts and complete received-word result.

Run `LEAN=/path/to/lean python audit_lean.py`. The script checks the pinned source hash, compiles it, and requires an axiom-free audit entry for every named theorem. The source listing below is byte-identical to the executed file.

3 Complete Lean source

```
set_option maxRecDepth 100000
set_option maxHeartbeats 0
```

```
/-
  FormalProof_A98.lean
  Machine-Checked Formal Proof Dossier for Article 98:
  "Quaternary Coding Theory of the Biological Ledger over GF(4)"
  Annals of the Chrysene Formalism, Vol. 1, No. 4, Article 98 (November 2026).

  Verified in Lean 4 (Core library only; pure ASCII, zero `sorry`, zero external
     dependencies).
-/

namespace Article98

-- ============================================================================
-- 1. Field Arithmetic of GF(4) = {0, 1, om, om2}
-- ============================================================================

inductive GF4 where
  | zero : GF4
  | one  : GF4
  | om   : GF4
  | om2  : GF4
  deriving DecidableEq, Repr

open GF4

abbrev GF4Pair  := Prod GF4 GF4
abbrev GF4Word5 := Prod GF4 (Prod GF4 (Prod GF4 (Prod GF4 GF4)))

def add (x y : GF4) : GF4 :=
  match x, y with
  | zero, a => a
  | a, zero => a
  | one, one => zero
  | om,  om  => zero
  | om2, om2 => zero
  | one, om  => om2
  | om,  one => om2
  | one, om2 => om
  | om2, one => om
  | om,  om2 => one
  | om2, om  => one

def mul (x y : GF4) : GF4 :=
  match x, y with
  | zero, _ => zero
  | _, zero => zero
  | one, a  => a
  | a, one  => a
  | om,  om  => om2
  | om2, om2 => om
  | om,  om2 => one
  | om2, om  => one

instance : Add GF4 where add := add
instance : Mul GF4 where mul := mul

theorem gf4_char_two (x : GF4) : x + x = zero := by
  cases x <;> rfl
```

```
theorem gf4_minimal_poly : (om * om) + om + one = zero /\ (om * om * om) = one
    := by
  decide

theorem gf4_comm_ring (x y z : GF4) :
    (x + y = y + x) /\ (x * y = y * x) /\ (x * (y + z) = (x * y) + (x * z)) :=
    by
  cases x <;> cases y <;> cases z <;> decide

-- ========================================================================
-- 2. Register Automorphism chi and Planar Collapse Additive F2^3 -> F2^2
    projection
-- ========================================================================

def xorB (a b : Bool) : Bool := bne a b

def phi (a b : Bool) : GF4 :=
  match a, b with
  | false, false => zero
  | true,  false => one
  | false, true  => om
  | true,  true  => om2

-- Register identity: Register shift chi(a, b) = (b, a xor b) equals
    multiplication by omega
theorem chi_is_mul_omega (a b : Bool) :
    phi b (xorB a b) = om * (phi a b) := by
  cases a <;> cases b <;> rfl

-- Additive projection only; no homomorphism of fields is asserted:
-- Planar Collapse projection pi_{8->4}(a1, a2, a3) = phi(a1, a2) is a group
    homomorphism
def planarCollapse (a1 a2 _a3 : Bool) : GF4 := phi a1 a2

theorem planarCollapse_hom (a1 a2 a3 b1 b2 b3 : Bool) :
    planarCollapse (xorB a1 b1) (xorB a2 b2) (xorB a3 b3) =
    (planarCollapse a1 a2 a3) + (planarCollapse b1 b2 b3) := by
  cases a1 <;> cases a2 <;> cases b1 <;> cases b2 <;> rfl

-- ========================================================================
-- 3. The [4, 2, 3]_4 Self-Dual MDS Locus Evaluation Code
-- ========================================================================

def wtBit (x : GF4) : Nat :=
  match x with
  | zero => 0
  | _    => 1

-- Hamming weight of the evaluation word ev(c0, c1) = (f(0), f(1), f(om), f(om2)
    )
def wtLocus (c0 c1 : GF4) : Nat :=
  wtBit c0 +
  wtBit (c0 + c1) +
  wtBit (c0 + (c1 * om)) +
  wtBit (c0 + (c1 * om2))

-- Euclidean bilinear form <ev(c0, c1), ev(d0, d1)> on GF(4)^4
def dotLocus (c0 c1 d0 d1 : GF4) : GF4 :=
  (c0 * d0) +
```

```
  ((c0 + c1) * (d0 + d1)) +
  ((c0 + (c1 * om)) * (d0 + (d1 * om))) +
  ((c0 + (c1 * om2)) * (d0 + (d1 * om2)))

-- Locus lower weight bound; an attaining witness is proved below.
theorem c_loc_mds_distance (c0 c1 : GF4) :
    (c0 = zero /\ c1 = zero) \/ (wtLocus c0 c1 >= 3) := by
  cases c0 <;> cases c1 <;> decide

-- Euclidean self-orthogonality; duality also requires the dimension argument.
theorem c_loc_self_orthogonal (c0 c1 d0 d1 : GF4) :
    dotLocus c0 c1 d0 d1 = zero := by
  cases c0 <;> cases c1 <;> cases d0 <;> cases d1 <;> rfl

-- ============================================================================
-- 4. The Perfect MDS [5, 3, 3]_4 Hexamer Contact Code
-- ============================================================================

-- Systematic generator G_5(u0, u1, u2) = (p0, p1, u0, u1, u2) in GF(4)^5
def hexP0 (u0 u1 u2 : GF4) : GF4 := u0 + u1 + u2
def hexP1 (u0 u1 u2 : GF4) : GF4 := u0 + (om * u1) + (om2 * u2)

-- Parity-check syndrome s = H_5(y0, y1, y2, y3, y4) in GF(4)^2
def syndrome5 (y0 y1 y2 y3 y4 : GF4) : GF4Pair :=
  (y0 + y2 + y3 + y4, y1 + y2 + (om * y3) + (om2 * y4))

-- Hamming weight of a 5-symbol contact word in GF(4)^5
def wt5 (y0 y1 y2 y3 y4 : GF4) : Nat :=
  wtBit y0 + wtBit y1 + wtBit y2 + wtBit y3 + wtBit y4

-- Encoding: Every encoded hexamer word lies in ker(H_5)
theorem c_hex_syndrome_zero (u0 u1 u2 : GF4) :
    syndrome5 (hexP0 u0 u1 u2) (hexP1 u0 u1 u2) u0 u1 u2 = (zero, zero) := by
  cases u0 <;> cases u1 <;> cases u2 <;> rfl

-- Hexamer lower weight bound; an attaining witness is proved below.
theorem c_hex_mds_distance (u0 u1 u2 : GF4) :
    (u0 = zero /\ u1 = zero /\ u2 = zero) \/
    (wt5 (hexP0 u0 u1 u2) (hexP1 u0 u1 u2) u0 u1 u2 >= 3) := by
  cases u0 <;> cases u1 <;> cases u2 <;> decide

-- Dual simplex: Dual code C_hex^perp = rowspan(H_5) is an equidistant [5, 2, 4]
    _4 simplex code
theorem c_hex_dual_equidistant (a0 a1 : GF4) :
    (a0 = zero /\ a1 = zero) \/
    (wt5 a0 a1 (a0 + a1) (a0 + (om * a1)) (a0 + (om2 * a1)) = 4) := by
  cases a0 <;> cases a1 <;> decide

-- Decoder: Bijective single-error syndrome decoder on GF(4)^5
def decodeSyndrome (s0 s1 : GF4) : GF4Word5 :=
  if s1 = zero then (s0, zero, zero, zero, zero)
  else if s0 = zero then (zero, s1, zero, zero, zero)
  else if s0 = s1 then (zero, zero, s0, zero, zero)
  else if s1 = om * s0 then (zero, zero, zero, s0, zero)
  else (zero, zero, zero, zero, s0)

-- Verifies that any single-contact error e0 != 0 at contact 0 is uniquely
    identified
theorem c_hex_perfect_decoding_contact0 (u0 u1 u2 e0 : GF4) :
```

```
    let y0 := (hexP0 u0 u1 u2) + e0
    let y1 := hexP1 u0 u1 u2
    let s  := syndrome5 y0 y1 u0 u1 u2
    decodeSyndrome s.1 s.2 = (e0, zero, zero, zero, zero) := by
  cases u0 <;> cases u1 <;> cases u2 <;> cases e0 <;> rfl



-- Additional actual algebraic certificates, not opaque parameter declarations.
theorem gf4_add_assoc (a b c : GF4) : (a+b)+c=a+(b+c) := by
 cases a <;> cases b <;> cases c <;> rfl
theorem gf4_mul_assoc (a b c : GF4) : (a*b)*c=a*(b*c) := by
 cases a <;> cases b <;> cases c <;> rfl
theorem gf4_inverse (a : GF4) : a=zero \/ a*(a*a)=one := by
 cases a <;> decide
theorem projection_fiber_two :
 planarCollapse false false false = planarCollapse false false true := by rfl
theorem locus_distance_witness : wtLocus zero one = 3 := by decide
theorem locus_four_witness : wtLocus one zero = 4 := by decide
theorem hex_distance_witness : wt5 (hexP0 one zero zero) (hexP1 one zero zero)
    one zero zero = 3 := by decide
-- Full five-symbol word representation and decoder, all received words.
def word5 (a b c d e : GF4) : GF4Word5 := (a,b,c,d,e)
def syndromeWord (w : GF4Word5) : GF4Pair :=
 syndrome5 w.1 w.2.1 w.2.2.1 w.2.2.2.1 w.2.2.2.2
def addWord (u v : GF4Word5) : GF4Word5 :=
 (u.1+v.1, u.2.1+v.2.1, u.2.2.1+v.2.2.1,
 u.2.2.2.1+v.2.2.2.1,u.2.2.2.2+v.2.2.2.2)
def weightWord (w : GF4Word5) : Nat :=
 wt5 w.1 w.2.1 w.2.2.1 w.2.2.2.1 w.2.2.2.2
def corrected (w : GF4Word5) : GF4Word5 :=
 let s := syndromeWord w
 addWord w (decodeSyndrome s.1 s.2)
theorem decoder_right_inverse (a b : GF4) :
 syndromeWord (decodeSyndrome a b)=(a,b) := by
 cases a <;> cases b <;> rfl
theorem decoder_weight_bound (a b : GF4) :
 weightWord (decodeSyndrome a b) <= 1 := by
 cases a <;> cases b <;> decide
theorem every_word_corrects (a b c d e : GF4) :
 syndromeWord (corrected (word5 a b c d e)) = (zero,zero) := by
 cases a <;> cases b <;> cases c <;> cases d <;> cases e <;> rfl
theorem decode_contact1 (u0 u1 u2 err : GF4) :
 let sy := syndrome5 (hexP0 u0 u1 u2) ((hexP1 u0 u1 u2) + err) (u0) (u1) (u2)
 decodeSyndrome sy.1 sy.2 = (zero, err, zero, zero, zero) := by
 cases u0 <;> cases u1 <;> cases u2 <;> cases err <;> rfl
theorem decode_contact2 (u0 u1 u2 err : GF4) :
 let sy := syndrome5 (hexP0 u0 u1 u2) (hexP1 u0 u1 u2) ((u0) + err) (u1) (u2)
 decodeSyndrome sy.1 sy.2 = (zero, zero, err, zero, zero) := by
 cases u0 <;> cases u1 <;> cases u2 <;> cases err <;> rfl
theorem decode_contact3 (u0 u1 u2 err : GF4) :
 let sy := syndrome5 (hexP0 u0 u1 u2) (hexP1 u0 u1 u2) (u0) ((u1) + err) (u2)
 decodeSyndrome sy.1 sy.2 = (zero, zero, zero, err, zero) := by
 cases u0 <;> cases u1 <;> cases u2 <;> cases err <;> rfl
theorem decode_contact4 (u0 u1 u2 err : GF4) :
 let sy := syndrome5 (hexP0 u0 u1 u2) (hexP1 u0 u1 u2) (u0) (u1) ((u2) + err)
 decodeSyndrome sy.1 sy.2 = (zero, zero, zero, zero, err) := by
 cases u0 <;> cases u1 <;> cases u2 <;> cases err <;> rfl
```

```
-- Euclidean and Hermitian products must not be conflated.
def hermitianLocus (c0 c1 d0 d1 : GF4) : GF4 :=
 (c0*(d0*d0)) + ((c0+c1)*((d0+d1)*(d0+d1))) +
 ((c0+c1*om)*((d0+d1*om)*(d0+d1*om))) +
 ((c0+c1*om2)*((d0+d1*om2)*(d0+d1*om2)))
theorem not_hermitian_self_orthogonal :
 hermitianLocus zero one zero one = one := by decide
-- Explicit square boundary, incidence in characteristic two.
def faceBoundary : List (List GF4) :=
 [[one,one,zero,zero],[one,zero,one,zero],
 [zero,one,zero,one],[zero,zero,one,one]]
theorem square_boundary_zero :
 faceBoundary.all (fun r => r.foldl add zero == zero) = true := by decide
theorem full_css_dimension : 117-53-64=0 := by decide
theorem removed_basis_check_dimension : 117-53-63=1 := by decide
theorem product_parameters : 4*5=20 /\ 2*3=6 /\ 3*3=9 /\ (9-1)/2=4 := by decide
theorem hamming_sphere_partition : 64*(1+5*3)=4^5 := by decide
theorem redundancy_bits : 2*5-2*3=4 := by decide
theorem box_counts : 3*3*6=54 /\ 2*3*6+3*2*6+3*3*5=117 := by decide
end Article98

#print axioms Article98.gf4_char_two
#print axioms Article98.gf4_minimal_poly
#print axioms Article98.gf4_comm_ring
#print axioms Article98.chi_is_mul_omega
#print axioms Article98.planarCollapse_hom
#print axioms Article98.c_loc_mds_distance
#print axioms Article98.c_loc_self_orthogonal
#print axioms Article98.c_hex_syndrome_zero
#print axioms Article98.c_hex_mds_distance
#print axioms Article98.c_hex_dual_equidistant
#print axioms Article98.c_hex_perfect_decoding_contact0
#print axioms Article98.gf4_add_assoc
#print axioms Article98.gf4_mul_assoc
#print axioms Article98.gf4_inverse
#print axioms Article98.projection_fiber_two
#print axioms Article98.locus_distance_witness
#print axioms Article98.locus_four_witness
#print axioms Article98.hex_distance_witness
#print axioms Article98.decoder_right_inverse
#print axioms Article98.decoder_weight_bound
#print axioms Article98.every_word_corrects
#print axioms Article98.decode_contact1
#print axioms Article98.decode_contact2
#print axioms Article98.decode_contact3
#print axioms Article98.decode_contact4
#print axioms Article98.not_hermitian_self_orthogonal
#print axioms Article98.square_boundary_zero
#print axioms Article98.full_css_dimension
#print axioms Article98.removed_basis_check_dimension
#print axioms Article98.product_parameters
#print axioms Article98.hamming_sphere_partition
#print axioms Article98.redundancy_bits
#print axioms Article98.box_counts
```