The Dual Cipher, Structurally Induced GF(4) Multiplication, and Real Spectral Structure

Companion-Matrix Realization of the Register Automorphism, Frobenius Reversal, and the Order-One Commutant Condition

Lean 4 Formal Proof Companion

Charles D. Schaper, Ph.D.
Annals of the Chrysene Formalism, Volume 1, Number 4
Article 92, November 2026

1 Formal scope and result

This is a **partial formal proof companion** to revised Article 92. Lean 4.24.0 freshly compiled **53 named theorems**. The source imports only `Std`; it uses kernel-checked `decide` and explicit proofs, not `native_decide`. There is no `sorry`, `admit`, custom axiom or unsafe declaration.

The axiom audit records 46 declarations without axiom dependencies, one using propositional extensionality, and six using propositional extensionality and quotient soundness. These are standard Lean axioms, not assumptions added about the manuscript's physical model.

2 Definitions and coverage

The four-element field.
Boolean pairs define addition and the stated multiplication explicitly. Theorems quantify over every pair or triple and prove additive and multiplicative laws, identity, nonzero inverses, the companion relation, its exact order, gate composition, the dual-number alternative, Frobenius, trace and norm. They establish a concrete field-law certificate; no abstract Mathlib field instance is imported.

RNA and frames.
Four explicit equations certify the corrected dictionary. Lists enumerate all three-register words, certify both frame involutions and commutation, and obtain the four fixed counts. The equality $(64 + 0 + 8 + 8)/4 = 20$ is a numerical certificate; the general Burnside theorem is not formalized here.

Perimeter and support.
The four locant relations commute with the defined inversion. The crossed-support involution is proved for arbitrary integers. Symmetric radius invariance is checked on $[-10, 10]^2$, and the original radius failure is exhibited. The arbitrary-support and operator-domain analysis remains in the manuscript.

Global obstruction.
The arbitrary-integer identity $0 - n - n + 0 = -2n$ and its nonvanishing for $n > 0$ are proved. Connecting this scalar coefficient to the product-trace vacuum and site unitary uses the manuscript's explicit representation argument; that infinite representation is not formalized in Lean.

Cross term.
For every four matrix-unit indices, the four superoperator positions of $[L(F)R(G), L(E_{ij})]$ followed by commutation with $R(E_{kl})$ are distinct. Their nonzero coefficients give the manuscript's

256 violations. Lean checks distinct positions and the count; the symbolic expansion into those positions is explained in the manuscript and independently evaluated by Python.

Integer matrix algebra.
A fully defined 2×2 integer matrix algebra proves associativity, commutation of left and right multiplication, and the left-generator double-commutator identity for arbitrary entries. The right-generator contribution vanishes by left/right commutation. This is a formal algebraic instance, not a claimed formal proof of the full complex M_4 theorem. The latter remains a manuscript proof with exact Python examples.

Doubling factors.
Explicit integer Pauli matrices certify their squares, grading anticommutation and doubling-factor identities. These support the finite sign calculation; the complete complex antiunitary 32×32 example is verified by Python, not claimed as a Lean theorem.

3 Reproduction and trust boundary

With Lean 4.24.0 available, run `lean FormalProof_A92.lean`, or run `python audit_lean.py` with `LEAN` set to the executable. The audit compiles an exact source copy with `#print axioms` for every named theorem and records the compiler version, hash and output. Only the standard library is required.

The standalone LaTeX source embeds the exact Lean source as a Base64 file through `filecontents*`. Decode the emitted Base64 file to recover the identical Lean source. The package also supplies the decoded source directly. Overleaf typesetting does not compile Lean. The package supplies the toolchain declaration, executable audit script, log and JSON record separately.

4 Concordance and remaining scope

The finite field construction is relative to the declared coordinate identification and cyclic vector. The corrected source separates the finite internal real triple from a J-compatible infinite filtration that still fails order-one. It also separates a four-state algebraic encoding from molecular realization. No theorem in this file claims chemical synthesis, an empirical genetic mechanism, an infinite spectral-triple completion or a physical identification of locant displacement with substrate orientation.

5 Complete Lean source

FormalProof_A92.lean

```
import Std
/- Article 92 finite certificates. Lean 4.24.0; no external libraries.
   Does not formalize chemical realization or an infinite spectral triple. -/
set_option maxRecDepth 100000
set_option maxHeartbeats 0
namespace Article92
abbrev F4 := Bool × Bool
instance (p : F4 → Prop) [DecidablePred p] : Decidable (∀ x, p x) :=
 decidable_of_iff (∀ a b : Bool, p (a,b)) ⟨fun h x => h x.1 x.2, fun h a b => h (a,b)⟩
instance (p : F4 → Prop) [DecidablePred p] : Decidable (∃ x, p x) :=
 decidable_of_iff (∃ a b : Bool, p (a,b)) ⟨fun ⟨a,b,h⟩ => ⟨(a,b),h⟩, fun ⟨x,h⟩ => ⟨x.1,x.2,h⟩⟩
def add (x y : F4) : F4 := (xor x.1 y.1, xor x.2 y.2)
def mul (x y : F4) : F4 :=
 (xor (x.1 && y.1) (x.2 && y.2),
  xor (xor (x.1 && y.2) (x.2 && y.1)) (x.2 && y.2))
def zero : F4 := (false,false)
def one : F4 := (true,false)
def w : F4 := (false,true)
def chi (x : F4) : F4 := (x.2,xor x.1 x.2)
def sq (x : F4) : F4 := mul x x
def flip (x : F4) : F4 := add x one
def c (x : F4) : F4 := (x.1,xor x.1 x.2)
def swap (x : F4) : F4 := (x.2,x.1)
def elems : List F4 := [(false,false),(true,false),(false,true),(true,true)]
theorem add_assoc : ∀ x y z : F4, add (add x y) z = add x (add y z) := by decide
theorem add_comm : ∀ x y : F4, add x y = add y x := by decide
theorem add_zero : ∀ x : F4, add x zero = x := by decide
theorem add_self : ∀ x : F4, add x x = zero := by decide
theorem mul_assoc : ∀ x y z : F4, mul (mul x y) z = mul x (mul y z) := by decide
theorem mul_comm : ∀ x y : F4, mul x y = mul y x := by decide
theorem mul_one : ∀ x : F4, mul x one = x := by decide
theorem distributive : ∀ x y z : F4, mul x (add y z) = add (mul x y) (mul x z) := by decide
theorem inverse_exists : ∀ x : F4, x ≠ zero → ∃ y : F4, mul x y = one := by decide
theorem companion : ∀ x : F4, add (add (chi (chi x)) (chi x)) x = zero := by decide
theorem order_three : ∀ x : F4, chi (chi (chi x)) = x := by decide
theorem not_order_one : chi one ≠ one := by decide
theorem not_order_two : chi (chi one) ≠ one := by decide
theorem gate_cs : ∀ x : F4, c (swap x) = chi x := by decide
theorem gate_sc : ∀ x : F4, swap (c x) = chi (chi x) := by decide
theorem rejected_gate_involution : ∀ x : F4, c (c x) = x := by decide
theorem rejected_gate_nilpotent : ∀ x : F4, add (c (add (c x) x)) (add (c x) x) = zero := by
    decide
theorem omega_action : ∀ x : F4, mul w x = chi x := by decide
theorem frobenius_register : ∀ x : F4, sq (chi x) = chi (swap x) := by decide
theorem register_flip : ∀ x : F4, chi (add x (true,true)) = flip (chi x) := by decide
theorem trace : ∀ x : F4, add x (sq x) = (x.2,false) := by decide
theorem norm_nonzero : ∀ x : F4, mul x (sq x) = (x != zero,false) := by decide
theorem frobenius_multiplicative : ∀ x y : F4, sq (mul x y) = mul (sq x) (sq y) := by decide
theorem frobenius_involution : ∀ x : F4, sq (sq x) = x := by decide

theorem dictionary_A : chi (false,false) = zero := by decide
theorem dictionary_C : chi (false,true) = (true,true) := by decide
theorem dictionary_G : chi (true,false) = w := by decide
theorem dictionary_U : chi (true,true) = one := by decide
```

```
def words : Nat → List (List F4)
 | 0 => [[]]
 | n+1 => elems.flatMap fun x => (words n).map fun t => x::t
def fl (t : List F4) := t.map flip
def rev (t : List F4) := t.reverse.map sq
def fixed (f : List F4 → List F4) : Nat := ((words 3).filter fun t => f t == t).length
theorem codons : (words 3).length = 64 := by decide
theorem fixed_id : fixed id = 64 := by decide
theorem fixed_flip : fixed fl = 0 := by decide
theorem fixed_reverse : fixed rev = 8 := by decide
theorem fixed_flip_reverse : fixed (fun t => fl (rev t)) = 8 := by decide
theorem frame_involutions : ((words 3).all fun t => fl (fl t) == t && rev (rev t) == t) =
    true := by decide
theorem frame_commutation : ((words 3).all fun t => fl (rev t) == rev (fl t)) = true := by
    decide
theorem burnside_count : (64+0+8+8)/4 = 20 := by decide

def pri : List Nat := [3,6,9,12]
def delta (j : Nat) := j-1
def iota (j : Nat) := if j ≤ 6 then j+6 else j-6
theorem locant_commutation : (pri.all fun j => delta (iota j) == iota (delta j)) = true :=
    by decide

def h (k m : Int) : Nat := max k.natAbs m.natAbs
def hs (k m : Int) : Nat := max (max k.natAbs (k-m).natAbs) m.natAbs
def window : List Int := (List.range 21).map fun k : Nat => (Int.ofNat k)-10
theorem support_involution : ∀ k m : Int, (k-m)-(-m) = k := by intros; omega
theorem symmetric_height_window : (window.all fun k => window.all fun m => hs k m == hs (k-m
    ) (-m)) = true := by decide
theorem uncalibrated_height_counterexample : h 1 (-1) = 1 ∧ h 2 1 = 2 := by decide
-- Vacuum and a trace-zero site unitary have degrees 0 and n;
-- L(a)R(a) takes the vacuum back to itself. The residual is -2n.
theorem site_double_commutator (n : Int) : 0-n-n+0 = -2*n := by omega
theorem site_residual_nonzero (n : Int) (hn : n > 0) : 0-n-n+0 ≠ 0 := by omega

-- Exact cross-term certificate on the 16 matrix units of M4.
-- [F,Eij] has the two distinct positions (i xor 2,j), (i,j xor 2).
-- [Ekl,G] has (k,l xor 1), (k xor 1,l).
-- Their left/right composition has four distinct superoperator positions.
abbrev Pos := Nat × Nat × Nat × Nat
def crossPositions (i j k l : Nat) : List Pos :=
 [(Nat.xor i 2,j,k,Nat.xor l 1),
  (Nat.xor i 2,j,Nat.xor k 1,l),
  (i,Nat.xor j 2,k,Nat.xor l 1),
  (i,Nat.xor j 2,Nat.xor k 1,l)]
def allFour : Bool := (List.range 4).all fun i => (List.range 4).all fun j =>
 (List.range 4).all fun k => (List.range 4).all fun l =>
 (crossPositions i j k l).eraseDups.length == 4
theorem every_cross_residual_has_four_positions : allFour = true := by decide
theorem cross_violation_count : 4*4*4*4 = 256 := by decide

-- A completely defined integer 2x2 matrix algebra, with arbitrary entries.
structure M2 where
 a : Int
 b : Int
 c : Int
 d : Int
 deriving DecidableEq
```

```
def mm (x y : M2) : M2 :=
 ⟨x.a*y.a+x.b*y.c, x.a*y.b+x.b*y.d,
   x.c*y.a+x.d*y.c, x.c*y.b+x.d*y.d⟩
def sub (x y : M2) : M2 := ⟨x.a-y.a,x.b-y.b,x.c-y.c,x.d-y.d⟩
def mz : M2 := ⟨0,0,0,0⟩
theorem matrix_assoc (x y z : M2) : mm (mm x y) z = mm x (mm y z) := by
 cases x; cases y; cases z
 simp only [mm, Int.add_mul, Int.mul_add, Int.mul_assoc]
 congr 1 <;> omega
theorem left_right_commute (a b x : M2) : mm a (mm x b) = mm (mm a x) b :=
 (matrix_assoc a x b).symm
-- The left-only commutator vanishes against right multiplication.
theorem split_order_one (h a b x : M2) :
 sub (sub (mm h (mm a (mm x b))) (mm a (mm h (mm x b))))
     (mm (sub (mm h (mm a x)) (mm a (mm h x))) b) = mz := by
 cases h; cases a; cases b; cases x
 simp only [mm, sub, mz, Int.add_mul, Int.mul_add, Int.sub_mul, Int.mul_assoc]
 congr 1 <;> omega

-- Doubling factors; integer witnesses for Pauli sign identities.
def sx : M2 := ⟨0,1,1,0⟩
def sz : M2 := ⟨1,0,0,-1⟩
def mi : M2 := ⟨1,0,0,1⟩
def neg (x : M2) : M2 := ⟨-x.a,-x.b,-x.c,-x.d⟩
theorem sx_square : mm sx sx = mi := by decide
theorem sz_square : mm sz sz = mi := by decide
theorem grading_anticommutes : mm sz sx = neg (mm sx sz) := by decide
theorem plus_real_factor : mm mi sx = mm sx mi := by decide
theorem minus_real_factor : mm sx sx = mi := by decide
theorem wrong_grading_commutation : mm sz sx ≠ mm sx sz := by decide
end Article92
```