ANNALS OF THE CHRYSENE FORMALISM
MACHINE-CHECKED FORMAL PROOF DOSSIER (LEAN 4) | ARTICLE 102 | VOL. 1, NO. 5 | NOVEMBER 2026

Formal Proof Supplement: Article 102

Arithmetic Geometry, Spectral Zeta Theory, and Non-Commutative Geometry of the Chrysene Formalism

Charles D. Schaper, Ph.D.

Lean 4.24.0: twelve checked finite statements. The complete source below compiled successfully with the official Lean 4.24.0 distribution. Every listed theorem has an empty kernel axiom dependency list. No `sorry`, admitted axiom, or external mathematical library is used. This is a partial formalization, not a certification of every theorem in the manuscript.

1 Exact formalized scope

1. Three finite identities establish associativity of addition and multiplication and distributivity for the explicitly encoded four-element operations.
2. The affine point-count theorem checks eight solutions and the arithmetic total $8 + 1 = 9$. It does not formalize projective geometry, smoothness, genus, or the Hasse–Weil theorem.
3. The tangent theorem checks a pointwise identity for field-valued arguments. The polynomial identity and geometric flex interpretation require the ordinary mathematical argument in the manuscript; the Lean statement alone is not a symbolic polynomial identity.
4. The generator Gram matrix is zero. This verifies self-orthogonality of the displayed generators. The passage to the whole linear span and self-duality uses linear algebra and the dimension argument outside this file.
5. Two exhaustive weight bounds plus explicit weight-4 and weight-5 witnesses verify the minimum nonzero weights of the evaluation expressions. The ordinary coding-theory argument identifies these with minimum distances of the linear codes.
6. The recurrence theorem evaluates a specified natural-number recurrence at indices 1–4. It does not identify that recurrence with the spanning-tree count of arbitrary graphs.
7. The extension-tower theorem evaluates a specified formula at degrees 1–4. The Frobenius argument establishing the formula for the curve is outside the formalization.

The finite graph determinants, matching counts, Hashimoto traces, general analytic norm estimates, order-one condition, and all empirical chemistry are outside this Lean file. The companion Python dossier provides independently executed finite checks for the graph and rational-matrix calculations.

2 Reproduction

Save the listing as `FormalProof_A102.lean` and run `lean FormalProof_A102.lean` using Lean 4.24.0. The separate raw source and actual compiler output are included in the publication package. Runtime varies by machine.

Companion Artifact: Schaper, C. D. (2026). Arithmetic Geometry, Spectral Zeta Theory, and Non-Commutative Geometry of the Chrysene Formalism: The (9, 8) Hermitian Curve over GF(4), Genus-1 Goppa Codes, the Ihara–Bass Zeta Function, and Connes' Spectral Triple. *Annals of the Chrysene Formalism*, 1(5), 1966–1974 https://chrysene.com/index.php/acf/article/view/102 • © 2026 Charles D. Schaper.

3 Complete Lean source

```
set_option maxHeartbeats 0
set_option maxRecDepth 100000
/-
  FormalProof_A102.lean
  Machine-Checked Formal Proof Dossier for Article 102:
  "Arithmetic Geometry, Spectral Zeta Theory, and Non-Commutative Geometry
   of the Chrysene Formalism"
  Annals of the Chrysene Formalism, Vol. 1, No. 5, pp. 1966--1974 (2026).

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

namespace ChryseneFormalism.Article102

-- ============================================================================
-- 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

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

```

```
def sq (x : GF4) : GF4 := x * x
def cb (x : GF4) : GF4 := x * x * x

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_distributive (a b c : GF4) : a*(b+c) = a*b+a*c := by
  cases a <;> cases b <;> cases c <;> rfl

-- ======================================================================
-- 2. The (9, 8) Hermitian Curve E_H: y^2 + y = x^3 over GF(4) (Thm 3.2)
-- ======================================================================

def onHermitian (x y : GF4) : Bool :=
  ((sq y) + y) == (cb x)

def ptBit (x y : GF4) : Nat :=
  if onHermitian x y then 1 else 0

def allGF4 : List GF4 := [zero, one, om, om2]

def countAffinePts : Nat :=
  ptBit zero zero + ptBit zero one + ptBit zero om + ptBit zero om2 +
  ptBit one  zero + ptBit one  one + ptBit one  om + ptBit one  om2 +
  ptBit om   zero + ptBit om   one + ptBit om   om + ptBit om   om2 +
  ptBit om2  zero + ptBit om2  one + ptBit om2  om + ptBit om2  om2

-- Theorem 3.2(a): E_H(GF(4)) has exactly 8 affine points and 9 projective points
theorem hermitian_affine_count :
    countAffinePts = 8 /\ (countAffinePts + 1 = 9) := by
  decide

-- Theorem 3.2(b): Every affine point (x0, y0) on E_H(GF(4)) is a flex point (3P = P_infty):
-- Substituting the tangent line y = y0 + x0^2 * (x + x0) into y^2 + y + x^3 yields (x + x0)^3
theorem hermitian_flex_tangent (x0 y0 x : GF4) :
    onHermitian x0 y0 = false \/
    (let y := y0 + (sq x0) * (x + x0)
     (sq y) + y + (cb x) = cb (x + x0)) := by
  cases x0 <;> cases y0 <;> cases x <;> decide

-- ======================================================================
-- 3. Genus-1 Hermitian Goppa Codes C_{Herm,4} = [8,4,4]_4 & C_{Herm,3} = [8,3,5]_4
-- ======================================================================

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

-- Evaluate f(x, y) = c0 + c1*x + c2*y + c3*x^2 at a point (x, y)
def evPt (c0 c1 c2 c3 x y : GF4) : GF4 :=
  c0 + (c1 * x) + (c2 * y) + (c3 * (sq x))

-- Hamming weight across the 8 affine points D = {(0,0),(0,1),(1,om),(1,om2),(om,om),(om,om2),(om2,om),(
    om2,om2)}
```

```
def wtHerm4 (c0 c1 c2 c3 : GF4) : Nat :=
  wtBit (evPt c0 c1 c2 c3 zero zero) +
  wtBit (evPt c0 c1 c2 c3 zero one)  +
  wtBit (evPt c0 c1 c2 c3 one  om)   +
  wtBit (evPt c0 c1 c2 c3 one  om2)  +
  wtBit (evPt c0 c1 c2 c3 om   om)   +
  wtBit (evPt c0 c1 c2 c3 om   om2)  +
  wtBit (evPt c0 c1 c2 c3 om2  om)   +
  wtBit (evPt c0 c1 c2 c3 om2  om2)

-- Euclidean bilinear form <ev_D(f), ev_D(g)> on GF(4)^8
def dotHerm4 (c0 c1 c2 c3 d0 d1 d2 d3 : GF4) : GF4 :=
  ((evPt c0 c1 c2 c3 zero zero) * (evPt d0 d1 d2 d3 zero zero)) +
  ((evPt c0 c1 c2 c3 zero one)  * (evPt d0 d1 d2 d3 zero one))  +
  ((evPt c0 c1 c2 c3 one  om)   * (evPt d0 d1 d2 d3 one  om))   +
  ((evPt c0 c1 c2 c3 one  om2)  * (evPt d0 d1 d2 d3 one  om2))  +
  ((evPt c0 c1 c2 c3 om   om)   * (evPt d0 d1 d2 d3 om   om))   +
  ((evPt c0 c1 c2 c3 om   om2)  * (evPt d0 d1 d2 d3 om   om2))  +
  ((evPt c0 c1 c2 c3 om2  om)   * (evPt d0 d1 d2 d3 om2  om))   +
  ((evPt c0 c1 c2 c3 om2  om2)  * (evPt d0 d1 d2 d3 om2  om2))

-- Generator Gram matrix: finite self-orthogonality certificate.
-- Linear-span self-duality requires dimension and linear algebra outside this file.
def basisCoeff (i : Fin 4) (j : Nat) : GF4 := if i.val == j then one else zero

theorem generator_gram_zero :
    (List.range 4).all (fun i => (List.range 4).all (fun j =>
       dotHerm4 (if i==0 then one else zero) (if i==1 then one else zero)
        (if i==2 then one else zero) (if i==3 then one else zero)
        (if j==0 then one else zero) (if j==1 then one else zero)
        (if j==2 then one else zero) (if j==3 then one else zero) == zero)) = true := by
  decide

-- Theorem 4.2(a.2): C_{Herm,4} = [8, 4, 4]_4 has minimum Hamming distance d_min = 4
theorem c_herm4_min_dist (c0 c1 c2 c3 : GF4) :
    (c0 = zero /\ c1 = zero /\ c2 = zero /\ c3 = zero) \/
    (wtHerm4 c0 c1 c2 c3 >= 4) := by
  cases c0 <;> cases c1 <;> cases c2 <;> cases c3 <;> decide

-- Theorem 4.2(b): C_{Herm,3} = [8, 3, 5]_4 has minimum Hamming distance d_min = 5
theorem c_herm3_min_dist (c0 c1 c2 : GF4) :
    (c0 = zero /\ c1 = zero /\ c2 = zero) \/
    (wtHerm4 c0 c1 c2 zero >= 5) := by
  cases c0 <;> cases c1 <;> cases c2 <;> decide

-- ======================================================================
-- 4. Chebyshev Spanning-Tree Recurrence & Supersingular Tower (Thms 5.1, 6.3)
-- ======================================================================

def kappa : Nat -> Nat
  | 0 => 1
  | 1 => 6
  | n + 2 => 6 * (kappa (n + 1)) - (kappa n)

-- Theorem 5.1(a): Evaluation of the specified numerical recurrence (not a graph universality theorem)
    kappa_4 = 1189
theorem chebyshev_trees_four :
    kappa 1 = 6 /\ kappa 2 = 35 /\ kappa 3 = 204 /\ kappa 4 = 1189 /\ (6 * (kappa 4) = 7134) := by
```

```
  decide

-- Theorem 6.3(a): Supersingular extension tower N_r = (2^r - (-1)^r)^2 for r = 1, 2, 3, 4
def hermExtPts (r : Nat) : Nat :=
  let two_r := 2 ^ r
  let base  := if r % 2 == 1 then two_r + 1 else two_r - 1
  base * base

theorem supersingular_tower :
    hermExtPts 1 = 9 /\ hermExtPts 2 = 9 /\ hermExtPts 3 = 81 /\ hermExtPts 4 = 225 := by
  decide

theorem weight4_witness : wtHerm4 zero one zero one = 4 := by decide

theorem weight5_witness : wtHerm4 zero one one zero = 5 := by decide

end ChryseneFormalism.Article102

-- Kernel Axiom Audit:
#print axioms ChryseneFormalism.Article102.hermitian_affine_count
#print axioms ChryseneFormalism.Article102.hermitian_flex_tangent
#print axioms ChryseneFormalism.Article102.generator_gram_zero
#print axioms ChryseneFormalism.Article102.c_herm4_min_dist
#print axioms ChryseneFormalism.Article102.c_herm3_min_dist
#print axioms ChryseneFormalism.Article102.chebyshev_trees_four
#print axioms ChryseneFormalism.Article102.supersingular_tower

#print axioms ChryseneFormalism.Article102.gf4_add_assoc

#print axioms ChryseneFormalism.Article102.gf4_mul_assoc

#print axioms ChryseneFormalism.Article102.gf4_distributive

#print axioms ChryseneFormalism.Article102.weight4_witness

#print axioms ChryseneFormalism.Article102.weight5_witness
```

4 Actual kernel axiom audit

```
'ChryseneFormalism.Article102.hermitian_affine_count' does not depend on any axioms
'ChryseneFormalism.Article102.hermitian_flex_tangent' does not depend on any axioms
'ChryseneFormalism.Article102.generator_gram_zero' does not depend on any axioms
'ChryseneFormalism.Article102.c_herm4_min_dist' does not depend on any axioms
'ChryseneFormalism.Article102.c_herm3_min_dist' does not depend on any axioms
'ChryseneFormalism.Article102.chebyshev_trees_four' does not depend on any axioms
'ChryseneFormalism.Article102.supersingular_tower' does not depend on any axioms
'ChryseneFormalism.Article102.gf4_add_assoc' does not depend on any axioms
'ChryseneFormalism.Article102.gf4_mul_assoc' does not depend on any axioms
'ChryseneFormalism.Article102.gf4_distributive' does not depend on any axioms
'ChryseneFormalism.Article102.weight4_witness' does not depend on any axioms
'ChryseneFormalism.Article102.weight5_witness' does not depend on any axioms
```