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

Verification Code Companion

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

1 Scope and principal findings

This companion supplies executable exact checks for revised Article 92. The standard-library verifier executes **6,325 explicit predicates** in each of normal and optimized Python modes, with identical outputs, including rejection of ten deliberate false controls. Arithmetic uses integers and explicitly implemented Gaussian integers, without floating-point comparisons.

The inherited verifier fails at its assertion that the cross term has 192 violating basis pairs. The actual count is **256 out of 256**. Its claimed filtration counterexample was not implemented, and its grading block merely emitted constants. The replacement implements these tests, non-real Hermitian examples and a non-scalar represented one-form.

2 Coverage by verification block

U1 Field structure.
All four elements and 64 triples test addition, multiplication, associativity, distributivity, identity and inverses. Companion order, the two gate orders and the nonzero-nilpotent alternative are checked. The cyclic vector and coordinate convention are declared choices.

U2 Dictionary and frames.
Frobenius, trace and norm identities are tested. The corrected dictionary is $A \mapsto 0$, $C \mapsto \omega^2$, $G \mapsto \omega$, $U \mapsto 1$. The prime subfield therefore corresponds to $\{A, U\}$, while the purines form the scalar line $\{0, \omega\}$. On all 64 length-three words, the two frame involutions commute and have fixed counts $(64, 0, 8, 8)$; explicit orbit representatives give 20 orbits independently of the Burnside arithmetic.

U3 Labelled perimeter.
The four primary locants shift by minus one on the stipulated eighteen-cycle and commute with the stated inversion. This is not a graph-automorphism or chemical-transformation test.

U4 Global obstruction.
On a product-trace Pauli monomial basis, a trace-zero site unitary a at distance $n > 0$ sends the vacuum to a degree-n vector and satisfies $a^2 = 1$. Both filtration operators have

$$[[D, L(a)], R(a)]1 = -2n\,1.$$

The script checks this for $1 \le n \le 20$, and separately witnesses failure of the original filtration's J invariance. The general obstruction is proved in the manuscript. Symmetrization repairs J compatibility, not order-one.

U5 Symmetric radius.
For $-20 \le k, m \le 20$, the radius $\max(|k|, |k-m|, |m|)$ is unchanged under $(k, m) \mapsto (k - m, -m)$. The arbitrary three-dimensional support argument, domain result, compact resolvent and bounded-commutator estimate remain proofs in the manuscript.

U6 Internal operators.
Exact sparse matrices represent left and right multiplication on M_4, with all 256 matrix-unit pairs tested. The split $L(H) \pm R(H)$ and nontrivial fluctuations satisfy order-one and the real-structure signs for the selected real and non-real Hermitian examples. The cross term $L(F)R(G)$ fails on all 256 pairs. A nonzero represented one-form is built as $W = a[H, b] + (a[H, b])^*$, rather than merely adding a scalar to H.

U7 Even doubling.
Explicit 32×32 matrices test self-adjointness, grading square and anticommutation, algebra compatibility, real-structure square and signs. For non-real examples, complex conjugation is applied explicitly: $J = PC$ is tested through $P\overline{D} = DP$. The two doubling factors give the stated finite KO sign triples $(+, +, +)$ and $(+, +, -)$.

3 Reproduction and safeguards

Run `python run_verification.py`. It executes `verify_a92.py` in isolated normal and optimized modes and compares both generated files byte for byte. Explicit `check` calls remain enabled under `-O`. The report separates the checks by block and records ten rejected false claims. These controls address the identity, companion order, gate order, trace, orbit count, cross-term count, original J compatibility, global order-one, grading sign and cross-term order-one.

The source and generated macros are pinned by hashes. The standalone LaTeX source extracts the same verifier through `filecontents*`; typesetting does not execute Python.

4 Limits and interpretation

The finite M_4 computations test chosen exact examples; the general complex-Hermitian result follows from the manuscript's left/right multiplication proof. Neither finite examples nor a finite internal real triple establishes order-one for the infinite filtration. The code instead records its obstruction. A labelled perimeter correspondence and unitary encoding do not prove a chemical conversion, a physical gate, biological operation, or compatibility of a proposed molecular coarse-graining.

5 Complete executable source

verify_a92.py

```
#!/usr/bin/env python3
"""Article 92 exact finite verifier. Standard library; checks survive -O.
Finite examples do not certify an infinite real spectral triple or chemistry.
"""
import itertools as it
import json
from collections import Counter
CHECKS=Counter()
def check(ok, block):
    CHECKS[block]+=1
    if not ok: raise AssertionError(block)
V=list(it.product(range(2),repeat=2))
zero=(0,0); one=(1,0)
def add(x,y): return (x[0]^y[0],x[1]^y[1])
def mul(x,y):
    a,b=x; c,d=y
    return ((a*c+b*d)%2,(a*d+b*c+b*d)%2)
def chi(x): return (x[1],x[0]^x[1])
def sq(x): return mul(x,x)
def flip(x): return add(x,one)
def rev(w): return tuple(sq(x) for x in w[::-1])
def fl(w): return tuple(flip(x) for x in w)
for x in V:
    check(mul(x,one)==x and mul(x,zero)==zero,'U1')
    check(x==zero or any(mul(x,y)==one for y in V),'U1')
    check(chi(chi(chi(x)))==x,'U1')
    check(add(add(chi(chi(x)),chi(x)),x)==zero,'U1')
    check(chi(x)==mul((0,1),x),'U1')
    check(sq(chi(x))==chi(x[::-1]),'U2')
    check(chi((x[0]^1,x[1]^1))==flip(chi(x)),'U2')
    check(add(x,sq(x))==(x[1],0),'U2')
    check(mul(x,sq(x))==(int(x!=zero),0),'U2')
for x,y,z in it.product(V,repeat=3):
    check(add(add(x,y),z)==add(x,add(y,z)),'U1')
    check(mul(mul(x,y),z)==mul(x,mul(y,z)),'U1')
    check(mul(x,add(y,z))==add(mul(x,y),mul(x,z)),'U1')
    check(mul(x,y)==mul(y,x),'U1')
    check(sq(mul(x,y))==mul(sq(x),sq(y)),'U2')
C=lambda x:(x[0],x[0]^x[1]); S=lambda x:x[::-1]
for x in V:
    check(C(S(x))==chi(x) and S(C(x))==chi(chi(x)),'U1')
    check(C(C(x))==x and C(x)!=chi(x) if x==(1,0) else C(C(x))==x,'U1')
    check(add(C(add(C(x),x)),add(C(x),x))==zero,'U1')
words=list(it.product(V,repeat=3)); fixed=[0]*4; orbits=set()
for w in words:
    images=(w,fl(w),rev(w),fl(rev(w)))
    for j,t in enumerate(images): fixed[j]+=t==w
    check(fl(fl(w))==w and rev(rev(w))==w,'U2')
    check(fl(rev(w))==rev(fl(w)),'U2')
    orbits.add(min(images))
check(fixed==[64,0,8,8] and len(orbits)==20,'U2')
cycle=['v1','v2','v3','v4','w3','w5','v5','v6','w6',
       'v7','v8','v9','v10','w4','w2','v11','v12','w1']
delta={3:2,6:5,9:8,12:11}; inv={j:((j+5)%12)+1 for j in range(1,13)}
for j,d in delta.items():
    check((cycle.index('v'+str(d))-cycle.index('v'+str(j)))%18==17,'U3')
    check(delta[inv[j]]==inv[d],'U3')
dictionary={'A':(0,0),'C':(0,1),'G':(1,0),'U':(1,1)}
expected={'A':(0,0),'C':(1,1),'G':(0,1),'U':(1,0)}
for base,bits in dictionary.items(): check(chi(bits)==expected[base],'U2')
check({expected[b] for b in 'AU'}=={zero,one},'U2')
check({expected[b] for b in 'AG'}=={zero,(0,1)},'U2')
# Labelled cycle check only, not a carbon-graph automorphism certificate.
```

```
def height(k,m,symmetric):
    return max(abs(k),abs(m),abs(k-m) if symmetric else 0)
for k,m in it.product(range(-20,21),repeat=2):
    check(height(k,m,True)==height(k-m,-m,True),'U5')
    check((k-m)-(-m)==k,'U5')
check(height(1,-1,False)==1 and height(2,1,False)==2,'U4')
# Pauli monomial basis (support,m); site factors square to identity.
def toggle(s,k): return s.symmetric_difference({k})
def left_a(state,k): s,m=state; return toggle(s,k),m
def right_a(state,k): s,m=state; return toggle(s,k+m),m
def degree(state,symmetric):
    s,m=state
    return max([abs(m)]+[abs(k) for k in s]+
               ([abs(k-m) for k in s] if symmetric else []))
for n in range(1,21):
    vac=(frozenset(),0); av=left_a(vac,n); bv=right_a(vac,n)
    check(left_a(bv,n)==right_a(av,n)==vac,'U4')
    for sym in (False,True):
        coefficient=degree(vac,sym)-degree(bv,sym)-degree(av,sym)+degree(vac,sym)
        check(coefficient==-2*n,'U4')
# Sparse exact integer matrices: dimensions are passed for empty matrices.
def plus(A,B):
    C=dict(A)
    for k,v in B.items(): C[k]=C.get(k,0)+v
    return {k:v for k,v in C.items() if v}
def scale(c,A): return {k:c*v for k,v in A.items() if c*v}
def mm(A,B):
    C={}
    rows={}
    for (k,j),b in B.items(): rows.setdefault(k,[]).append((j,b))
    for (i,k),a in A.items():
        for j,b in rows.get(k,[]): C[i,j]=C.get((i,j),0)+a*b
    return {k:v for k,v in C.items() if v}
def comm(A,B): return plus(mm(A,B),scale(-1,mm(B,A)))
def tr(A): return {(j,i):v for (i,j),v in A.items()}
def eye(n): return {(i,i):1 for i in range(n)}
def L(A): return {(4*i+j,4*k+j):v for (i,k),v in A.items() for j in range(4)}
def R(A): return {(4*i+j,4*i+k):v for (k,j),v in A.items() for i in range(4)}
def kron(A,B,n): return {(n*i+k,n*j+l):a*b for (i,j),a in A.items() for (k,l),b in B.items()}
units=[{(i,j):1} for i,j in it.product(range(4),repeat=2)]
F={(i,i^2):1 for i in range(4)}; G={(i,i^1):1 for i in range(4)}
H=plus(F,scale(2,G)); K={(0,0):1,(1,1):-2,(0,3):2,(3,0):2}
J={(4*i+j,4*j+i):1 for i,j in it.product(range(4),repeat=2)}
check(mm(J,J)==eye(16),'U6')
for A in units:
    check(mm(mm(J,L(tr(A))),J)==R(A),'U6')
for A,B in it.product(units,repeat=2):
    check(comm(L(A),R(B))=={},'U6')
counts={}
for name,T in [('H',H),('fluctuated',plus(H,K))]:
    for sign in (1,-1):
        D=plus(L(T),scale(sign,R(T)))
        check(tr(D)==D,'U6')
        check(mm(mm(J,D),J)==scale(sign,D),'U6')
        bad=0
        for A,B in it.product(units,repeat=2):
            residual=comm(comm(D,L(A)),R(B))
            check(residual=={},'U6'); bad+=bool(residual)
        counts[name+str(sign)]=bad
Dcross=mm(L(F),R(G)); bad=0
for A,B in it.product(units,repeat=2):
    residual=comm(comm(Dcross,L(A)),R(B)); bad+=bool(residual)
    check(residual==mm(L(comm(F,A)),R(plus(mm(B,G),scale(-1,mm(G,B))))),'U6')
check(bad==256,'U6')
# A nonzero represented one-form: a[H,b], symmetrized.
a=units[0]; b=units[3]; W=mm(a,comm(H,b)); W=plus(W,tr(W))
check(W!={} and tr(W)==W,'U6')
Dform=plus(L(plus(H,W)),R(plus(H,W)))
```

```
for A,B in it.product(units,repeat=2):
    check(comm(comm(Dform,L(A)),R(B))=={},'U6')
# J is anti-linear; real coefficient matrices permit exact integer tests.
sx={(0,1):1,(1,0):1}; sz={(0,0):1,(1,1):-1}; I2=eye(2)
D=plus(L(H),R(H)); Dt=kron(D,sx,2); Gamma=kron(eye(16),sz,2)
check(tr(Dt)==Dt and tr(Gamma)==Gamma,'U7')
check(mm(Gamma,Gamma)==eye(32),'U7')
check(plus(mm(Gamma,Dt),mm(Dt,Gamma))=={},'U7')
for sign,B in ((1,I2),(-1,sx)):
    Jt=kron(J,B,2)
    check(mm(Jt,Jt)==eye(32),'U7')
    check(mm(Jt,Dt)==mm(Dt,Jt),'U7')
    check(mm(Jt,Gamma)==scale(sign,mm(Gamma,Jt)),'U7')
    for A in units: check(comm(Gamma,kron(L(A),I2,2))=={},'U7')
# Gaussian integers give exact non-real Hermitian examples, without floats.
class Gaussian:
    def __init__(self,a=0,b=0): self.a,self.b=a,b
    @staticmethod
    def lift(v): return v if isinstance(v,Gaussian) else Gaussian(v)
    def __add__(self,v):
        v=self.lift(v); return Gaussian(self.a+v.a,self.b+v.b)
    __radd__=__add__
    def __mul__(self,v):
        v=self.lift(v)
        return Gaussian(self.a*v.a-self.b*v.b,self.a*v.b+self.b*v.a)
    __rmul__=__mul__
    def __eq__(self,v):
        v=self.lift(v); return self.a==v.a and self.b==v.b
    def __bool__(self): return bool(self.a or self.b)
    def conj(self): return Gaussian(self.a,-self.b)
def conjugate(A):
    return {k:(v.conj() if isinstance(v,Gaussian) else v) for k,v in A.items()}
Hc=plus(H,{(0,1):Gaussian(0,1),(1,0):Gaussian(0,-1)})
check(tr(conjugate(Hc))==Hc,'U6_complex')
for sign in (1,-1):
    Dc=plus(L(Hc),scale(sign,R(Hc)))
    check(tr(conjugate(Dc))==Dc,'U6_complex')
    check(mm(mm(J,conjugate(Dc)),J)==scale(sign,Dc),'U6_complex')
    for A,B in it.product(units,repeat=2):
        check(comm(comm(Dc,L(A)),R(B))=={},'U6_complex')
Dct=kron(plus(L(Hc),R(Hc)),sx,2)
for sign,B in ((1,I2),(-1,sx)):
    Jt=kron(J,B,2)
    check(mm(Jt,conjugate(Dct))==mm(Dct,Jt),'U7_complex')
    check(mm(Jt,conjugate(Gamma))==scale(sign,mm(Gamma,Jt)),'U7_complex')
# Negative controls are intentionally false claims, explicitly rejected.
controls={
 'wrong_identity':all(mul(x,(0,1))==x for x in V),
 'wrong_companion_order':all(chi(chi(x))==x for x in V),
 'wrong_gate_order':all(S(C(x))==chi(x) for x in V),
 'wrong_trace':all(add(x,sq(x))==(x[0],0) for x in V),
 'wrong_orbits':len(orbits)==16,
 'wrong_cross_count':bad==192,
 'uncalibrated_J_invariance':height(1,-1,False)==height(2,1,False),
 'global_sym_order_one':-2==0,
 'wrong_grading_sign':mm(Gamma,Dt)==mm(Dt,Gamma),
 'cross_term_order_one':all(comm(comm(Dcross,L(A)),R(B))=={} for A,B in it.product(units,repeat=2))}
for name,claim in controls.items(): check(not claim,'negative_controls')
macros={'valAlphabetSize':4,'valMChiOrder':3,'valMChiPrimeOrder':2,
 'valFieldTriples':64,'valNonZeroUnits':3,'valCodonCount':64,
 'valBurnsideId':64,'valBurnsideFlip':0,'valBurnsideRev':8,
 'valBurnsideFlipRev':8,'valFrameClassesSix':20,'valPerimLength':18,
 'valPriLocantCount':4,'valDualLocantCount':4,'valCyclicShiftStep':-1,
 'valInternalDim':16,'valBasisPairs':256,'valOrderOneViolationsCross':bad,
 'valOrderOneViolationsSplit':0,'valOrderOneViolationsFluct':0,
 'valGradedDim':32,'valKOZero':0,'valKOSix':6}
with open('verification_values_a92.tex','w') as f:
    for k,v in macros.items(): f.write('\\providecommand{\\'+k+'}{'+str(v)+'}\n')
```

```
report={'checks':sum(CHECKS.values()),'by_block':dict(CHECKS),
 'negative_controls_rejected':len(controls),'fixed_points':fixed,
 'orbits':len(orbits),'cross_violations':bad,'split_violations':counts,
 'scope':'Exact finite field, cycle, bounded support and integer and Gaussian-integer matrix examples;
     not global order-one, chemistry, or all complex operators.'}
with open('verification_report.json','w') as f: json.dump(report,f,indent=2)
print(json.dumps(report,sort_keys=True))
```