log_deriv

July 11, 2025

The manuscript "SLVer Bullet: Straight-Line Verification for Bulletproofs" by Goodell, Salazar, Slaughter, Szramowski contains a modification of the verification equations obtained by Eagen in the manuscript "Zero Knowledge Proofs of Elliptic Curve Inner Products from Principal Divisors and Weil Reciprocity". Below is a modification of a SAGE implementation by Eagen in order to check these equations on particular examples. There also seems to be an additional 2y factor in the terms in SLVer Bullet. We also provide the corresponding values for these modified expressions.

[1]:
```
# Initialize an elliptic curve
p=103
Fp = GF(p)  # Base Field
A = 0
B = 7
E = EllipticCurve(GF(p), [A,B])
```

[2]:
```
K.<x> = Fp[]
L.<y> = K[]
eqn = y^2 - x^3 - A * x - B
```

[3]:
```
# Returns line passing through points, works for all points and returns 1 for O␣
↪+ O = O
def line(A, B):
    if A == 0 and B == 0:
        return 1
    else:
        [a, b, c] = Matrix([A, B, -(A+B)]).transpose().kernel().basis()[0]
        return a*x + b*y + c
```

[4]:
```
# Works for A == B but not A == -A, as the line has no slope or intercept
def slope_intercept(A, B):
    [a, b, c] = Matrix([A, B, -(A+B)]).transpose().kernel().basis()[0]
    return (-a/b, -c/b)
```

[5]:
```
# Fails at O
def eval_point(f, P):
    (x, y) = P.xy()
    return f(x=x, y=y)
```

[6]:
```
# f(x) + y g(x) -> (f(x), g(x)), should reduce mod eqn first
def get_polys(D):
    return ( K(D(y=0)), K(D(y=1) - D(y=0)) )
```

[7]:
```
# Accepts arbitrary list of points, including duplicates and inverses, and constructs function
# intersecting exactly those points if they form a principal divisor (i.e. sum to zero).
def construct_function(Ps):
    # List of intermediate sums/principal divisors, removes 0
    xs = [(P, line(P, -P)) for P in Ps if P != 0]

    while len(xs) != 1:
        assert(sum(P for (P, _) in xs) == 0)
        xs2 = []

        # Carry extra point forward
        if mod(len(xs), 2) == 1:
            x0 = xs[0]
            xs = xs[1:]
        else:
            x0 = None

        # Combine the functions for all pairs
        for n in range(0, floor(len(xs)/2)):
            (A, aNum) = xs[2*n]
            (B, bNum) = xs[2*n+1]

            # Divide out intermediate (P, -P) factors
            num = L((aNum * bNum * line(A, B)).mod(eqn))
            den = line(A, -A) * line(B, -B)
            D = num / K(den)

            # Add new element
            xs2.append((A+B, D))

        if x0 != None:
            xs2.append(x0)

        xs = xs2

    assert(xs[0][0] == 0)

    # Normalize, might fail but negl probability for random points. Must be done for zkps
    # although free to use any coefficient
    return D / D(x=0, y=0)
```

[8]:
```
# Random principal divisor with n points
def random_principal(n):
    # For general elliptic curves, might want to clear cofactor depending on␣
 ↪application
    # Works for arbitrary curve groups
    Ps = [E.random_element() for _ in range(0, n-1)]
    Ps.append(-sum(Ps))
    return Ps
```

[9]:
```
N=5
DIV = random_principal(N)  #create a random principal divisor with N elements␣
 ↪in its support
D = construct_function(DIV)
(a,b)=get_polys(D)
```

[10]:
```
Dx = D.differentiate(x)
Dy = D.differentiate(y)
Dz = Dx + Dy * ((3*x^2 + A) / (2*y))
```

[11]:
```
#Generate random challenge points, compute slope lam and intercept cons
[A0, A1] = [E.random_element() for _ in range(0, 2)]
(lam,cons)=slope_intercept(A0,A1)
A2 = -(A0 + A1)
```

[12]:
```
#the common expression for the numerator appearing in Eagen and for the terms␣
 ↪c1 and c3 in SLVer Bullet
NUM=2*y*Dx+(3*x^2+A)*b
#the common expression for the numerator appearing in Eagen and for the terms␣
 ↪c1 and c3 in SLVer Bullet
DENOM=D*(3*x^2+A-lam*2*y)

print("Value of expression for the logarithmic derivative of L(div f):")
print(sum(eval_point(1/(cons-(y-lam*x)), P) for P in DIV))

print("Value of expression derived by Eagen for the logarithmic derivative of␣
 ↪the norm:")
print(sum(eval_point(NUM, P)/eval_point(DENOM, P) for P in [A0, A1, A2]))

#c2, b3, c4, b4 according to SLVer Bullet with 2y factor in the denominator␣
 ↪removed
c2=-(2*lam*A0[1]*(2*A2[1]*eval_point(Dx, A2)+(3*A2[0]^2+A)*eval_point(b,␣
 ↪A2))*(3*A0[0]^2+A-2*A0[1]*(lam+A1[0]-A0[0])))
b2=A2[1]*eval_point(D, A2)*(3*A0[0]^2+A-2*A0[1]*lam)*(A1[0]-A0[0])
c4=(2*lam*A1[1]*(2*A2[1]*eval_point(Dx, A2)+(3*A2[0]^2+A)*eval_point(b,␣
 ↪A2))*(3*A1[1]^2+A-2*A1[1]*(lam+A1[0]-A0[0])))
b4=A2[1]*eval_point(D, A2)*(3*A1[0]^2+A-2*A1[1]*lam)*(A1[0]-A0[0])
```

```
print("Value of expression derived in SLVer Bullet for the logarithmic␣
 ↪derivative of the norm (with extra 2y factors in denominator removed):")
print(c2/b2+c4/b4+sum(eval_point(NUM, P)/eval_point(DENOM, P) for P in [A0,␣
 ↪A1]))

#expressions given in SLVer Bullet without correcting the denominators
NUMp=2*y*Dx+(3*x^2+A)*b
DENOMp=2*y*D*(3*x^2+A-lam*2*y) #here 2y is added
c2p=-(2*lam*A0[1]*(2*A2[1]*eval_point(Dx, A2)+(3*A2[0]^2+A)*eval_point(b,␣
 ↪A2))*(3*A0[0]^2+A-2*A0[1]*(lam+A1[0]-A0[0])))
b2p=2*A0[1]*A2[1]*eval_point(D, A2)*(3*A0[0]^2+A-2*A0[1]*lam)*(A1[0]-A0[0])␣
 ↪#here 2y is added
c4p=(2*lam*A1[1]*(2*A2[1]*eval_point(Dx, A2)+(3*A2[0]^2+A)*eval_point(b,␣
 ↪A2))*(3*A1[1]^2+A-2*A1[1]*(lam+A1[0]-A0[0])))
b4p=2*A1[1]*A2[1]*eval_point(D, A2)*(3*A1[0]^2+A-2*A1[1]*lam)*(A1[0]-A0[0])␣
 ↪#here 2y is added
print("Value of expression derived in SLVer Bullet for the logarithmic␣
 ↪derivative of the norm as given in the manuscript:")
print(c2p/b2p+c4p/b4p+sum(eval_point(NUMp, P)/eval_point(DENOMp, P) for P in␣
 ↪[A0, A1]))
```

```
Value of expression for the logarithmic derivative of L(div f):
40
Value of expression derived by Eagen for the logarithmic derivative of the norm:
40
Value of expression derived in SLVer Bullet for the logarithmic derivative of
the norm (with extra 2y factors in denominator removed):
82
Value of expression derived in SLVer Bullet for the logarithmic derivative of
the norm as given in the manuscript:
81
```

[ ]: