MimbleWimble

Prof Bill Buchanan, The Cyber Academy

http://asecuritysite.com

CYBER ACADEMY

Tom Elvis Jedusor

```
MIMBLEWIMBLE
Tom Elvis Jedusor
19 July, 2016

\****/
Introduction
/****\

Bitcoin is the first widely used financial system for which all the necessary
data to validate the system status can be cryptographically verified by anyone.
However, it accomplishes this feat by storing all transactions in a public
database called "the blockchain" and someone who genuinely wishes to check
this state must download the whole thing and basically replay each transaction,
check each one as they go. Meanwhile, most of these transactions have not
affected the actual final state (they create outputs that are destroyed
a transaction later).
```

Github

Ignotus Peverell

ignopeverell

Block or report user

Sign in to view email

Overview Repositories 5 Stars 2 Followers 135 Following 0

Popular repositories

grin
Forked from mimblewimble/grin

Minimal implementation of the MimbleWimble protocol.

Rust ★ 1

site-test

Test repository for a slightly more elaborate Grin Jekyll site

HTML ★ 1 2

hacker
Forked from pages-themes/hacker

Hacker is a Jekyll theme for GitHub Pages

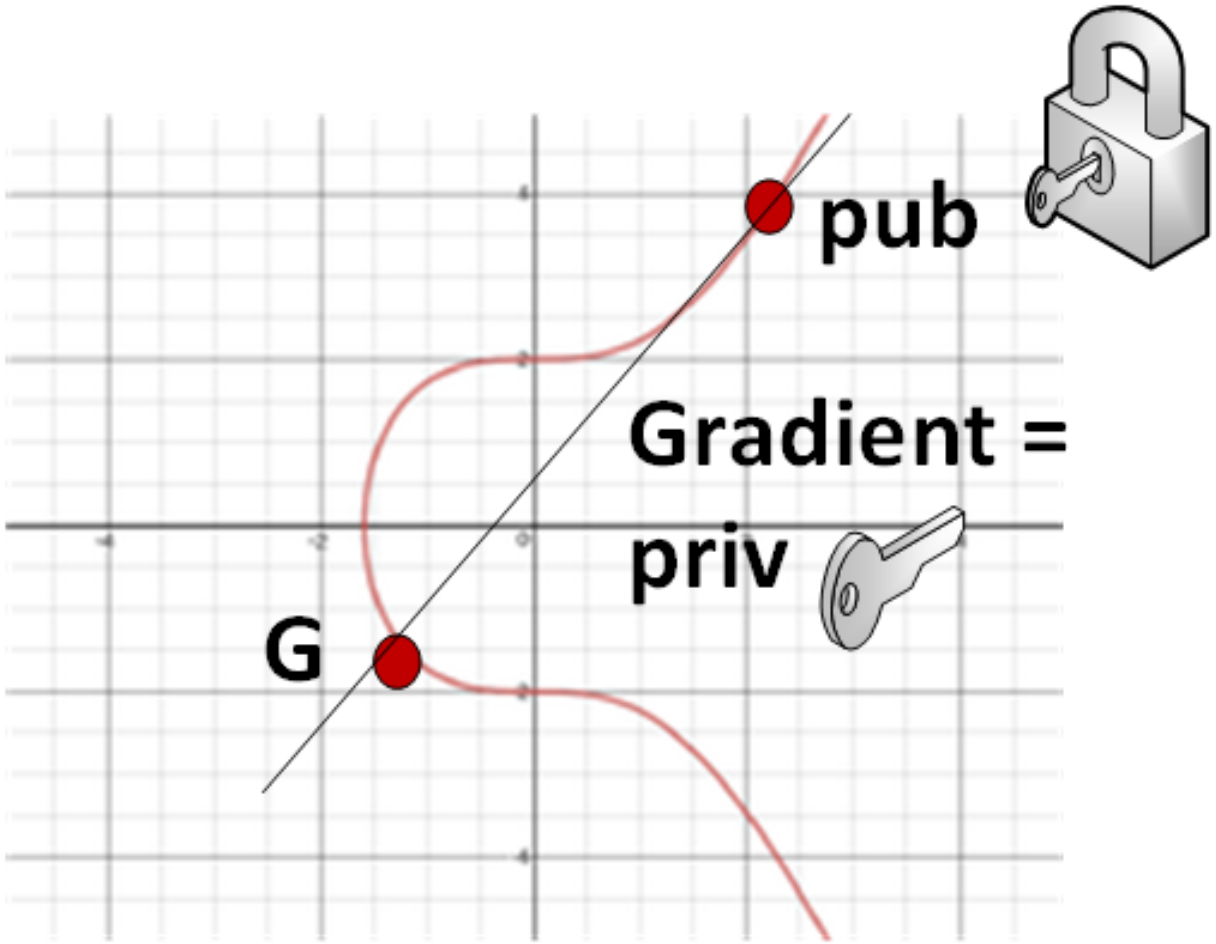
CSS 1

enumset
Forked from Lymia/enumset

A library for defining enums that can be used in compact bit sets.

Rust

Public and private keys with ECC



Private key:
0xc9f4f55bdeb5ba0bd337f2dbc952a5439e20ef9af6203d25d014e7102d86aaeeL
Public key:
0xc44370819cb3b7b57b2aa7edf550a9a5410c234d27aff497458bbbfec8b6a327,
0x52a1a3e222cd89cbd2764b69bd9b0ea5c4fd6ca28861e1f2140eeff9c2e76487

G:
(50662630222773436695787188951685343262506034537775941755001873603891167292 40L,
32670510020758816978083085130507043184471273380659243275938904335757337482424L)

Public and private keys with ECC

```
curve = EllipticCurve(
    'secp256k1',
    # Field characteristic.
    p=0xfffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffefffffc2f,
    # Curve coefficients.
    a=0,
    b=7,
    # Base point.
    g=(0x79be667ef9dcbbac55a06295ce870b07029bfcdb2dce28d959f2815b16f81798,
       0x483ada7726a3c4655da4fbfc0e1108a8fd17b448a68554199c47d08ffb10d4b8),
    # Subgroup order.
    n=0xfffffffffffffffffffffffffffffffebaaedce6af48a03bbfd25e8cd0364141,
    # Subgroup cofactor.
    h=1,
)
```

52a5439e20ef9a

50a9a5410c234

d9b0ea5c4fd6ca

G:
(50662630222773436695787188951685343262506034537775941755001873603891167292 40L,
32670510020758816978083085130507043184471273380659243275938904335757337482424L)

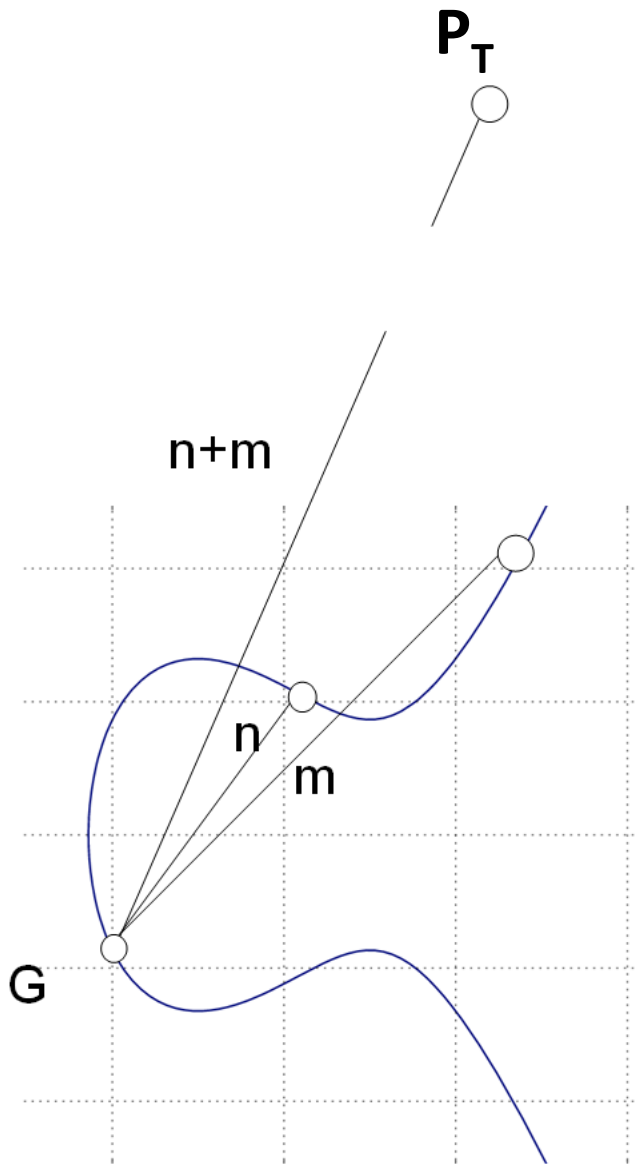
Public and private keys with ECC

```
curve = EllipticCurve(
    'secp256k1',
    # Field characteristic.
    p=0xfffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffefffffc2f,
    # Curve coefficients.
    a=0,
    b=7,
    # Base point.
    g=(0x79be667ef9dcbbac55a06295ce870b07029bfcdb2dce28d959f2815b16f81798,
       0x483ada7
    # Subgroup o
    n=0xffffffff
    # Subgroup c
    h=1,
)
```

52a5439e20ef9a

```
def is_on_curve(point):
    """Returns True if the given point lies on the elliptic curve."""
    if point is None:
        # None represents the point at infinity.
        return True

    x, y = point

    return (y * y - x * x * x - curve.a * x - curve.b) % curve.p == 0
```

Point addition in ECC



$P_1 = n\,G$

$P_2 = m\,G$

$P_T = nG + mG = (n+m)\,G$

```
aliceSecretKey, alicePublicKey = make_keypair()
bobSecretKey, bobPublicKey = make_keypair()


print "\nAlice\'s secret key:\t", aliceSecretKey
print "Alice\'s public key:\t", alicePublicKey
print "Bob\'s secret key:\t", bobSecretKey
print "Bob\'s public key:\t", bobPublicKey

print "\n=========Now we compare if same ================"


sharedSecret = scalar_mult(bobSecretKey+aliceSecretKey,curve.g)


sharedPublic = point_add(alicePublicKey,bobPublicKey)

print "(Alice Private+Bob Private)*G",sharedSecret
print "Alice Public+Bob Public",sharedPublic

print "\nNow let's compare..."
if (sharedSecret[0]==sharedPublic[0]):
        print "Success!"
else:
        print "Failure!"
```

$$P_1 = n\,G$$

$$P_2 = m\,G$$

$$P_T = nG + mG = (n+m)\,G$$

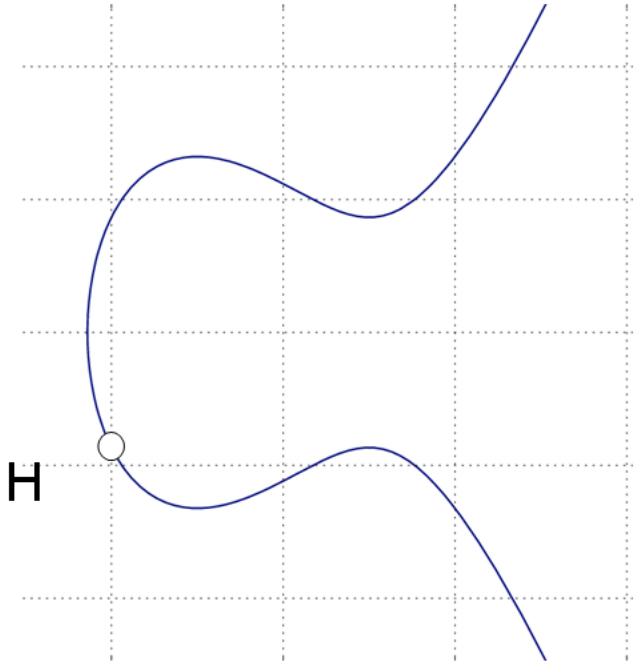
```
Basepoint G:    (55066263022277343669578718895168534326250603453777594175500187360389116729240L, 32670510020758816978083085130507043184
471273380659243275938904335757337482424L)

Alice's secret key:     43117620402230955476965528669042438660394005541196203406020707489909525604940
Alice's public key:     (40196430737305660836176506472910673282156981607990481027922846579400699488 02L, 70465223976866971083435294728 44
5479148035854275287211179449856163787341254166L)

Bob's secret key:       11394682118866760477924072570938370526460252973828812888633400359419567357551 2
Bob's public key:       (10043534705938086153423427897730637068495497915871538814283052494724877041 7046L, 94160905864433229108459971809
35613795023706765768139657670252247446419646048 4L)

=========Now we compare if same =================
(Alice Private+Bob Private)*G (76993010474774159598056594977435333329474333327636586317573761552043530263605L, 75175255398933451677789 8
25153696420983321174467684797211715435808493620129877L)
Alice Public+Bob Public (76993010474774159598056594977435333329474333327636586317573761552043530263605L, 75175255398933451677789825153 6
96420983321174467684797211715435808493620129877L)
```

$P_T = nG + mG = (n+m)\,G$

```
sharedPublic = point_add(alicePublicKey,bobPublicKey)

print "(Alice Private+Bob Private)*G",sharedSecret
print "Alice Public+Bob Public",sharedPublic

print "\nNow let's compare..."
if (sharedSecret[0]==sharedPublic[0]):
        print "Success!"
else:
        print "Failure!"
```

Hiding Transaction Values

Transaction values: v_1 v_2 v_3 - - - - - - - - - - - - →

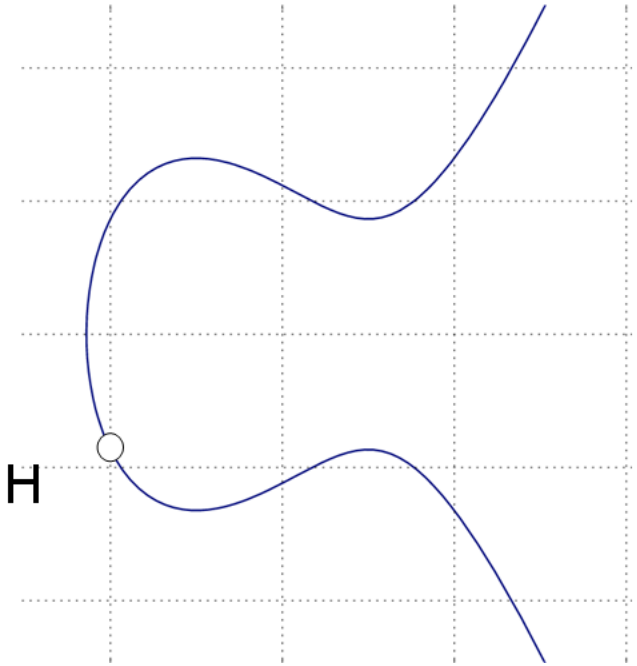


$$T = (v_1 + v_2 + v_3)\, H$$

Eve would be able to determine the transaction values over time

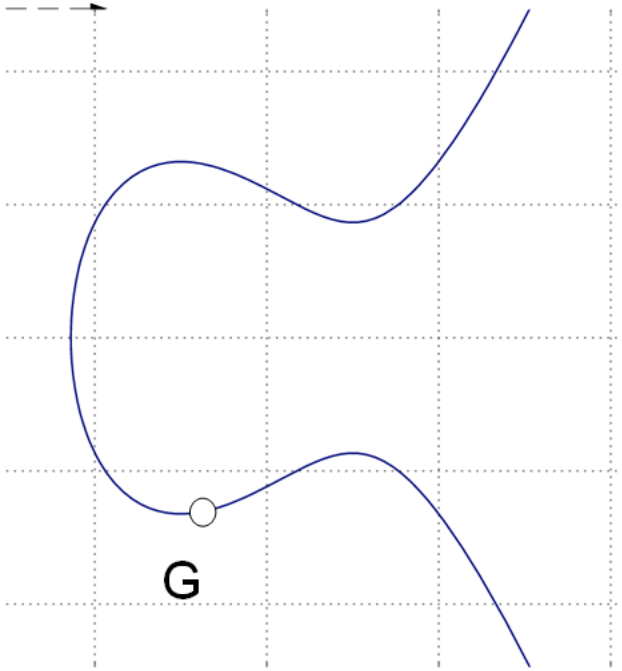
Blinding factor

Transaction values: v_1 v_2 v_3



$$T_1 = v_1 H + r_1 G$$

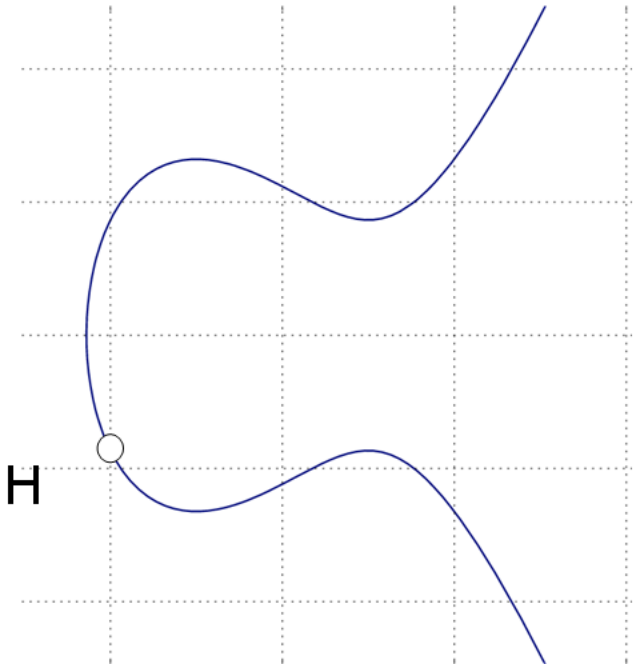
Blinding factor



Blinding factor

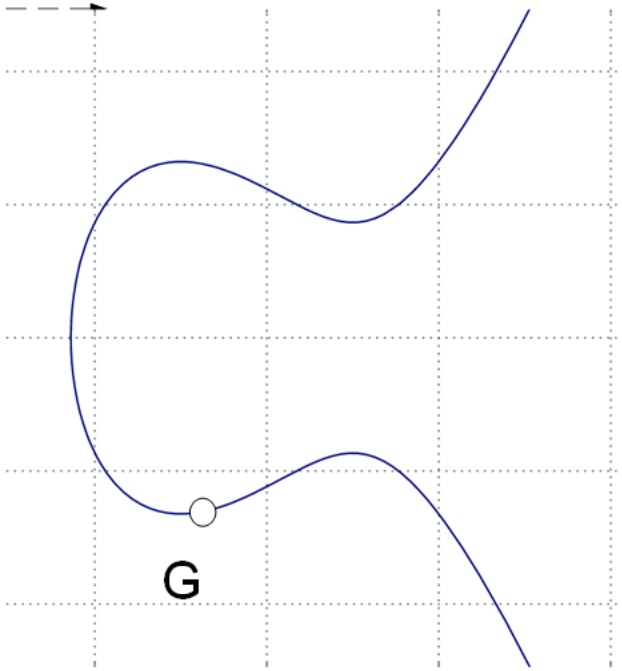
```
(ri1*G + vi1*H) + (ri2*G + vi2*H) = (ro3*G + vo3*H)
```

Transaction values: v_1 v_2 v_3



$$T_1 = v_1 H + r_1 G$$

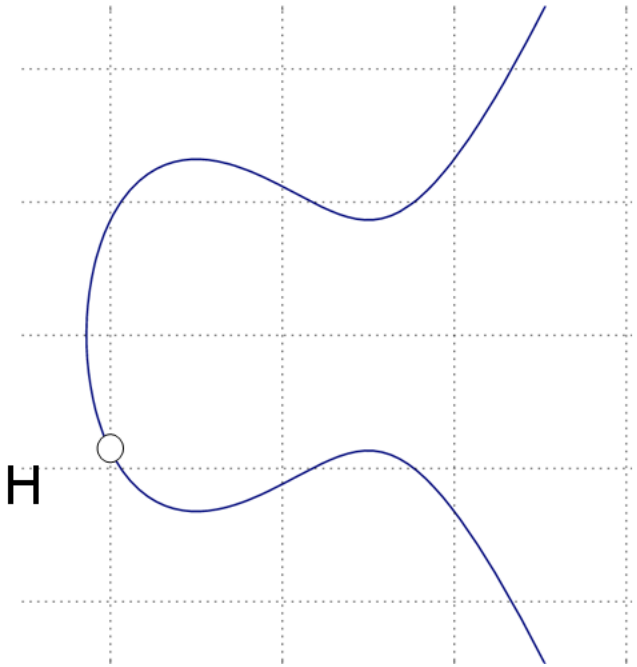
Blinding factor



Blinding factor

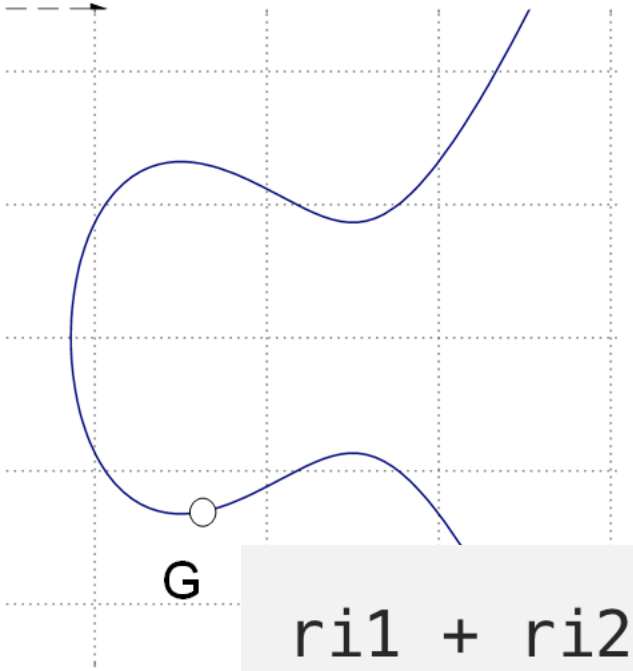
```
(ri1*G + vi1*H) + (ri2*G + vi2*H) = (ro3*G + vo3*H)
```

Transaction values: v_1 v_2 v_3



$$T_1 = v_1 H + r_1 G$$

Blinding factor




```
ri1 + ri2 = ro3
```

Blinding factor

(ri1*G +

Transac

H

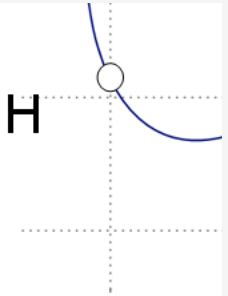
```
r3=r1+r2
v3=v1+v2

aliceSecretKey, alicePublicKey = make_keypair()
bobSecretKey, bobPublicKey = make_keypair()


print "\nAlice\'s secret key:\t", aliceSecretKey
print "Alice\'s public key:\t", alicePublicKey

print "\n========================="

va = point_add(scalar_mult(r1*bobSecretKey,curve.g),scalar_mult(v1*bobSecretKey ,curve.g))
vb = point_add(scalar_mult(r2*bobSecretKey ,curve.g),scalar_mult(v2*bobSecretKey ,curve.g))
vr1 = point_add(va,vb)

print "Transaction (r1*G + v1*G) + (r2*G +v2*G): ",vr1

vr2 = point_add(scalar_mult(r3*bobSecretKey ,curve.g),scalar_mult(v3*bobSecretKey,curve.g))
print "Transaction (r3*G + v3*G): ",vr2

print "\nNow let's compare..."
if (vr1[0]==vr2[0]):
        print "Success!"
else:
        print "Failure!"
```

Blinding factor

```
(ri1*G +
```

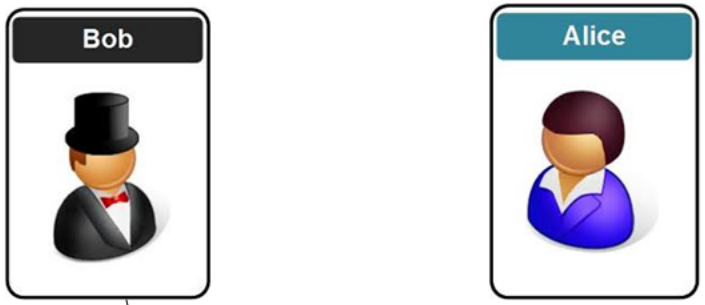
```
r3=r1+r2
v3=v1+v2

aliceSecretKey, alicePublicKey = make_keypair()
bobSecretKey, bobPublicKey = make_keypair()


print "\nAlice\'s secret key:\t", aliceSecretKey
```

```
Alice's secret key:     110672200165973757670706606013807185435784586203078289927107093258368544244209
Alice's public key:     (97877912897005737218928527353789673381377340700475699151216756276689766435015L, 87838438629927847912046825769741818157159420065666269753906778707978168856075L)

==========================
Transaction (r1*G + v1*G) + (r2*G +v2*G):  (75012975245730542912551430086949413769034689023115807054516566426350436876319L, 42840948626806857828893112378852997190784128370121817339191672098791182051629L)
Transaction (r3*G + v3*G):  (75012975245730542912551430086949413769034689023115807054516566426350436876319L, 42840948626806857828893112378852997190784128370121817339191672098791182051629L)

Now let's compare...
Success!
```




```
print "Transaction (r3*G + v3*G): ",vr2

print "\nNow let's compare..."
if (vr1[0]==vr2[0]):
        print "Success!"
else:
        print "Failure!"
```

Proving No New Currency



r=10

Pay Alice 4 BTC

Blockchain

T = 4*G + 10*H

Only Bob knows 10.
Bob and Alice know 4.

Now we sent to Trent

Bob can prove -> X = 4*G + 10*H - 4*G + 10*H => 0

Signing Transaction

r=10

Pay Alice 4 BTC

Blockchain

T = 4*G + 10*H

Only Bob knows 10.
Bob and Alice know 4.

Now we sent to Trent

Bob can prove -> X = 4*G + 10*H - 4*G + 10*H => 0

Trent has a private key of 11

Y - Xi = (11*G + 10*H) - (4*G + 10*H) = 8*G

Valid public key to prove Y-Xi

Sum of blinding factors

MimbleWimble

Prof Bill Buchanan, The Cyber Academy

http://asecuritysite.com

CYBER ACADEMY