Suppose I have committed to 3 large (230+ bits), high-entropy values a, b, c

by mapping them to points on the elliptic curve g

with generator G

and order q

. So, the commitments are A = a \cdot G

, B = b \cdot G

, C = c \cdot G

.

The scheme below can be used to prove, very efficiently, that a = b + c

in Z

. I'm wondering if there are any holes in it - so, would appreciate any feedback.

Commitments over two elliptic curves

A naive way to check this equality would be to check if A = B + C

. However, this would not be in Z

since the addition can underflow or overflow. To address this, we could use range proofs, but these are large and inefficient. So, we define a new commitment scheme over two elliptic curves:

Select another elliptic curve g'

with generator G'

and order q'

, such that q < q'

. Then, compute a commitment to a

as follows:

1. Map a

to both curves as A = a \cdot G

and A' = a \cdot G'

.

1. Choose a random value r

and compute T = r \cdot G

and T' = r \cdot G'

.

1. Compute u = H(G, G', A, A', T, T')

, where H

is a cryptographic hash function.

1. Compute v = r + a \cdot u

(this is done in Z

), such that \frac{v}{u} < q

. This may require iterating through several values of r

before a suitable v

is found.

1. The commitment is then defined as a 5-tuple (A, A', T, T', v)

.

Using this commitment anyone can verify that A

and A'

are derived from the same value a

, and that a < q

. The verification can be done as follows:

1. Compute u = H(G, G', A, A', T, T')

.

1. Check that v \cdot G = T + u \cdot A

and v \cdot G' = T' + u \cdot A'

. This check uses proof of equality of discrete logarithms to ensure that A

and A'

are derived from the same value a

.

1. Check that \frac{v}{u} < q

. This ensures that a < q

because \frac{v}{u} = \frac{(r + a \cdot u)}{u} = \frac{r}{u} + a

and, thus, \frac{r}{u} + a

can be less than q

only if a < q

The commitment is very compact. Assuming we use 256-bit elliptic curves, points A

, A'

, T

, T'

can be encoded in ≈32 bytes each. To ensure that the commitment doesn’t leak any information about the magnitude of a

, the random value r

should be close to 64 bytes. Thus, the total size of the commitment is 192 bytes.

Proof of addition

Using the commitments defined in the previous section we can prove that a = b + c

without revealing values of a

, b

, or c

as follows:

Assuming (A, A', T_1, T_1', v_1)

, (B, B', T_2, T_2', v_2)

, (C, C', T_3, T_3', v_3)

are valid commitments for a

, b

, and c

respectively, a = b + c

if and only if:

1. A = B + C

and

1. A' = B' + C'

.

Checking the equality on both elliptic curves at the same time is required to ensure that the addition does not underflow or overflow.

Where this can be useful

Suppose we define a coin as 10^{69}

indivisible units (similar to how, for example, a single bitcoin is defined as 10^8

satoshi). Under this definition, it takes 230 bits to encode a single coin but it also allows us to represent all amounts as unique values. By randomizing the lower 200 bits of an amount, we can guarantee, with extremely high degree of probability, that the same amount will not appear in the ledger more than once. For example, an amount of 2 coins may appear as 2.000000000090324324320… one time, and as 1.99999999970934309280… another time. However, from the end-user perspective, both amounts are identical and would appear as just 2 coins in wallet software.

This technique would allow us to use the scheme above to effectively hide all transaction amounts in the blockchain.