Demystifying Zero-Knowledge Proofs

Elena Nadolinski
@leanthebean
elena@beanstalk.network

Beanstalk

Who is this workshop for:

- No prior knowledge necessary
- Some experience with Solidity smart contracts
- Basic math

Where to get these slides & links

Twitter: @leanthebean

What we’ll go over

Part 1 - theory

- Some background knowledge
- Overview of how ZKPs work

Part 2 - coding

- Use Zokrates to make a smart contract that will generate NFT tokens only if the correct proof to a puzzle is given

Coding Part:

What we’ll go over

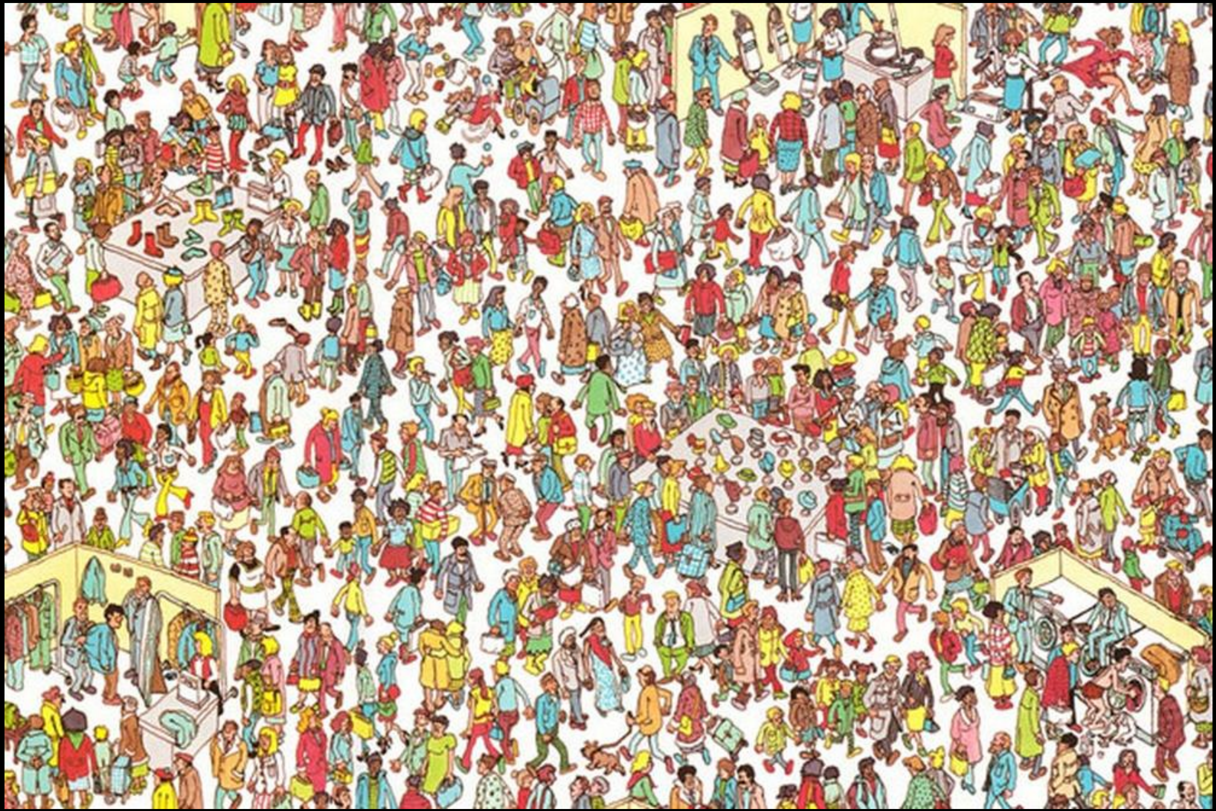
Part 2 - coding

- https://github.com/leanthebean/puzzle-hunt
- Install Docker & Zokrates

What is a ZKP?

What is a ZKP?

The ability to prove honest computation without revealing inputs



ZKPs ≠ privacy

SNARKs
STARKs
Bulletproofs

ZKPs ==

honest computation

$$f(x) = y$$

+ proof

$$f(x) = y$$

- Merkle path to my commitment notes that cover my transaction
- Entire Blockchain

$$f(x) = y$$

- Change to a smart contract
    - One machine does the computation, all others accept new state change with a proof

History

- 1985 "The Knowledge Complexity of Interactive Proof-Systems"
- Shafi Goldwasser, Silvio Micali, and Charles Rackoff

- http://web.mit.edu/~ezyang/Public/graph/svg.html

	Proof Size	Prover Time	Verification Time
SNARKs (has trusted setup)			
STARKs			
Bulletproofs			

	Proof Size	Prover Time	Verification Time
SNARKs (has trusted setup)	288 bytes	2.3s	10ms
STARKs	45KB-200KB	1.6s	16ms
Bulletproofs	~1.3KB	30s	1100ms

Projects using ZKPs

SNARKs

Zcash: has optional **shielded transactions** that use zk-SNARKs

Has a ceremony for the trusted setup per circuit upgrade

Projects using ZKPs

SNARKs

CODA

Use **recursive SNARKs** to give a merkle root of latest global state with a proof

New nodes do not have to sync from the genesis block to build up their in-memory global state

Projects using ZKPs

Bulletproofs

Use Bulletproofs for more efficient **range proofs only** and **<u>not for privacy directly</u>**

Projects using ZKPs

zk-STARKs

Batching transactions off-chain for a decentralized exchange (DEX)

Open Source Projects

SNARKs	Zokrates a great SNARK domain specific language (DSL) for generating proofs and validating them on Ethereum Bellman Rust implementation Snarky OCaml implementation (DSL) LIbsnark C++ Iden3's Circum (DSL) & SnarkJS Javascript Implementation Republic Protocol's zksnark-rs (DSL) Rust implementation DIZK Java Distributed system Go-SNARK zkSNARK library implementation in Go
STARKs	Go implementation (with pretty useful links!) C++ implementation
Bulletproofs	Ristretto Rust implementation with GREAT documentation (maintained by Chain/Interstellar) Benedikt's Bunz Java implementation

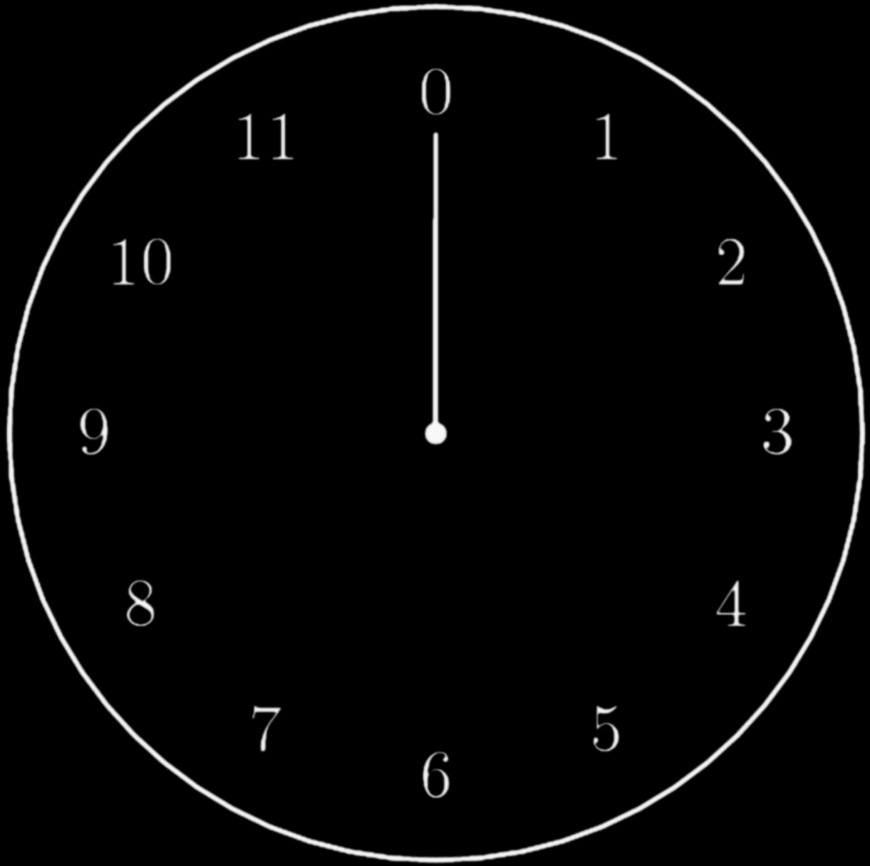
Modular Math

22 (mod 12) = 10

Because 12 divides 22 one time evenly with 10 as the remainder

3 + 16 (mod 12) = 7

And so on



Diffie Hellman (1976)

- First published method of public-key cryptography
- Secret sharing of an encryption key

(Learn more on early cryptography from [The Code Book])

Diffie Hellman (1976)

1. **p = 23** (modulus) **g = 5** (base)

Diffie Hellman (1976)

1. **p = 23** (modulus) **g = 5** (base)
2. Alice chooses a secret $a = 4$, and sends Bob **A =** g^a **mod p**
    - **A = 5^4 mod 23 = 4**

Diffie Hellman (1976)

1. **p = 23** (modulus) **g = 5** (base)
2. Alice chooses a secret a = 4, and sends Bob **A = g^a mod p**
   - **A = 5^4 mod 23 = 4**
3. Bob chooses a secret b = 3, and sends Alice **B = g^b mod p**
   - **B = 5^3 mod 23 = 10**

Diffie Hellman (1976)

1. **p = 23** (modulus) **g = 5** (base)
2. Alice chooses a secret $a = 4$, and sends Bob $\mathbf{A = g^a \bmod p}$
   - $\mathbf{A = 5^4 \bmod 23 = 4}$
3. Bob chooses a secret $b = 3$, and sends Alice $\mathbf{B = g^b \bmod p}$
   - $\mathbf{B = 5^3 \bmod 23 = 10}$
4. Alice computes $\mathbf{s}$ = $B^a \bmod p$
   - $\mathbf{s = g^{b\,a} \bmod p = 10^4 \bmod 23 = 18}$

Diffie Hellman (1976)

1. **p = 23** (modulus) **g = 5** (base)
2. Alice chooses a secret $a = 4$, and sends Bob $\mathbf{A = g^a \bmod p}$
   - $\mathbf{A = 5^4 \bmod 23 = 4}$
3. Bob chooses a secret $b = 3$, and sends Alice $\mathbf{B = g^b \bmod p}$
   - $\mathbf{B = 5^3 \bmod 23 = 10}$
4. Alice computes **s** $= B^a \bmod p$
   - $\mathbf{s = g^{b\,a} \bmod p = 10^4 \bmod 23 = 18}$
5. Bob computes **s** $= A^b \bmod p$
   - $\mathbf{s = g^{a\,b} \bmod p = 4^3 \bmod 23 = 18}$

Diffie Hellman (1976)

1. **p = 23** (modulus) **g = 5** (base)
2. Alice chooses a secret $a = 4$, and sends Bob $\mathbf{A = g^a \bmod p}$
   - $\mathbf{A = 5^4 \bmod 23 = 4}$
3. Bob chooses a secret $b = 3$, and sends Alice $\mathbf{B = g^b \bmod p}$
   - $\mathbf{B = 5^3 \bmod 23 = 10}$
4. Alice computes $\mathbf{s} = B^a \bmod p$
   - $\mathbf{s = g^{b\,a} \bmod p = 10^4 \bmod 23 = 18}$
5. Bob computes $\mathbf{s} = A^b \bmod p$
   - $\mathbf{s = g^{a\,b} \bmod p = 4^3 \bmod 23 = 18}$

Now Alice and Bob know to encrypt/decrypt their messages using the shared secret ***18***

RSA (1977)

- One of the first public key cryptosystems
- Uses Diffie-Hellman
- Digital Signatures

RSA

1. Choose 2 primes: $p = 5$ $q = 11$

   $n = p * q = 55$

RSA

1. Choose 2 primes: $p = 5$ $q = 11$
   $n = p * q = 55$
2. $\mathrm{lcm}(p-1, q-1) = 40$ (totient)

RSA

1. Choose 2 primes: $p = 5$ $q = 11$
   $n = p * q = 55$
2. $\text{lcm}(p-1, q-1) = 40$ (totient)
3. Choose a private key e that is $1 < e < 40$ (coprime to 40)
   $e = 7$

RSA

1. Choose 2 primes: $p = 5$ $q = 11$

   $n = p * q = 55$
2. $\mathrm{lcm}(p-1, q-1) = 40$ (totient)
3. Choose a private key e that is $1 < e < 40$ (coprime to 40)

   $e = 7$
4. Choose a public key d such that

   $d * e \bmod(\text{totient}) = 1$

   $d = 23$

RSA

private key $e = 7$ and public key $d = 23$
message to sign= **8**

RSA

private key $e = 7$ and public key $d = 23$
message to sign= **8**

$c = 8^{7} \bmod 55 = 2$

RSA

private key $e = 7$ and public key $d = 23$
message to sign= **8**

$c = 8^7 \bmod 55 = 2$

Anyone can check the signature using the public key 23

RSA

private key $e = 7$ and public key $d = 23$
message to sign= **8**

$c = 8^{7} \bmod 55 = 2$

Anyone can check the signature using the public key 23

$m = 2^{23} \bmod 55 = 8$

Fiat-Shamir (1986)

- Interactive proof of knowledge
- “Grandfather” of ZKPs
- Allows one to prove information about a number, without revealing the number

Fiat-Shamir

- ***g*** *is a number (called generator) everyone knows*

Fiat-Shamir

- ***g** is a number (called generator) everyone knows*
- Alice wants to prove that she knows $\mathbf{x}$, such that $y = g^x$

Fiat-Shamir

- ***g** is a number (called generator) everyone knows*
- Alice wants to prove that she knows **x**, such that $y = g^x$
- She picks a random **v**, and sends ***t*** such that $t = g^v$

Fiat-Shamir

- ***g** is a number (called generator) everyone knows*
- Alice wants to prove that she knows **x**, such that $y = g^x$
- She picks a random **v**, and sends t such that $t = g^v$
- Bob pics a random **c**, and sends that to Alice

Fiat-Shamir

- ***g** is a number (called generator) everyone knows*
- Alice wants to prove that she knows **x**, such that $y = g^x$
- She picks a random **v**, and sends t such that $t = g^v$
- Bob pics a random **c**, and sends that to Alice
- Alice sends back $r = v - cx$

Fiat-Shamir

- *g is a number (called generator) everyone knows*
- Alice wants to prove that she knows x, such that $y = g^x$
- She picks a random v, and sends t such that $t = g^v$
- Bob pics a random c, and sends that to Alice
- Alice sends back $r = v - cx$
- Bob checks $t = g^r y^c$

Fiat-Shamir

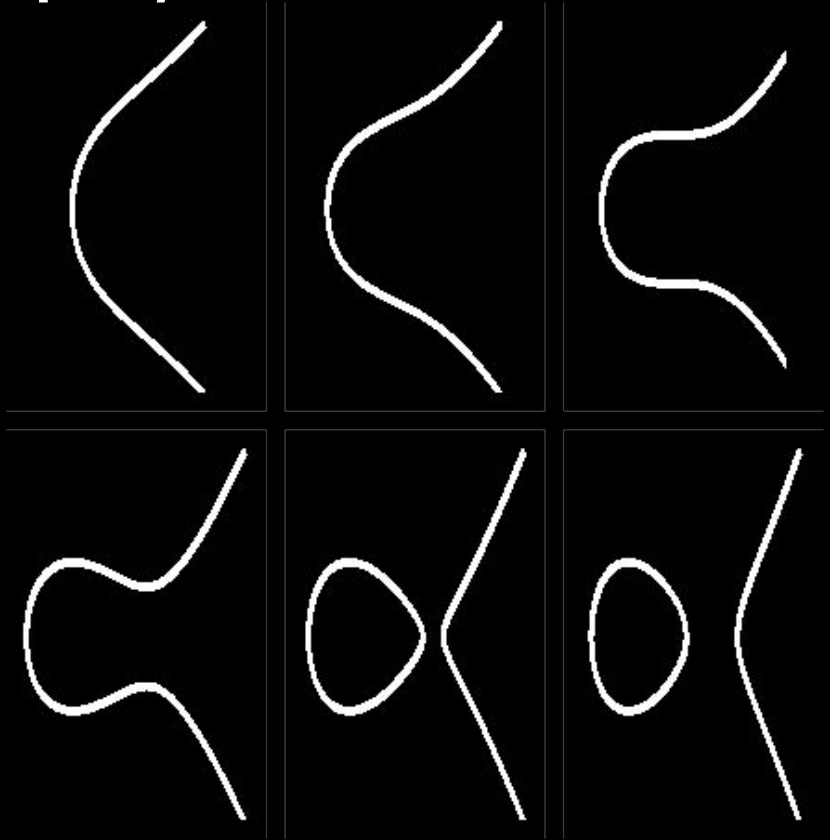
- ***g*** *is a number (called generator) everyone knows*
- Alice wants to prove that she knows $\mathbf{x}$, such that $y = g^x$
- She picks a random $\mathbf{v}$, and sends t such that $t = g^v$
- Bob pics a random $\mathbf{c}$, and sends that to Alice
- Alice sends back $r = v - cx$
- Bob checks $t = g^r y^c$
- Since $g^r y^c = g^{v-cx}\, g^{xc} = g^v = t$

Fiat-Shamir

- *g is a number (called generator) everyone knows*
- Alice wants to prove that she knows x, such that $y = g^x$
- She picks a random v, and sends t such that $t = g^v$
- Bob pics a random c, and sends that to Alice
- Alice sends back $r = v - cx$
- Bob checks $t = g^r y^c$
- Since $g^r y^c = g^{v-cx} g^{xc} = g^v = t$
- **Bob knows that Alice knows the “preimage” x**

Elliptic Curve Cryptography

- Much more powerful and efficient tool than exponentiation & modular math



Elliptic Curve Cryptography

- Much more powerful and efficient tool than exponentiation & modular math
- Great tutorial by Andrea Corbellini
  - (much of which we'll go over here)

$$y^2 = x^3 + ax + b$$

(where $4a^3 + 27b^2 \neq 0$)

Elliptic Curve Cryptography

~ ¾ of 100k top websites use ECDHE (https://)
(Elliptic Curve Diffie-Hellman Exchange)

96.1% of those use P256 curve (a NIST standard):

$$\mathbf{y^2 = x^3 - 3x + b \ (mod\ p)}$$

Parameters generated from hashing a seed

From a [video by Dan Boneh]

Elliptic Curve Cryptography

~ ¾ of 100k top websites use ECDHE (https://)

(Elliptic Curve Diffie-Hellman Exchange)

96.1% of those use P256 curve (a NIST standard):

$$y^2 = x^3 - 3x + b \bmod p$$

Parameters generated from hashing seed!

NSA Conspiracy Theory

From a [video by Dan Boneh]

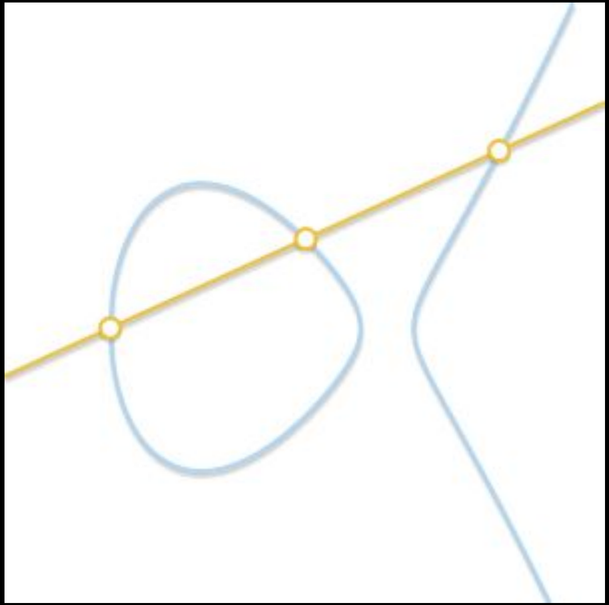
Elliptic Curve Cryptography

Abelian Group Properties for a set $\mathbb{G}$

1) **Closure**: if a and b are members of $\mathbb{G}$ then a + b is also
2) **Associativity**: (a + b) + c = a + (b + c)
3) **Identity**: a + 0 = 0 + a = a
4) **Inverse**: for every a, there exists b such that a + b = 0
5) **Commutativity**: a + b = b + a

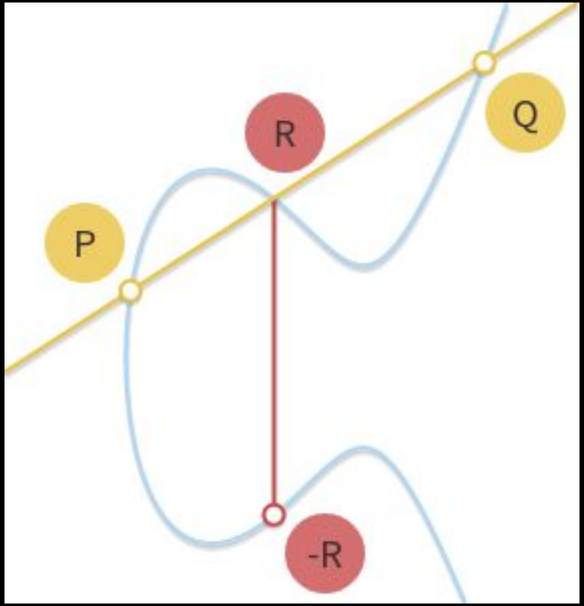
ECC - Addition

Given 3 aligned non-zero points, their sum is: P + Q + R = 0



ECC - Addition

If P + Q + R = 0, then P + Q = -R



If we draw a line between 2 pts P and Q, it'll intersect at a point R

Since P + Q = -R we can easily find R and use it to compute addition of two points from it

ECC - Addition

$P + Q = -R$

$m = (3x_P{}^3 + a) / 2y_P$

$x_R = m^2 - x_p - x_Q$

$y_R = y_P + m(x_R - x_P) = y_Q + m(x_R - x_Q)$

ECC - Addition

$P + P = -R$ *for $Q = P = (1, 2)$ on $y^2 = x^3 - 7x + 10$*

$m = (3x_P{}^3 + a) / 2y_P$ $= -1$

$x_R = m^2 - x_p - x_Q$ $= -1$

$y_R = y_P + m(x_R - x_P) = y_Q + m(x_R - x_Q)$ $= 4$

So $P + P = (-1, -4)$

Curve:	a -7	b 10
P:	x 1	y 2
Q:	x 1	y 2
R = *P* + *Q*:	x -1	y -4

Point addition over the elliptic curve $y^2 = x^3 - 7x + 10$ in $\mathbb{R}$.

ECC - Multiplication

$nP = P + P + \ldots + P$

n times

(We have clever optimization techniques to do this fast)

ECC - Multiplication

$nP = P + P + \ldots + P$

n times

$nP = Q$: fairly easily to compute
$n = Q/P$: very hard to compute (no efficient method)

Logarithm Problem (it’s not called division for conformity reasons)

ECC - Multiplication

nP = P + P + … + P

n times

nP = Q : fairly easily to compute
n = Q/P : very hard to compute (no efficient method)

Logarithm Problem (it's not called division for conformity reasons)

But we're missing cycles

ECC - Finite Fields & Discrete Log

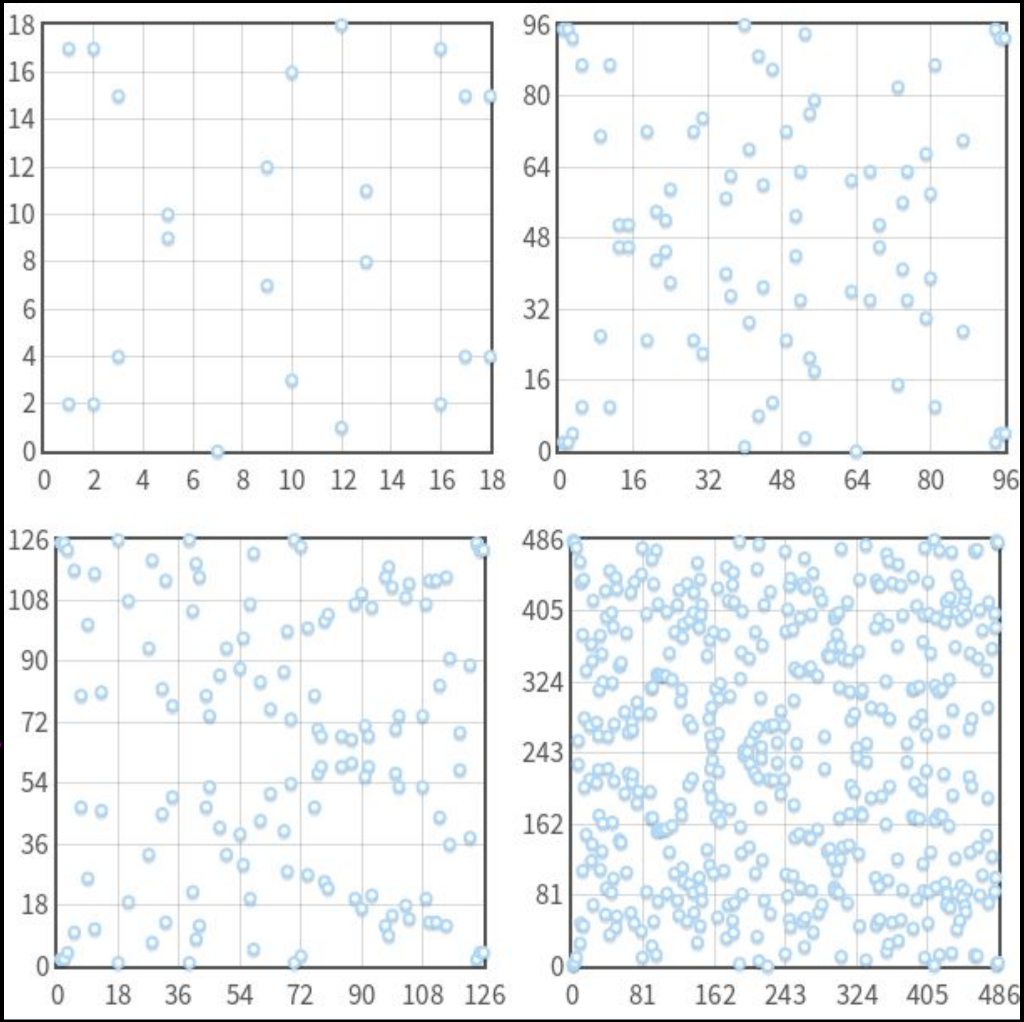
Finite field $\mathbb{F}p$: set of elements (mod p) where p is prime

$y^2 = x^3 + ax + b \pmod{p}$

p = 19

p = 97

$$y^2 = x^3 - 7x + 10$$



p = 127

p = 487

ECC - Finite Fields (Addition)

P + P = ? in a finite field

ECC - Finite Fields (Addition)

P + P = ? in a finite field

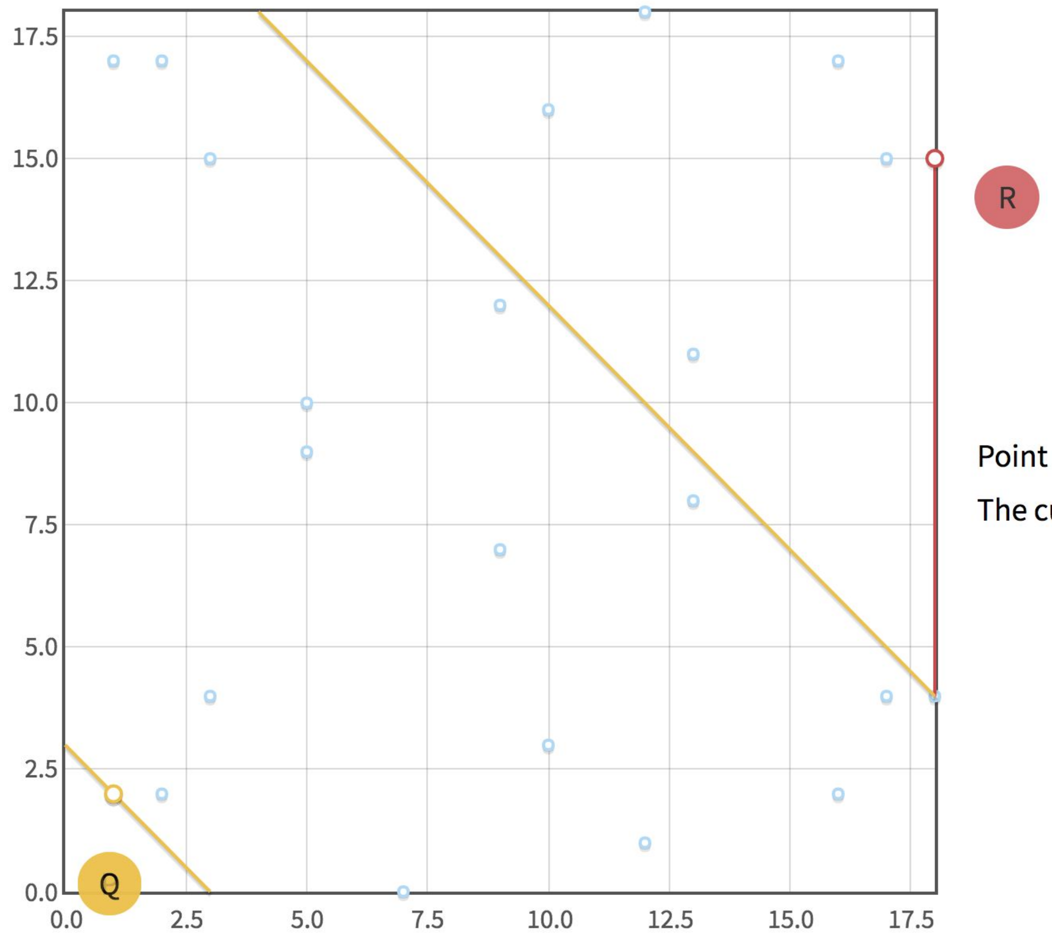
P + P = (-1, -4) on $y^2 = x^3 - 7x + 10$

For a finite field on p = 19

-1 (mod 19) = 18

-4 (mod 19) = 15

So P + P on $y^2 = x^3 - 7x + 10$ (mod 19) = (18, 15)



Curve:	a -7	b 10
Field:	p 19	
P:	x 1	y 2
Q:	x 1	y 2
$R = P + Q$:	x 18	y 15

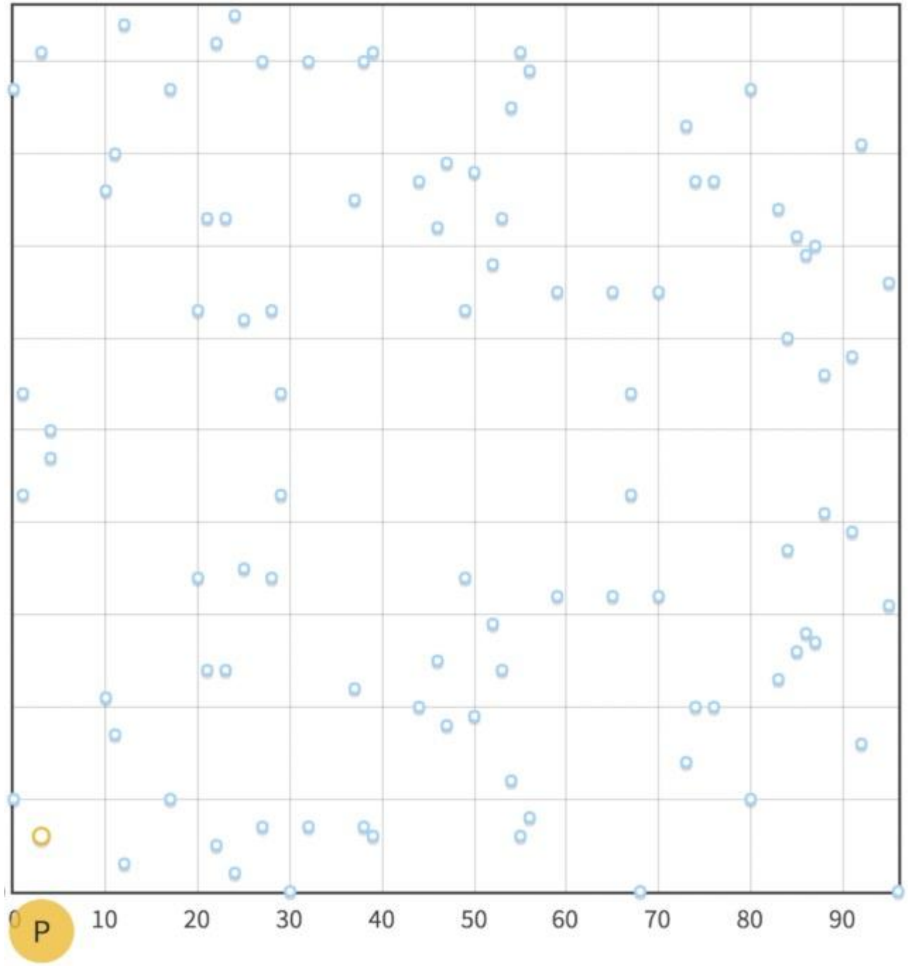
Point addition over the elliptic curve $y^2 = x^3 - 7x + 10$ in $\mathbb{F}_{19}$.

The curve has 24 points (including the point at infinity).

ECC - Groups

$y^2 = x^3 + 2x + 3 \pmod{97}$ at point $P(3, 6)$

Let's calculate some multiples of P



Curve: a 2 b 3

Field: p 97

n: n 0

P: x 3 y 6

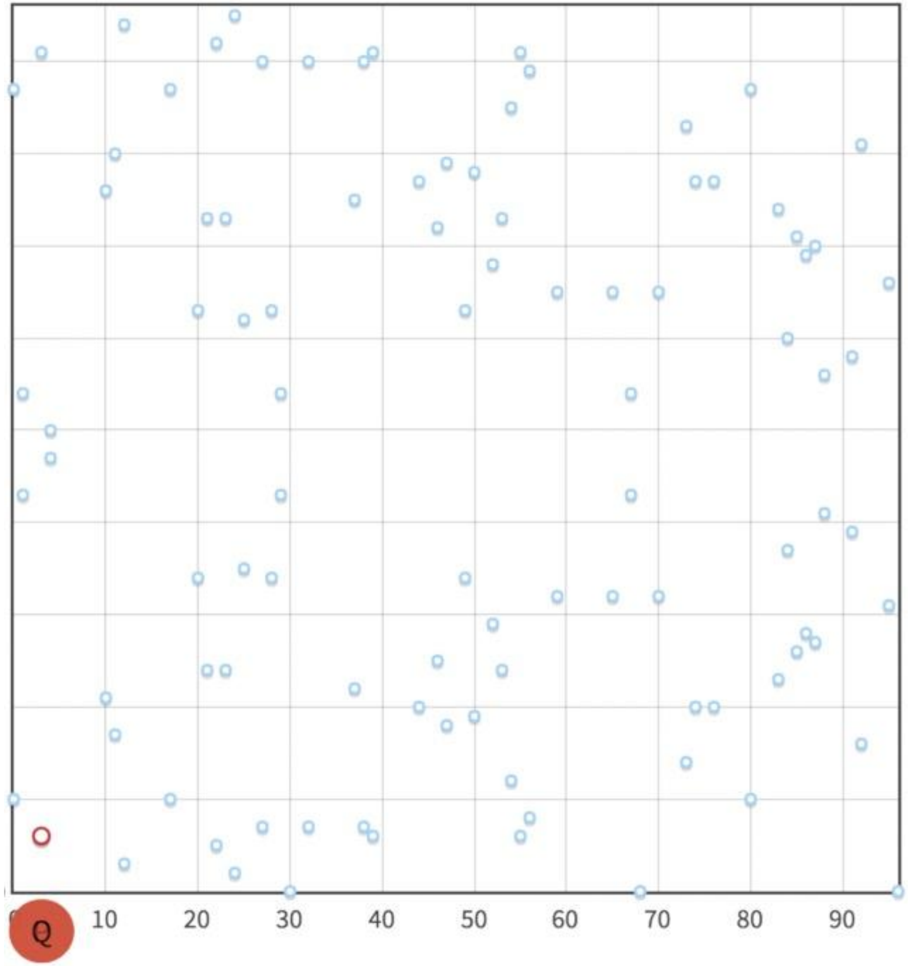
$Q = n \cdot P$: x Inf y Inf

Scalar multiplication over the elliptic curve $y^2 = x^3 + 2x + 3$ in $\mathbb{F}_{97}$.

The curve has 100 points (including the point at infinity).

The subgroup generated by P has 5 points.

n = 0



Curve: a 2 b 3

Field: p 97

n: n 1

P: x 3 y 6

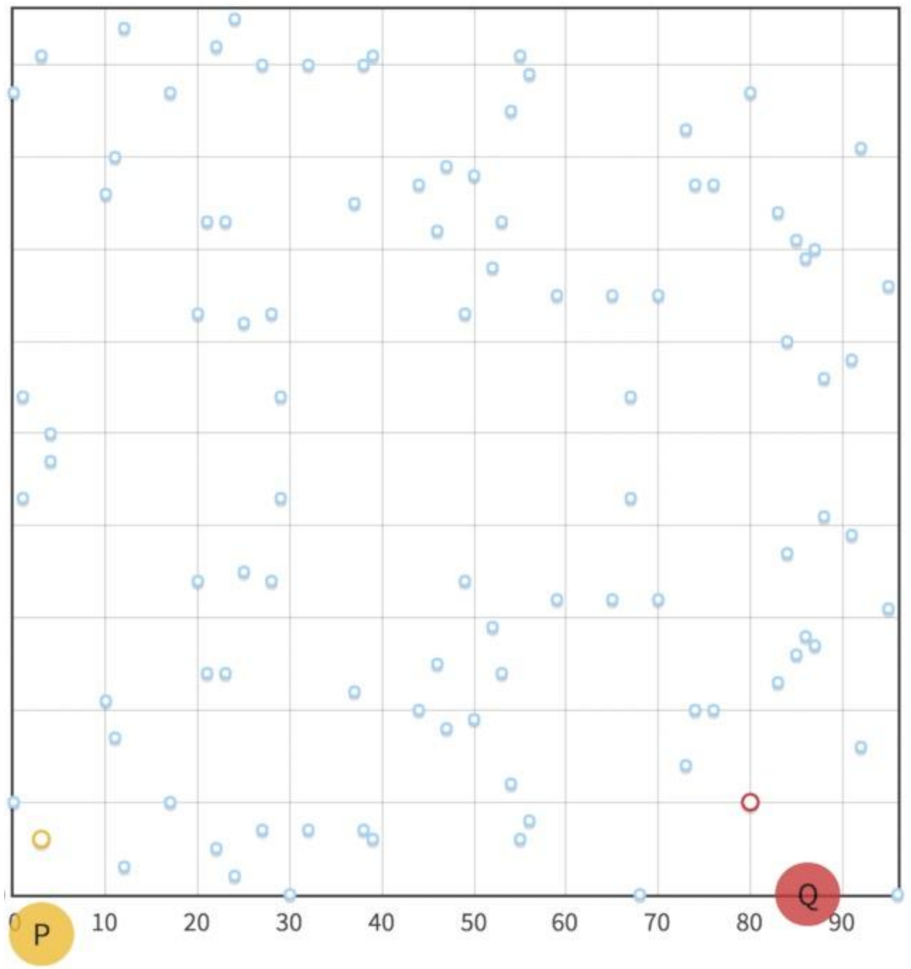
$Q = n \cdot P$: x 3 y 6

Scalar multiplication over the elliptic curve $y^2 = x^3 + 2x + 3$ in $\mathbb{F}_{97}$.

The curve has 100 points (including the point at infinity).

The subgroup generated by P has 5 points.

n = 1



Curve: a 2 b 3

Field: p 97

n: n 2

P: x 3 y 6

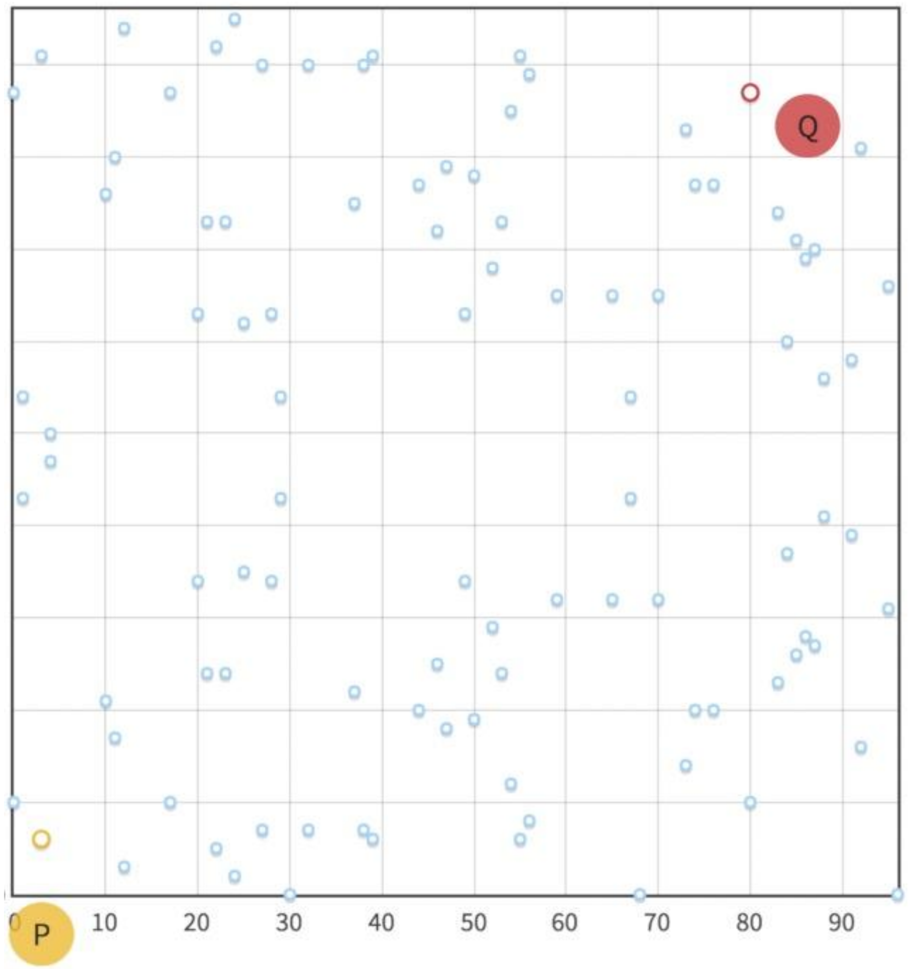
$Q = n \cdot P$: x 80 y 10

Scalar multiplication over the elliptic curve $y^2 = x^3 + 2x + 3$ in $\mathbb{F}_{97}$.

The curve has 100 points (including the point at infinity).

The subgroup generated by P has 5 points.

n = 2



Curve: a 2 b 3

Field: p 97

n: n 3

P: x 3 y 6

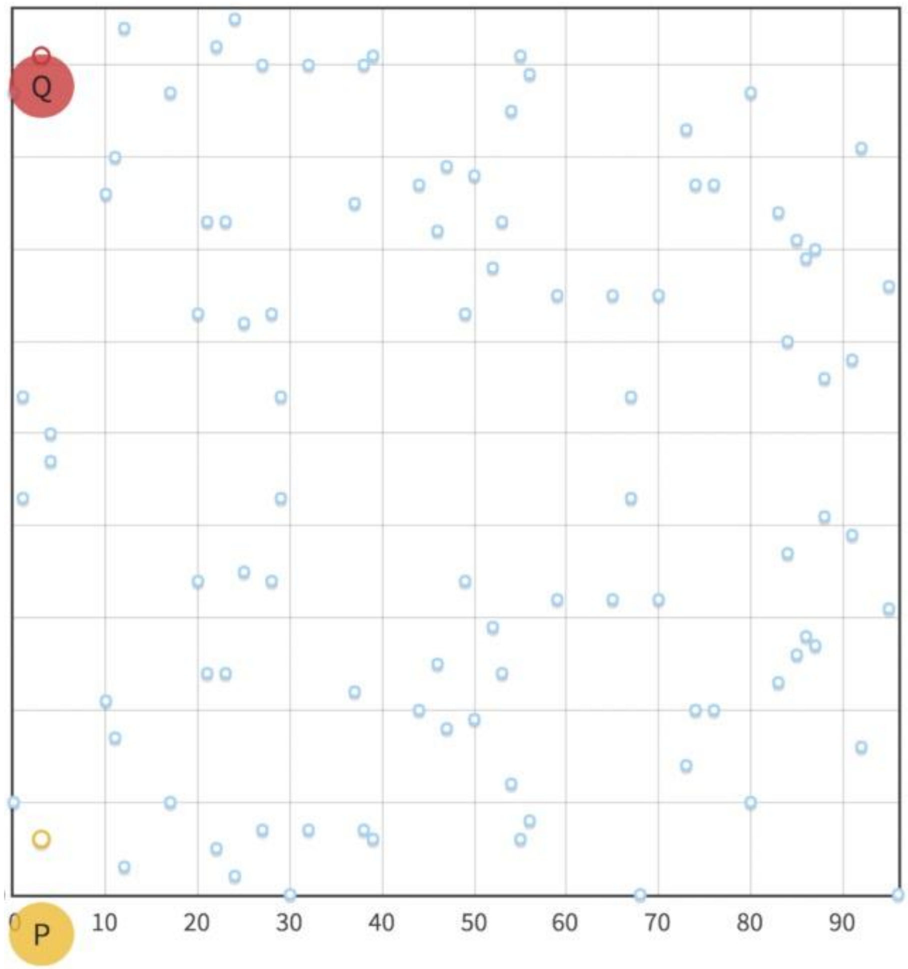
$Q = n \cdot P$: x 80 y 87

Scalar multiplication over the elliptic curve $y^2 = x^3 + 2x + 3$ in $\mathbb{F}_{97}$.

The curve has 100 points (including the point at infinity).

The subgroup generated by *P* has 5 points.

n = 3



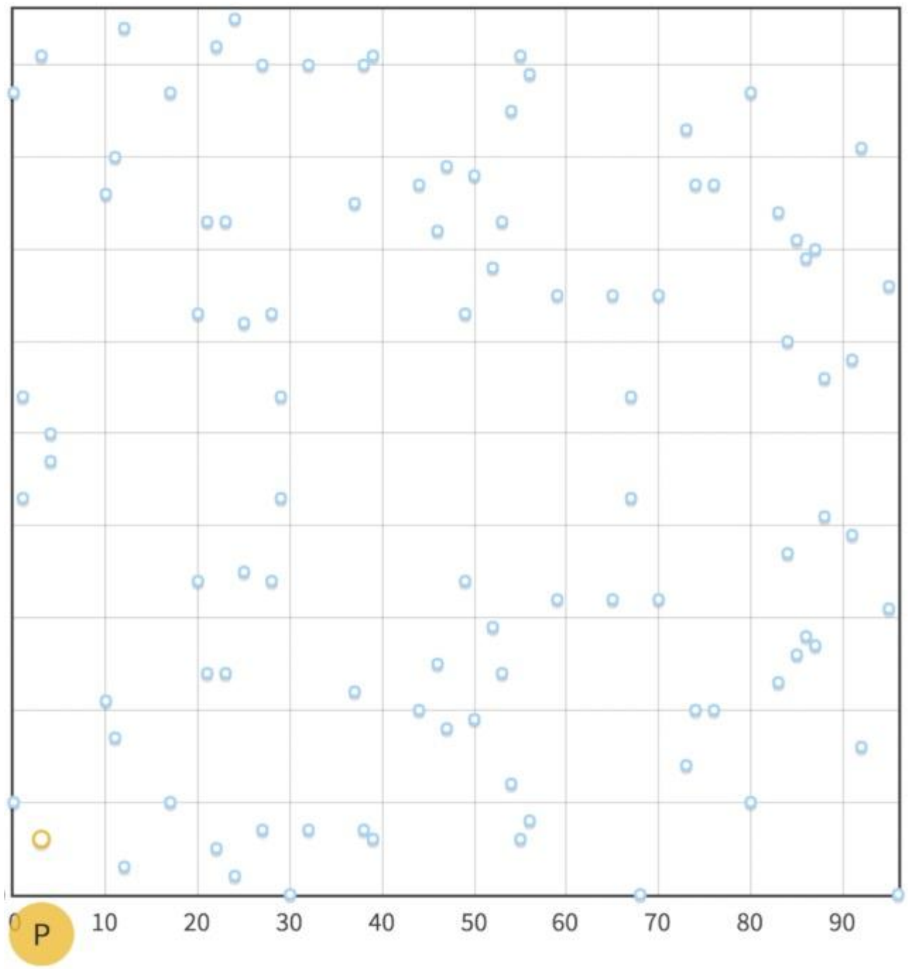
Curve:	a 2	b 3
Field:	p 97	
n:	n 4	
P:	x 3	y 6
$Q = n \cdot P$:	x 3	y 91

Scalar multiplication over the elliptic curve $y^2 = x^3 + 2x + 3$ in $\mathbb{F}_{97}$.

The curve has 100 points (including the point at infinity).

The subgroup generated by P has 5 points.

n = 4



Curve: a 2 b 3

Field: p 97

n: n 5

P: x 3 y 6

$Q = n \cdot P$: x Inf y Inf

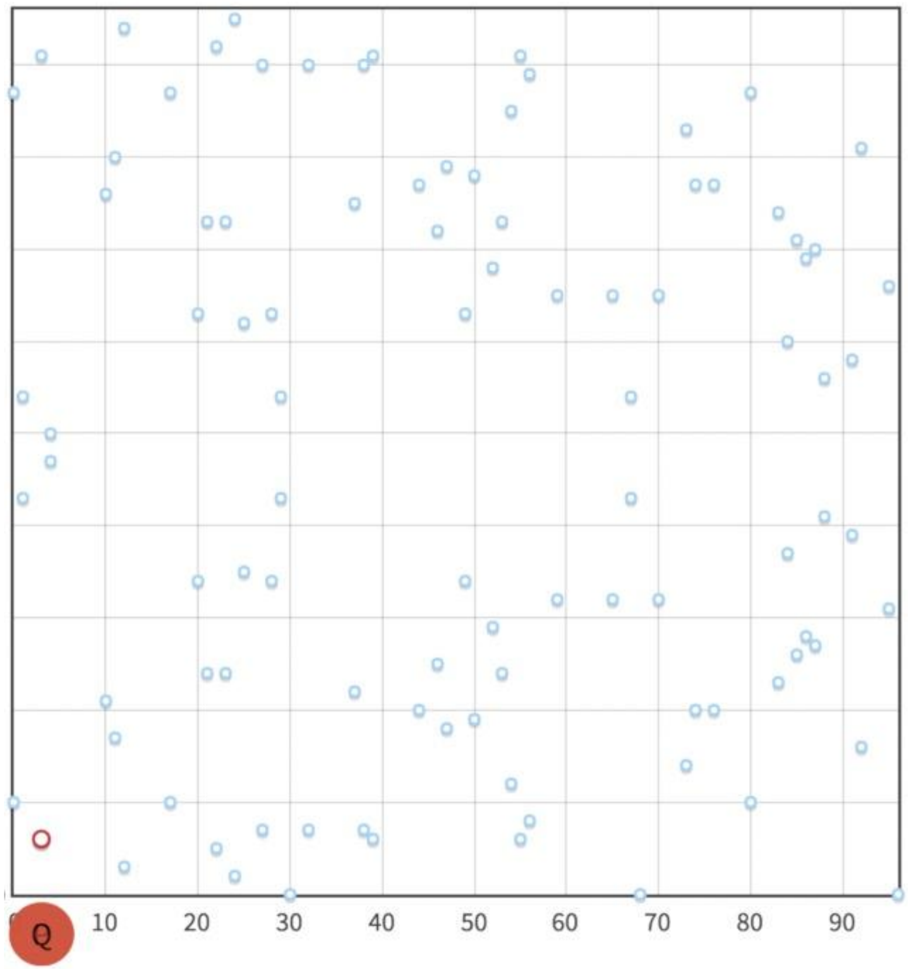
Scalar multiplication over the elliptic curve $y^2 = x^3 + 2x + 3$ in $\mathbb{F}_{97}$.

The curve has 100 points (including the point at infinity).

The subgroup generated by P has 5 points.

n = 5

Same as n = 0



Curve:	a 2	b 3
Field:	p 97	
n:	n 6	
P:	x 3	y 6
$Q = n \cdot P$:	x 3	y 6

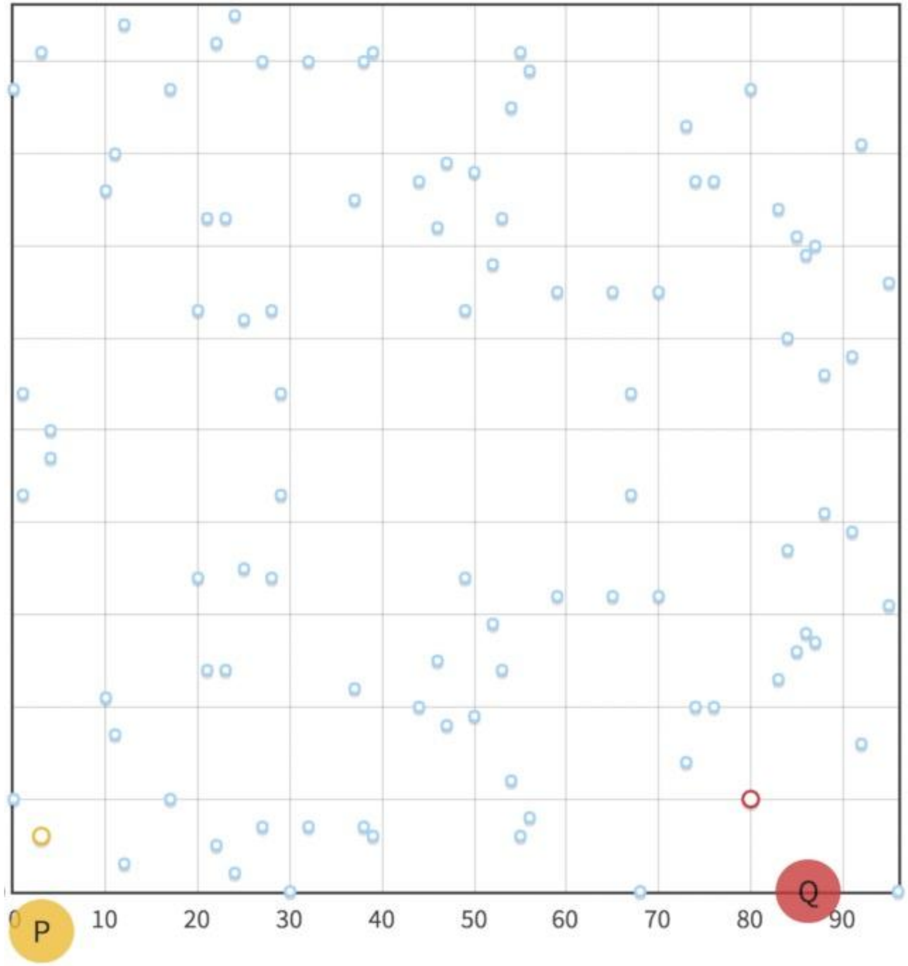
Scalar multiplication over the elliptic curve $y^2 = x^3 + 2x + 3$ in $\mathbb{F}_{97}$.

The curve has 100 points (including the point at infinity).

The subgroup generated by P has 5 points.

n = 6

Same as n = 1



Curve: a 2 b 3

Field: p 97

n: n 7

P: x 3 y 6

$Q = n \cdot P$: x 80 y 10

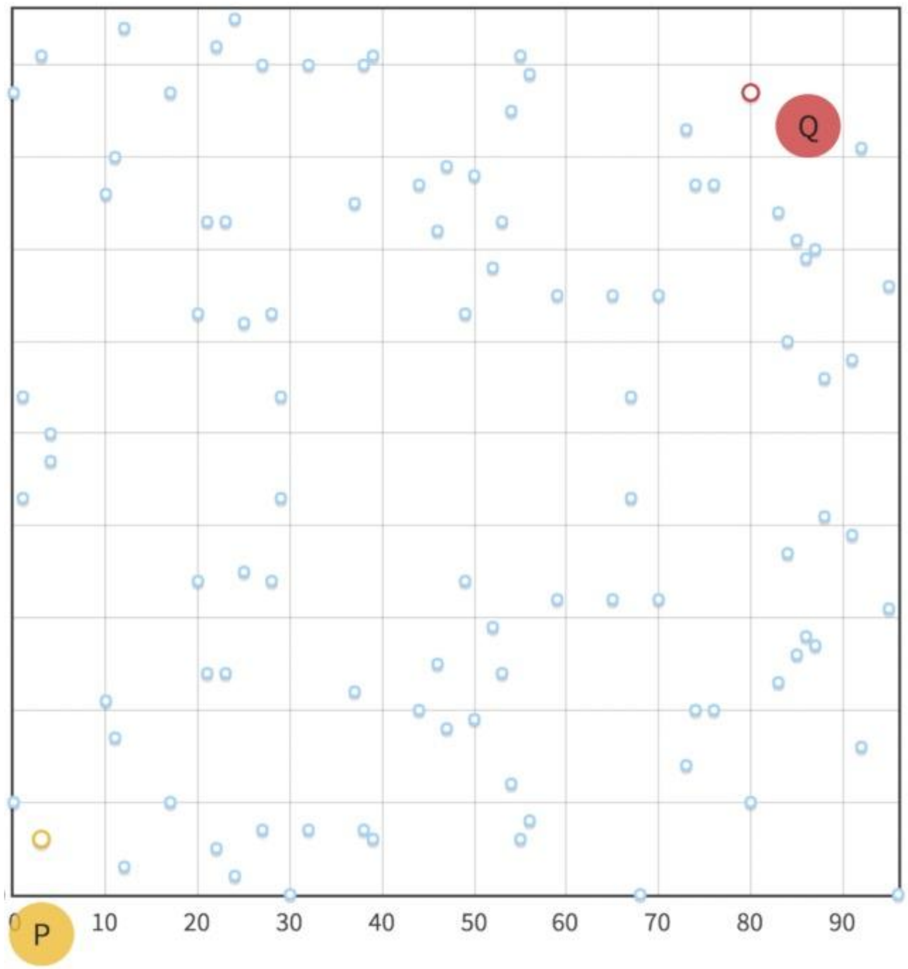
Scalar multiplication over the elliptic curve $y^2 = x^3 + 2x + 3$ in $\mathbb{F}_{97}$.

The curve has 100 points (including the point at infinity).

The subgroup generated by P has 5 points.

n = 7

Same as n = 2



Curve:	a 2	b 3
Field:	p 97	
n:	n 8	
P:	x 3	y 6
$Q = n \cdot P$:	x 80	y 87

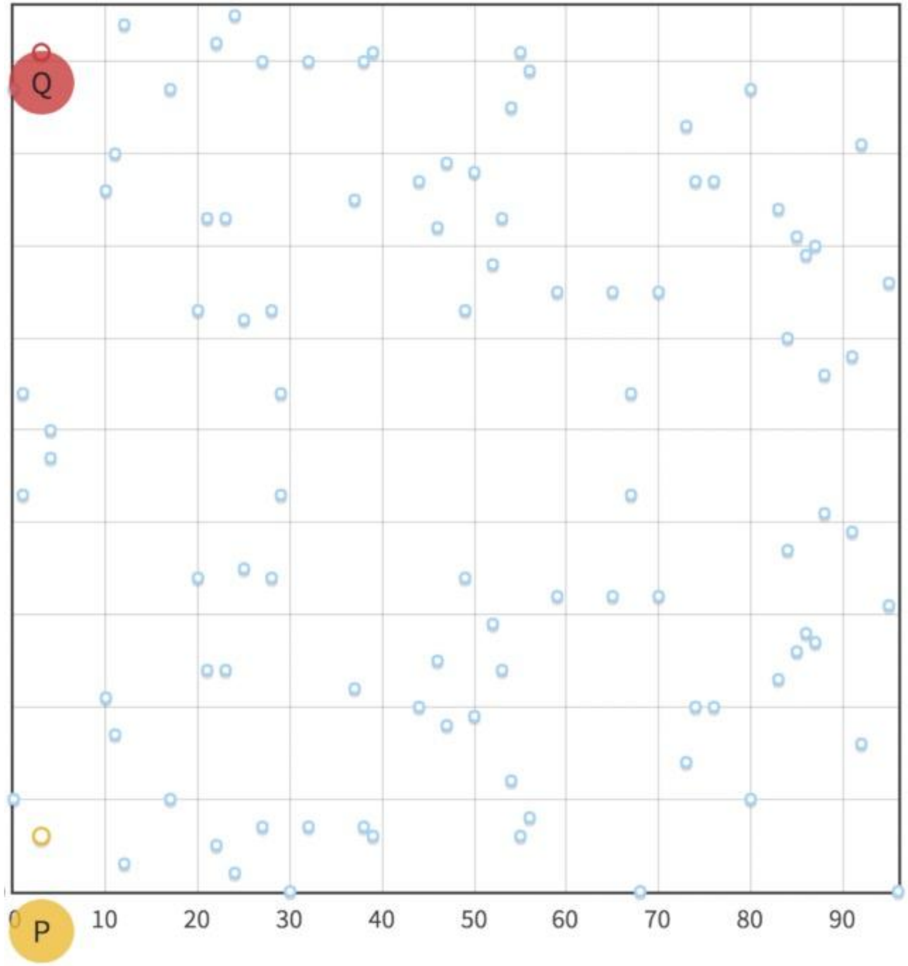
Scalar multiplication over the elliptic curve $y^2 = x^3 + 2x + 3$ in $\mathbb{F}_{97}$.

The curve has 100 points (including the point at infinity).

The subgroup generated by P has 5 points.

n = 8

Same as n = 3



Curve: a 2 b 3

Field: p 97

n: n 9

P: x 3 y 6

$Q = n \cdot P$: x 3 y 91

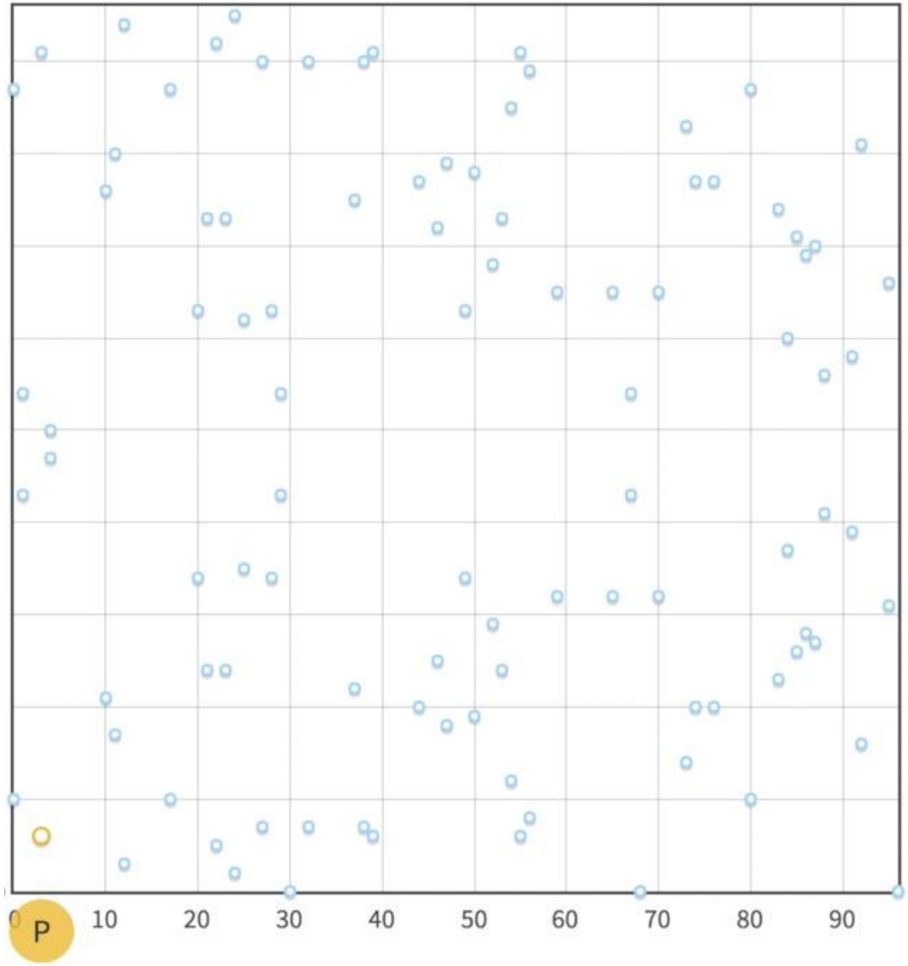
Scalar multiplication over the elliptic curve $y^2 = x^3 + 2x + 3$ in $\mathbb{F}_{97}$.

The curve has 100 points (including the point at infinity).

The subgroup generated by P has 5 points.

n = 9

Same as n = 4



Curve:	a 2	b 3
Field:	p 97	
n:	n 10	
P:	x 3	y 6
$Q = n \cdot P$:	x Inf	y Inf

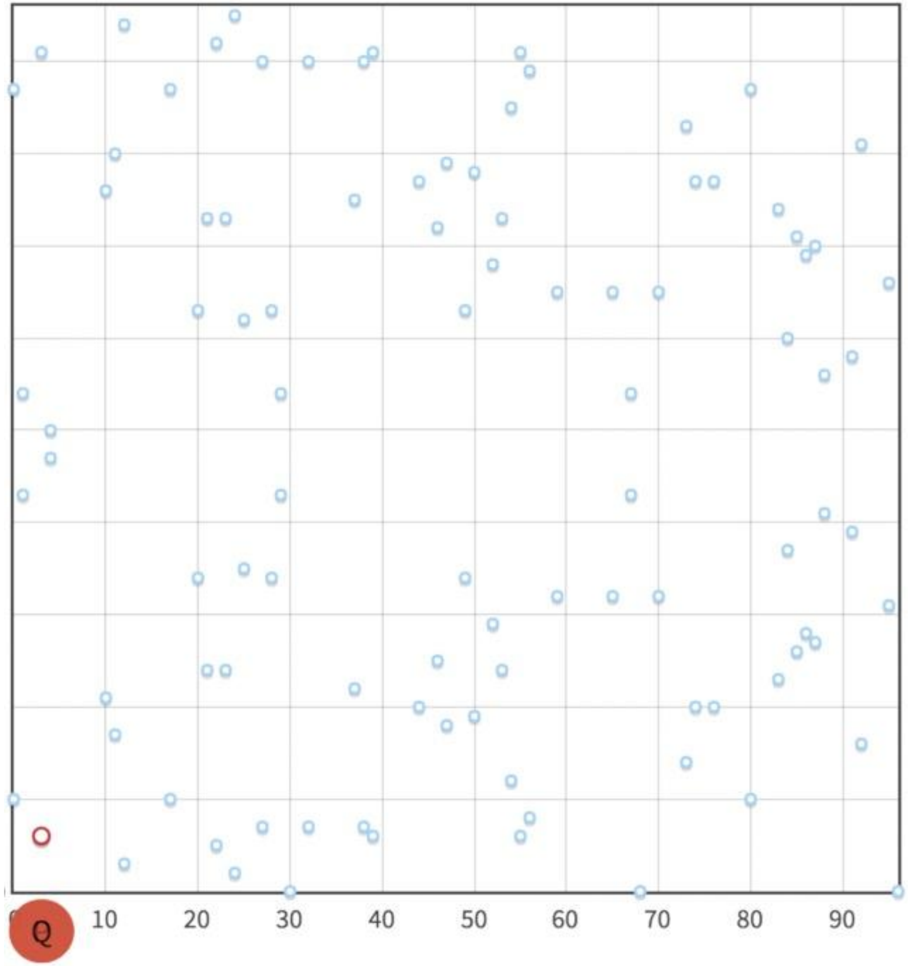
Scalar multiplication over the elliptic curve $y^2 = x^3 + 2x + 3$ in $\mathbb{F}_{97}$.

The curve has 100 points (including the point at infinity).

The subgroup generated by P has 5 points.

n = 10

Same as n = 5, 0



Curve:	a 2	b 3
Field:	p 97	
n:	n 11	
P:	x 3	y 6
$Q = n \cdot P$:	x 3	y 6

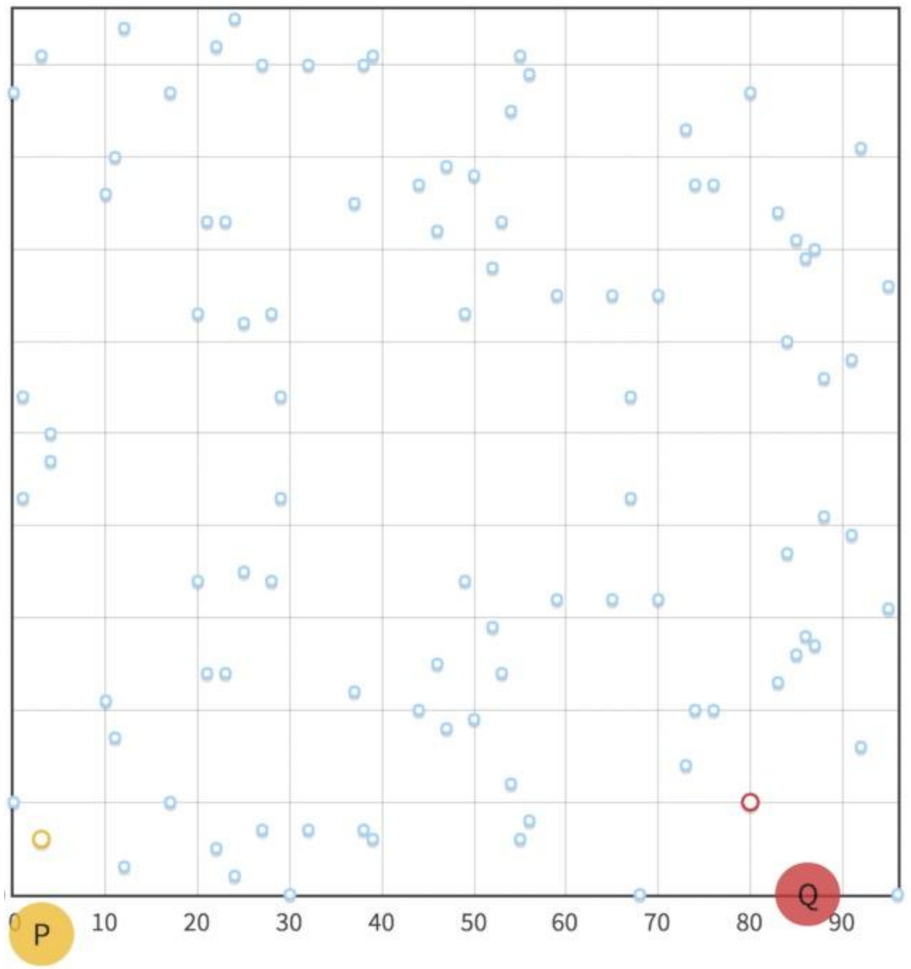
Scalar multiplication over the elliptic curve $y^2 = x^3 + 2x + 3$ in $\mathbb{F}_{97}$.

The curve has 100 points (including the point at infinity).

The subgroup generated by P has 5 points.

n = 11

Same as n = 1, 6



Curve:	a 2	b 3
Field:	p 97	
n:	n 12	
P:	x 3	y 6
$Q = n \cdot P$:	x 80	y 10

Scalar multiplication over the elliptic curve $y^2 = x^3 + 2x + 3$ in $\mathbb{F}_{97}$.

The curve has 100 points (including the point at infinity).

The subgroup generated by P has 5 points.

n = 12

Same as n = 2, 7

… and the pattern continues

ECC - Groups

$y^2 = x^3 + 2x + 3 \pmod{97}$ on P(3, 6) we saw a cycle

There are just 5 distinct points:

0, P, 2P, 3P, 4P

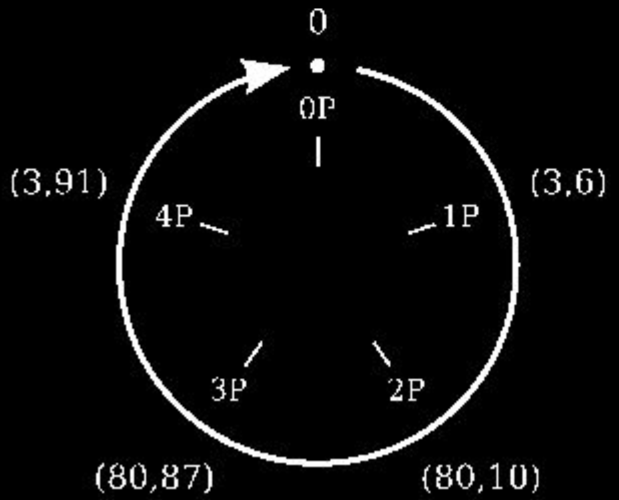
ECC - Groups

$y^2 = x^3 + 2x + 3 \pmod{97}$ on P(3, 6) we saw a cycle

There are just 5 distinct points:

0, P, 2P, 3P, 4P

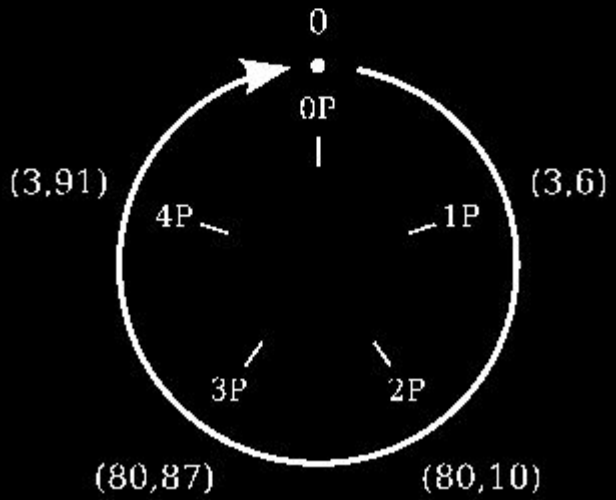
These five points are closed under addition.
However you add 0, P, 2P, 3P or 4P, the result
is always one of these five points.



ECC - Groups

The point P(3, 6) on $y^2 = x^3 + 2x + 3 \pmod{97}$
is therefore said to be a generator
or base point of the cyclic subgroup

and the order of this subgroup is
therefore 5
(the smallest n such that nP=0)



ECC - Groups

$Q = nP$ mod p : easy to calculate
n = P/Q mod p : discrete logarithm problem

If you choose your curve, parameters, and generator point carefully, finding n becomes very very very hard

And this is exactly how the **ECDSA** signature scheme works

ECDSA

ECDSA signature (used in Bitcoin, Ethereum, etc):

1. private key random integer **d** from {1, … n-1} where n is the order of the subgroup
2. public key where G is the base point **H = dG**

ECDSA

A Bitcoin private key is 256 bits and gives 128-bit security level

To achieve the same security with RSA you would need **3092** bit length key

Pedersen Commitments

Tool for Confidential Transaction: Pedersen Commitment

Used in privacy coin Grin

[Learn more about how Grin works]

Pedersen Commitments

You could use $Q = vG$ to “hash” or hide amounts per tx

But that’s susceptible to ‘Rainbow table’ attacks as one could precompute popular amount denominations

Pedersen Commitments

You could use $Q = vG$ to "hash" or hide amounts per tx

But that's susceptible to 'Rainbow table' attacks as one could precompute popular amount denominations

Solution: add a blinding factor

$$\mathbf{Com(v) = vG + bH}$$

Pedersen Commitments

You could use $Q = vG$ to “hash” or hide amounts per tx

But that’s susceptible to ‘Rainbow table’ attacks as one could precompute popular amount denominations

Solution: add a blinding factor

$$\mathrm{Com}(v) = vG + bH$$

Where G and H are generator points
b is random number used as a **blinding factor**

Pedersen Commitments

The cool property of Pedersen Commitments (or sometimes called hashes) is that you can add them to check equality

$$Com(v1) = v1G + b1H$$

$$Com(v2) = v2G + b2H$$

Such that:

$$v2G + b2H + v1G + b1H = G(v2 + v1) + H(b1 + b2)$$

Bulletproofs (Range Proofs)

Proving that a number is within a range

$v \in [0,2^n)$

Zero Knowledge about the Inner Product of Two Vectors

[Learn more how Bulletproofs work]()

Bulletproofs (Range Proofs)

Any number can be represented as inner product of two vectors.

$5 = \langle [1, 0, 1], [2^2, 2^1, 2^0] \rangle$

5 equals inner product of 2 vectors [1, 0, 1] and $[2^2, 2^1, 2^0]$

Bulletproofs (Range Proofs)

This is also how binary works

$101_{binary} = 5_{decimal}$ since $1(2^2) + 0(2^1) + 1(2^0)$

Bulletproofs (Range Proofs)

$v = \langle a, 2^n \rangle$

Example:
v = 5 and we wanted to prove that 5 is in range of 0 to 2^n
without showing 5

$$v \in [0, 2^n)$$

Bulletproofs (Range Proofs)

$5_{decimal} = 101_{binary} = 1(2^2) + 0(2^1) + 1(2^0)$

Need at least 3 bits to represent 5: **n has to be at least 3**

$2^3 = 8$

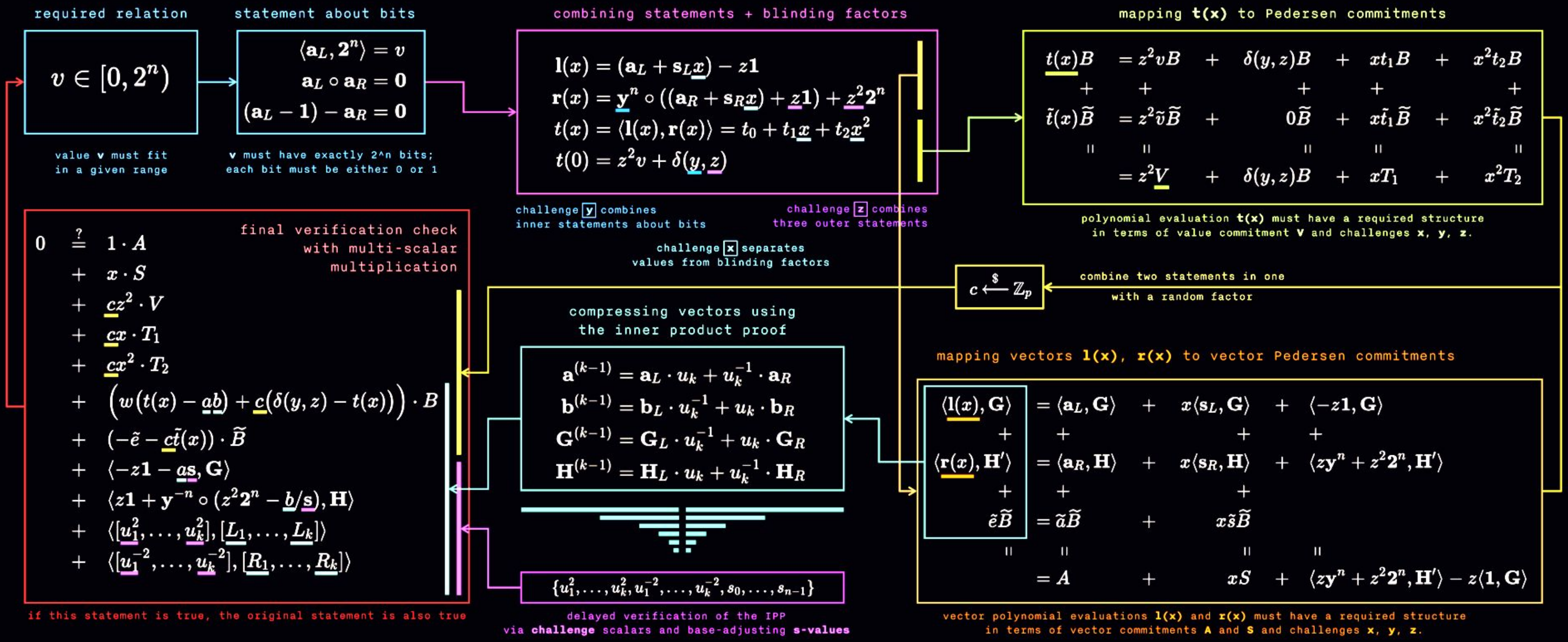
Therefore it's clear that 5 must be in range of 0 to 2^n

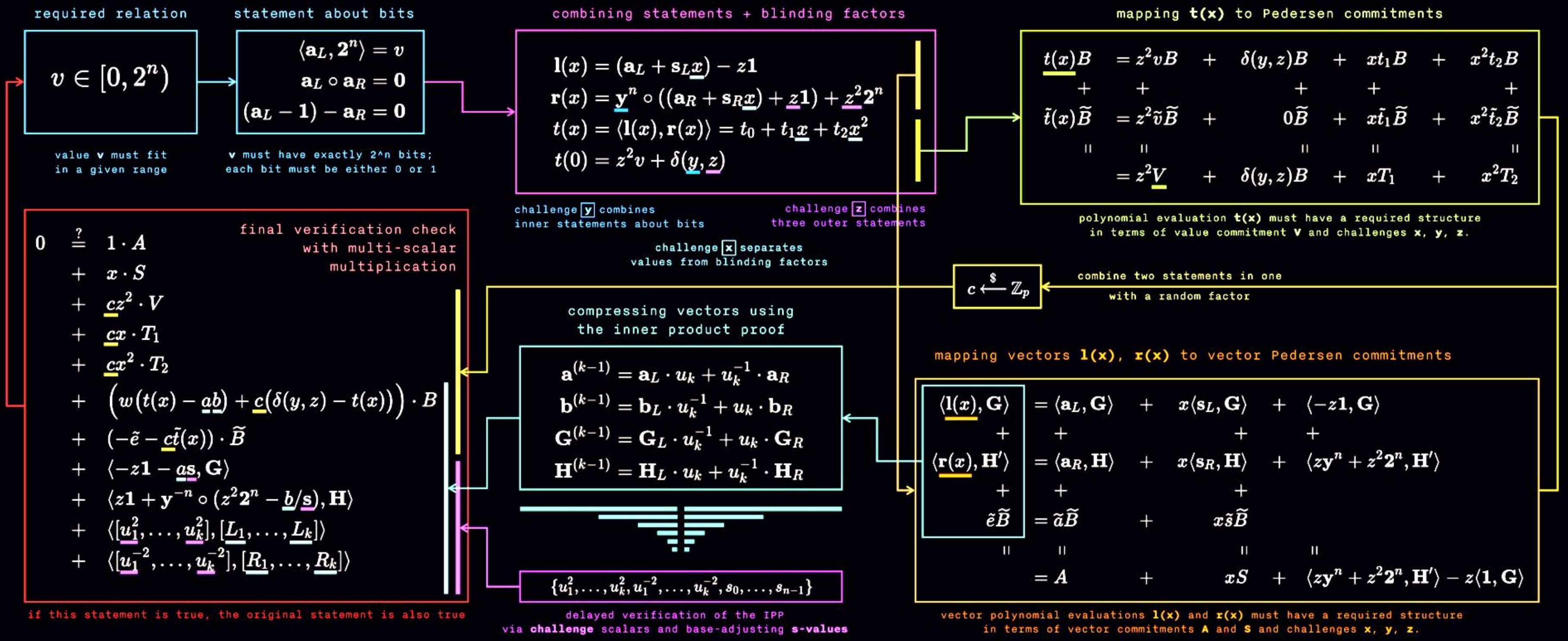
Bulletproofs (Range Proofs)

$v = \langle a, 2^n \rangle$

We need to prove that:

1) That assignment of **a** is correct
2) We can commit to secret values using Pedersen commitments and prove we know the value using a technique similar to **Fiat-Shamir**


required relation
$v \in [0, 2^n)$
value v must fit in a given range
statement about bits
$\langle \mathbf{a}_L, \mathbf{2}^n \rangle = v$
$\mathbf{a}_L \circ \mathbf{a}_R = \mathbf{0}$
$(\mathbf{a}_L - \mathbf{1}) - \mathbf{a}_R = \mathbf{0}$
v must have exactly 2^n bits; each bit must be either 0 or 1
combining statements + blinding factors
$\mathbf{l}(x) = (\mathbf{a}_L + \mathbf{s}_L x) - z\mathbf{1}$
$\mathbf{r}(x) = \mathbf{y}^n \circ ((\mathbf{a}_R + \mathbf{s}_R x) + z\mathbf{1}) + z^2\mathbf{2}^n$
$t(x) = \langle \mathbf{l}(x), \mathbf{r}(x) \rangle = t_0 + t_1 x + t_2 x^2$
$t(0) = z^2 v + \delta(y, z)$
challenge $\boxed{y}$ combines inner statements about bits
challenge $\boxed{z}$ combines three outer statements
challenge $\boxed{x}$ separates values from blinding factors
mapping t(x) to Pedersen commitments
$t(x)B = z^2 vB + \delta(y,z)B + xt_1B + x^2t_2B$
$\tilde{t}(x)\widetilde{B} = z^2\tilde{v}\widetilde{B} + 0\widetilde{B} + x\tilde{t}_1\widetilde{B} + x^2\tilde{t}_2\widetilde{B}$
$= z^2V + \delta(y,z)B + xT_1 + x^2T_2$
polynomial evaluation t(x) must have a required structure in terms of value commitment V and challenges x, y, z.
$c \xleftarrow{\$} \mathbb{Z}_p$
combine two statements in one with a random factor
final verification check with multi-scalar multiplication
$0 \stackrel{?}{=} 1 \cdot A$
$+ x \cdot S$
$+ cz^2 \cdot V$
$+ cx \cdot T_1$
$+ cx^2 \cdot T_2$
$+ \left(w\left(t(x) - ab\right) + c\left(\delta(y,z) - t(x)\right)\right) \cdot B$
$+ (-\tilde{e} - c\tilde{t}(x)) \cdot \widetilde{B}$
$+ \langle -z\mathbf{1} - a\mathbf{s}, \mathbf{G} \rangle$
$+ \langle z\mathbf{1} + \mathbf{y}^{-n} \circ (z^2\mathbf{2}^n - b/\mathbf{s}), \mathbf{H} \rangle$
$+ \langle [u_1^2, \ldots, u_k^2], [L_1, \ldots, L_k] \rangle$
$+ \langle [u_1^{-2}, \ldots, u_k^{-2}], [R_1, \ldots, R_k] \rangle$
if this statement is true, the original statement is also true
compressing vectors using the inner product proof
$\mathbf{a}^{(k-1)} = \mathbf{a}_L \cdot u_k + u_k^{-1} \cdot \mathbf{a}_R$
$\mathbf{b}^{(k-1)} = \mathbf{b}_L \cdot u_k^{-1} + u_k \cdot \mathbf{b}_R$
$\mathbf{G}^{(k-1)} = \mathbf{G}_L \cdot u_k^{-1} + u_k \cdot \mathbf{G}_R$
$\mathbf{H}^{(k-1)} = \mathbf{H}_L \cdot u_k + u_k^{-1} \cdot \mathbf{H}_R$
$\{u_1^2, \ldots, u_k^2, u_1^{-2}, \ldots, u_k^{-2}, s_0, \ldots, s_{n-1}\}$
delayed verification of the IPP via **challenge** scalars and base-adjusting **s-values**
mapping vectors l(x), r(x) to vector Pedersen commitments
$\langle \mathbf{l}(x), \mathbf{G} \rangle = \langle \mathbf{a}_L, \mathbf{G} \rangle + x\langle \mathbf{s}_L, \mathbf{G} \rangle + \langle -z\mathbf{1}, \mathbf{G} \rangle$
$\langle \mathbf{r}(x), \mathbf{H}' \rangle = \langle \mathbf{a}_R, \mathbf{H} \rangle + x\langle \mathbf{s}_R, \mathbf{H} \rangle + \langle z\mathbf{y}^n + z^2\mathbf{2}^n, \mathbf{H}' \rangle$
$\tilde{e}\widetilde{B} = \tilde{a}\widetilde{B} + x\tilde{s}\widetilde{B}$
$= A + xS + \langle z\mathbf{y}^n + z^2\mathbf{2}^n, \mathbf{H}' \rangle - z\langle \mathbf{1}, \mathbf{G} \rangle$
vector polynomial evaluations l(x) and r(x) must have a required structure in terms of vector commitments A and S and challenges x, y, z.



Learn more how Bulletproofs work

STARKs

Please refer to Remco from 0x for all your STARK questions :)

SNARKs

There are ***many*** (50+) variants of SNARKs

QAP variant (quadratic arithmetic program):

Computation → Arithmetic Circuit → R1CS → QAP → SNARK

Check out this [excellent primer by Decentriq]()

And [Zcash's explainer]()

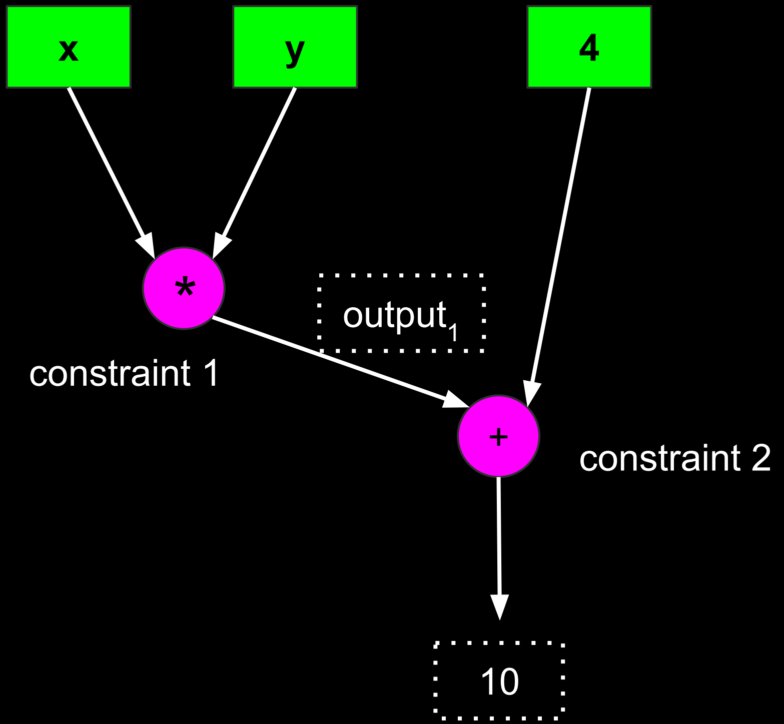
SNARKs - Computation

x * y + 4 = 10

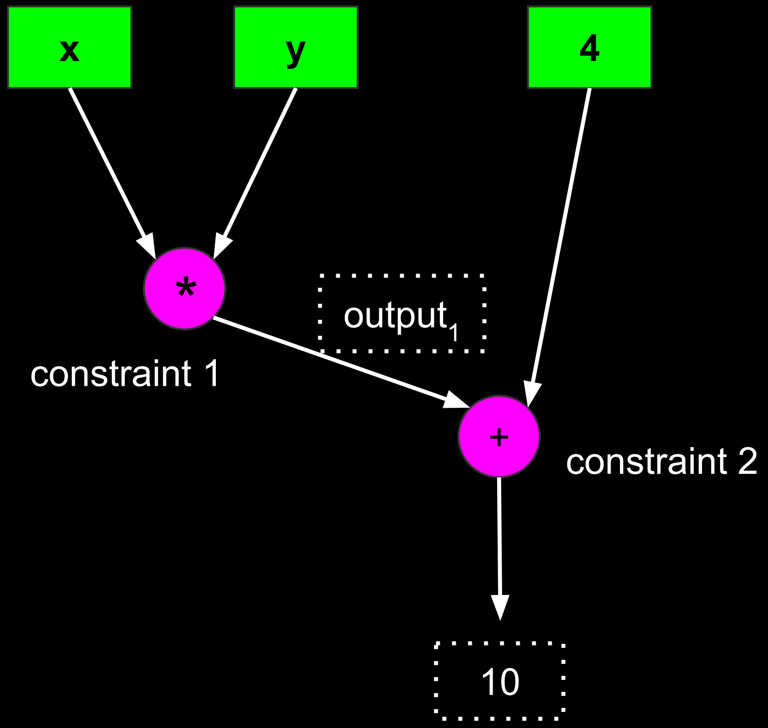
Prover wants to prove that they know values **x** and **y**

SNARKs - Arithmetic Circuit

$x * y + 4 = 10$



SNARKs - Arithmetic Circuit



Constraint 1:

$$output_1 = x * y$$

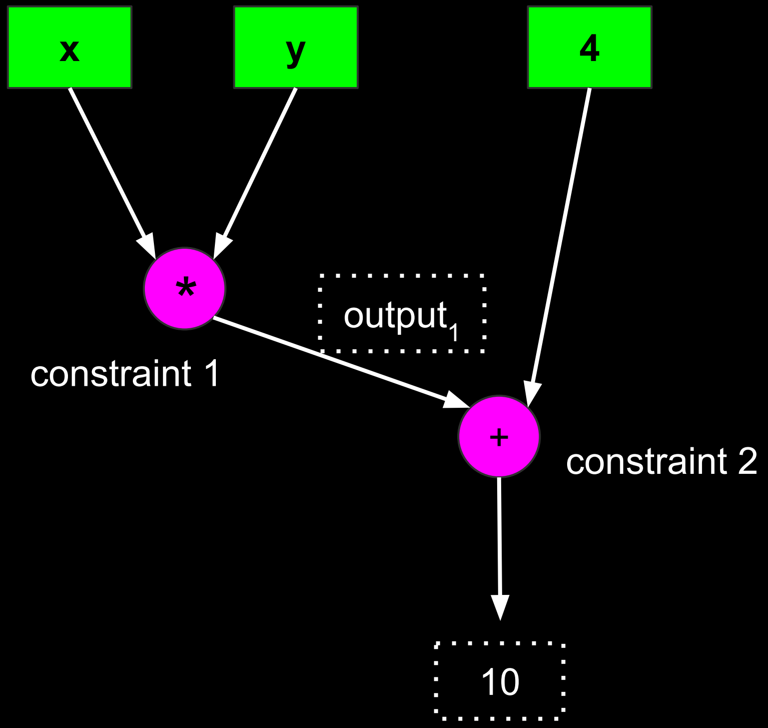
Constraint 2:

$$output_1 + 4 = 10$$

Constraint 3:

Equality constraint

SNARKs - R1CS



Constraint has a **left** input, **right** input and **output**

Such that:
left(L) * **right**(R) = **output**(O)

With 3 vectors:
<L, v> * <R, v> = <O, v>

v is the variable vector
v = [1, x, y, output$_1$]

SNARKs - Arithmetic Circuit

<L, v> * <R, v> = <O, v>

$v = [1, x, y, output_1]$

Constraint 1: $x * y = output_1$

L = [0, 1, 0, 0] // x

R = [0, 0, 1, 0] // y

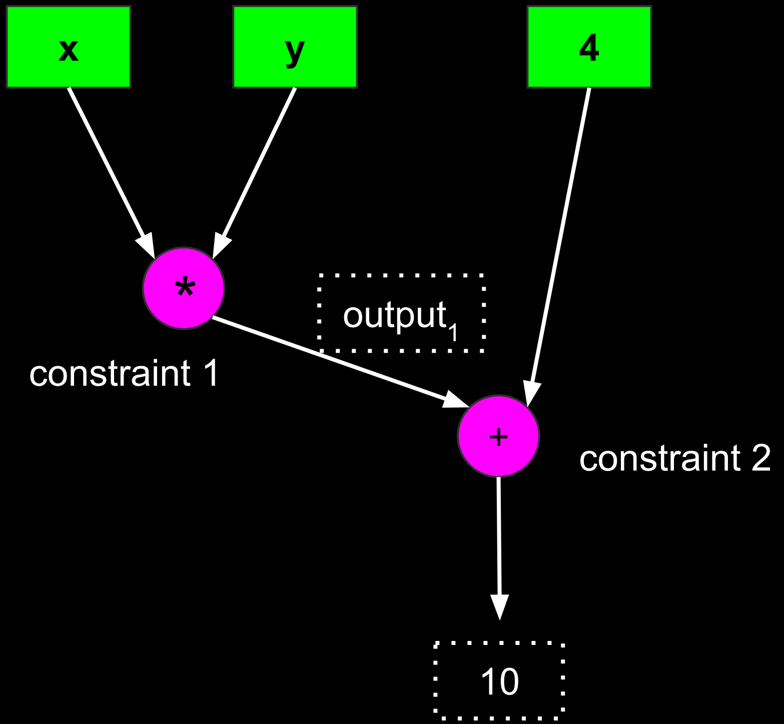
O = [0, 0, 0, 1] // $output_1$

Constraint 2: $(output_1 + 4) * 1 = 10$

L = [4, 0, 0, 1] // $output_1 + 5$

R = [1, 0, 0, 0] // 1

O = [10, 0, 0, 0] // 10



SNARKs - QAP

Create polynomials for each constraint using ***Lagrange interpolation***

L * R = O

Constraint 1:
L = [0, 1, 0, 0] // x
R = [0, 0, 1, 0] // y
O = [0, 0, 0, 1] // $output_1$

Constraint 2:
L = [4, 0, 0, 1] // $output_1$ + 5
R = [1, 0, 0, 0] // 1
O = [10, 0, 0, 0] // 10

$L_1[1] = 0$ (look at 1st constraint, 1st element of L)
$L_1[2] = 4$ (look at 2nd constraint, 1st element of L)

2 points: (1, 0) and (2, 4). We can use Lagrange interpolation to get a polynomial: **4x - 4**

$$\mathbf{L_1(x) = 4x - 4}$$

And we do this for each constraint, for each variable
$\mathbf{L_i(x), R_i(x), O_i(x)}$

SNARKs - QAP

L * **R** = **O**

Constraint 1:
L = [0, 1, 0, 0] // x
R = [0, 0, 1, 0] // y
O = [0, 0, 0, 1] // $output_1$

Constraint 2:
L = [4, 0, 0, 1] // $output_1$ + 5
R = [1, 0, 0, 0] // 1
O = [10, 0, 0, 0] // 10

$L_1(x) = 4x - 4$
$L_2(x) = -1x + 2$
$L_3(x) = 0$
$L_4(x) = 1x - 1$

SNARKs - QAP

L(x) * R(x) = O(x) for x in {1, 2} (since we have 2 constraints)

SNARKs - QAP

$L(x) * R(x) = O(x)$ for x in $\{1, 2\}$ (since we have 2 constraints)

$P = L(x) * R(x) - O(x)$

SNARKs - QAP

$L(x) * R(x) = O(x)$ for x in {1, 2} (since we have 2 constraints)

$P = L(x) * R(x) - O(x) = T(x) * H(x)$

$T(x)$ is a publicly known evaluation used for Verification

SNARKs - QAP

$L(x) * R(x) = O(x)$ for x in {1, 2} (since we have 2 constraints)

$P = L(x) * R(x) - O(x) = T(x) * H(x)$

$T(x)$ is a publicly known evaluation used for Verification

$H(x)$ is provided by the Prover and divides
$L(x) * R(x) - O(x)$ evenly (without remainder)

SNARKs - QAP

$$L(x) * R(x) - O(x) = T(x) * H(x)$$

The Prover would send evaluations of x for **L**, **R**, **O**, and **H** polynomials

But how would we ensure that:

1) This “x” is hidden
2) Prover actually uses its polynomials

SNARKs - QAP

L(s) * R(s) - O(s) = T(s) * H(s)

s is a linear combination of $(g, s \cdot g, \ldots, s^d \cdot g)$ of length d (which corresponds to the max degrees of these polynomials)

This achieves two things:

1) We're forcing the Prover to evaluate s on its polynomials
2) This s is **encrypted**, or "hidden"

SNARKs

How can we evaluate a value that is encrypted?

Ex: Instead of sending *s* plaintext, we can send E(s)

E(s) = sG

(where G is a generator point on a elliptic curve)

SNARKs

We can still do evaluations of E(s), even though it's encrypted (sometimes called homomorphic addition):

Example:

E(3) + E(4) = E(3 + 4) = E(7)

SNARKs

$L(s) * R(s) - O(s) = T(s) * H(s)$

Using encrypted s, the Prover would send back:

$E(L(s)), E(L(s)), E(O(s)), E(L(s)),$

SNARKs

L(s) * R(s) - O(s) = T(s) * H(s)

Using encrypted s, the Prover would send back:

E(L(s)), E(L(s)), E(O(s)), E(L(s)),

But we still don't have *zero-knowledge* as *some* information about the assignment is leaked

SNARKs

$L(s) * R(s) - O(s) = T(s) * H(s)$

Adding zero-knowledge:

The Prover simply adds a form of a blinding factor to the L, R, O evaluations

SNARKs - Protocol

1. **Setup**: E(s) is known to everyone, T(s) is known to everyone **(s itself is thrown away, “toxic waste”)**
2. Prover has **L**, **R**, O and **H** polynomials
3. Prover sends E(**L**(s)), E(**R**(s)), E(O(s)), E(**H**(s))
4. Anyone in the network can verify:
   E(L(s)) * E(R(s)) - E(O(s)) = E(H(s)) * E(T(s))

SNARKs - what’s left

1. This is a huge oversimplification, and we skipped some details regarding “blinding factors”

2. Pairing of Elliptic Curves for the Verifier

3. This went over the Pinocchio Protocol (**PGHR**) 2013
   Better proving systems already out there: **Groth16**

SNARKs - the Future

1. Enormous effort in research for further optimization
    a. Research in pairings
    b. Research in optionality of elliptic curves
    c. Lattice-based SNARKs
    d. And a lot, lot more

Let’s Go Ahead and Build One!

Part 2:
Zokrates