Get these slides ⇨

Zero-knowledge-proofs - part 1

Engineering workshop - Dec 2025 (Ireland)

Pawel Jakubas

Plan of the tutorial

Plan of the tutorial

Let’s get a little deeper than usual and understand what main building blocks of ZKP looks like

This tutorial will focus on

Plan of the tutorial

Let's get a little deeper than usual and understand what main building blocks of ZKP looks like

This tutorial will focus on

1. sketching the landscape of what we want to understand during 3-4 parts

2.

Plan of the tutorial

Let's get a little deeper than usual and understand what main building blocks of ZKP looks like

This tutorial will focus on

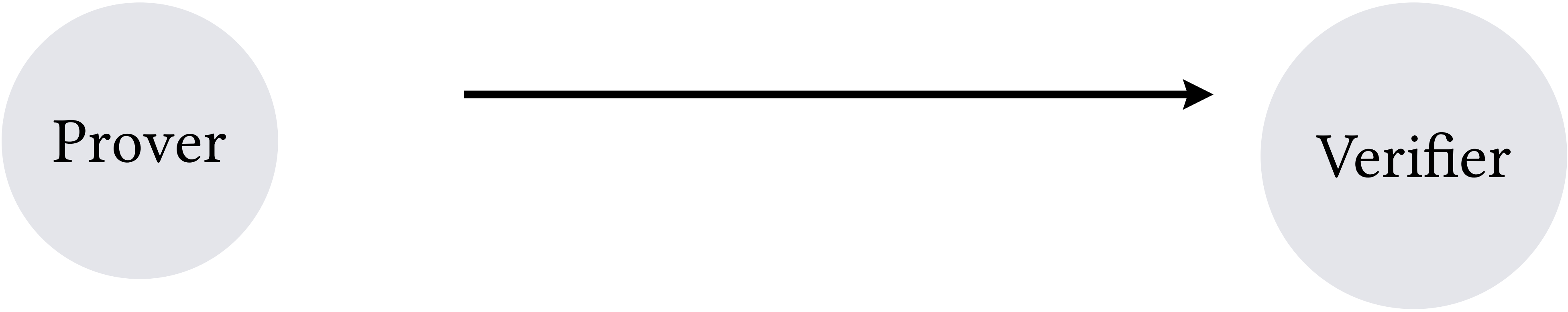
1. sketching the landscape of what we want to understand during 3-4 parts

2. cover the first part in some detail **elliptic curves**

Verifiable computing vs ZKP (1)

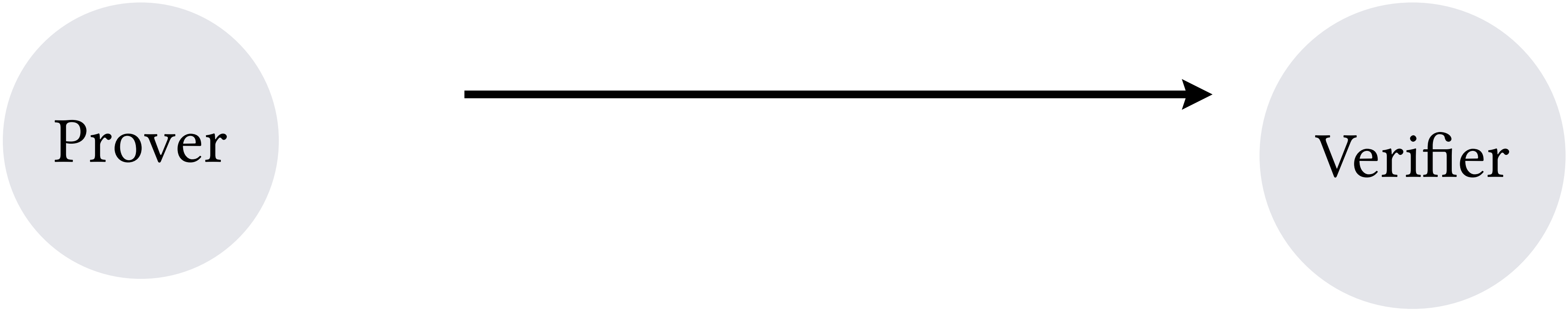
Verifiable computing vs ZKP (1)

There is **asymmetry** built into those systems. It is much easy to get public key from secret. But not the other way



Verifiable computing vs ZKP (1)

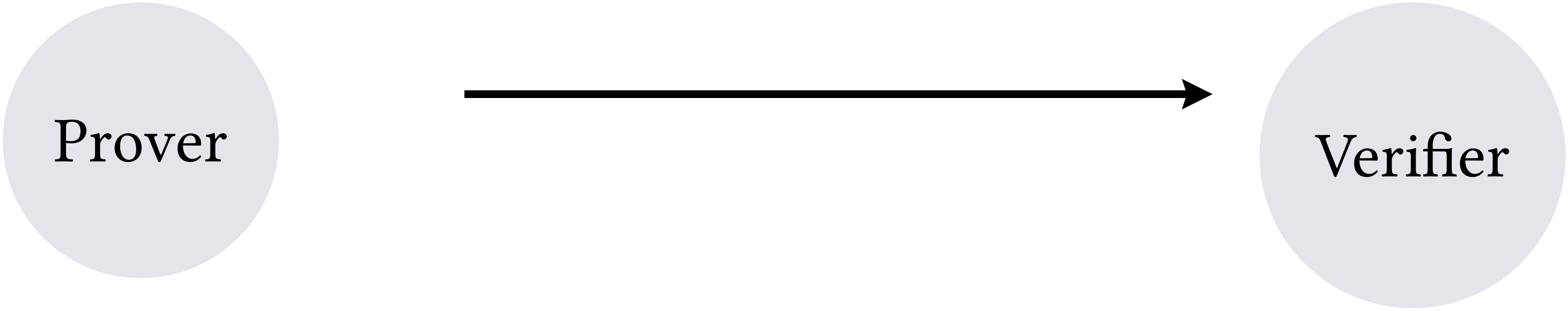
There is **asymmetry** built into those systems. It is much easy to get public key from secret. But not the other way



secret -> (easy) -> public

Verifiable computing vs ZKP (1)

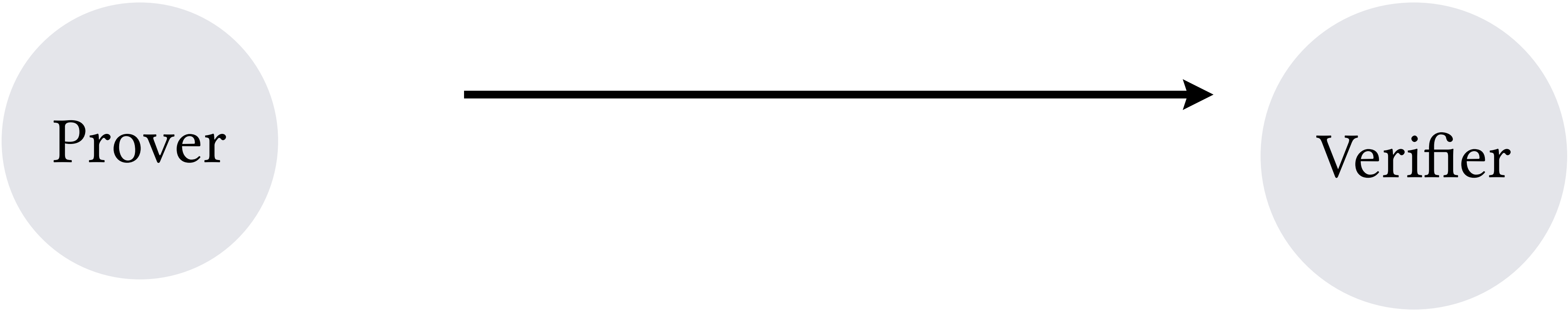
There is **asymmetry** built into those systems. It is much easy to get public key from secret. But not the other way



proof + public data shared

Verifiable computing vs ZKP (1)

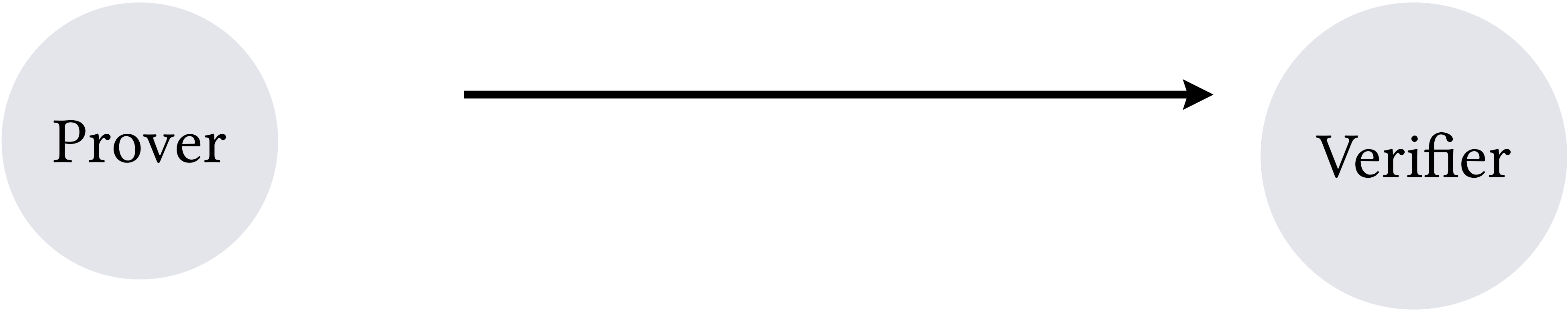
There is **asymmetry** built into those systems. It is much easy to get public key from secret. But not the other way



public -> (very hard) -> secret

Verifiable computing vs ZKP (1)

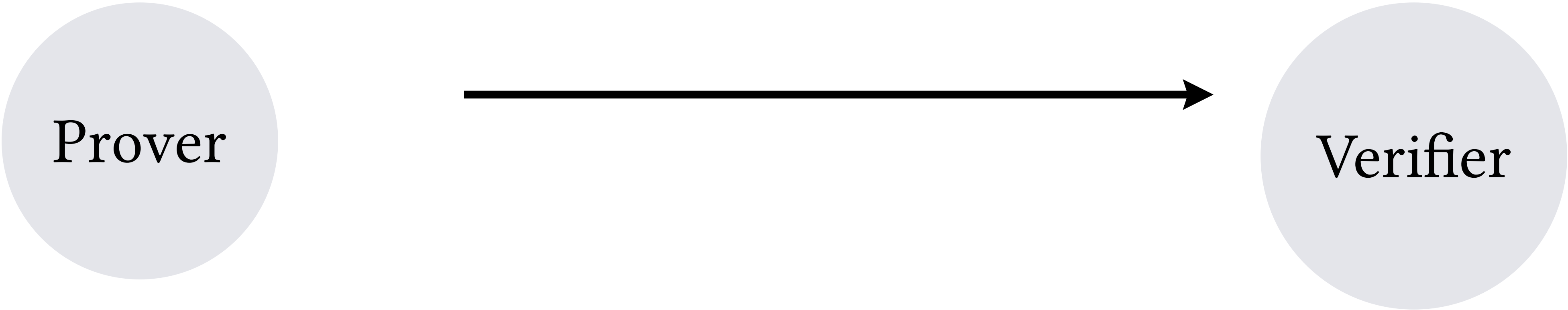
There is **asymmetry** built into those systems. It is much easy to get public key from secret. But not the other way



proof verified using public data

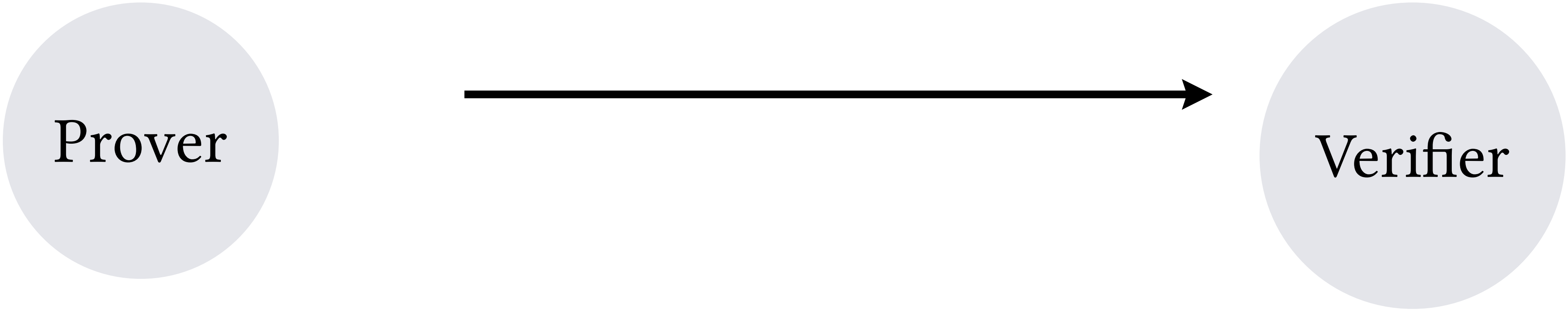
Verifiable computing vs ZKP (2)

There is **asymmetry** built into those systems. It is much quicker to verify than prove something.



Verifiable computing vs ZKP (2)

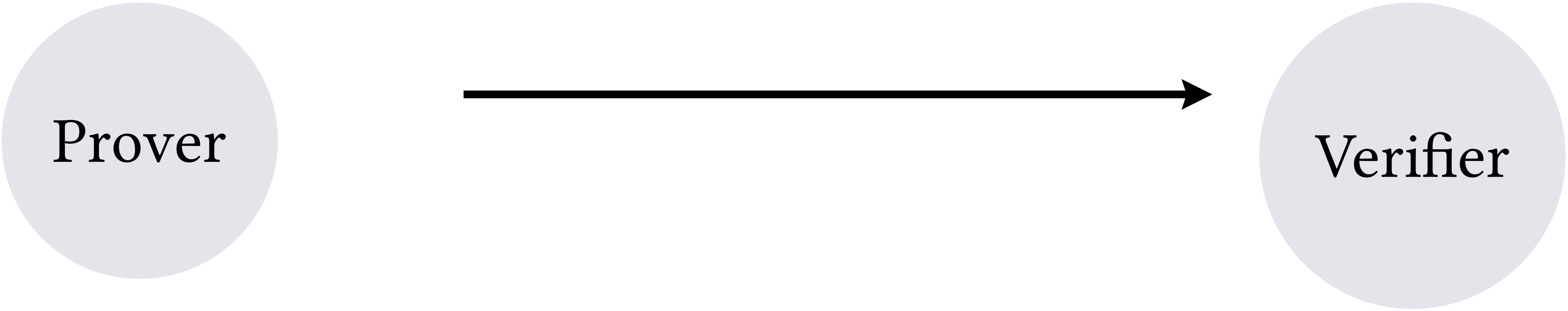
There is **asymmetry** built into those systems. It is much quicker to verify than prove something.



O(n) off-chain

Verifiable computing vs ZKP (2)

There is **asymmetry** built into those systems. It is much quicker to verify than prove something.

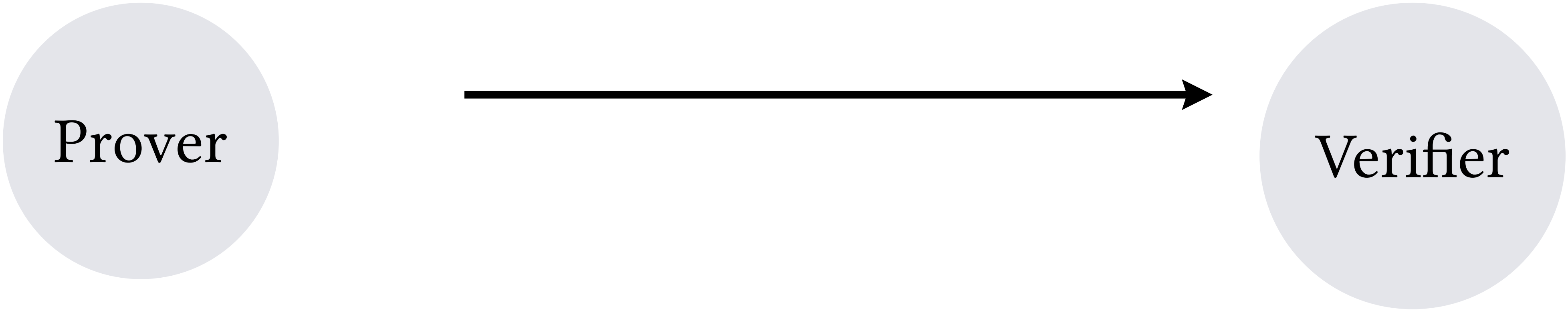


O(n) off-chain

O(log n) on-chain

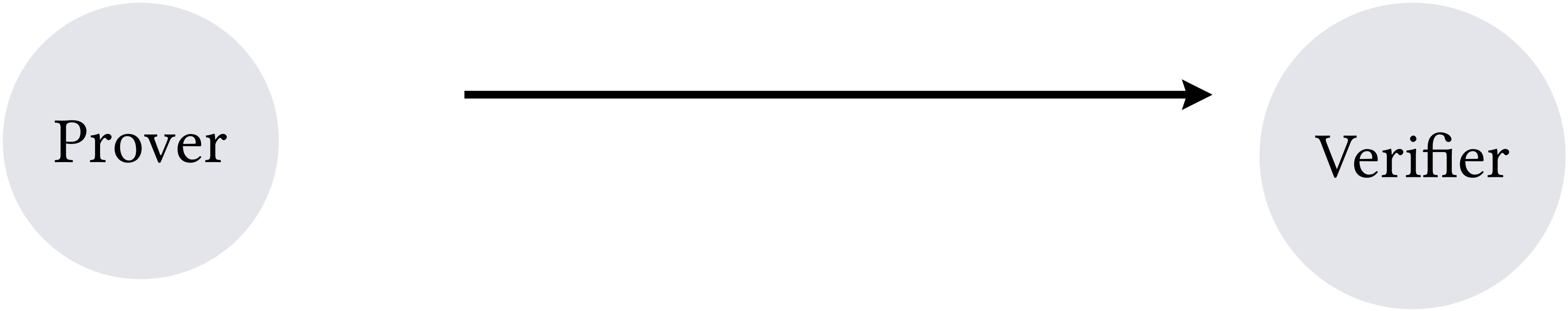
ZKP (3)

Data sent to verifier is compressed, and can be hidden



ZKP (3)

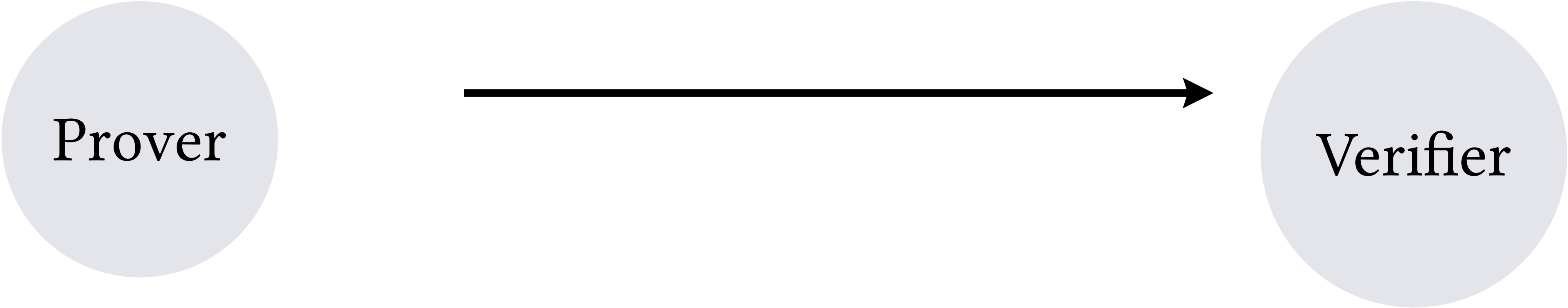
Data sent to verifier is compressed, and can be hidden



size: n

ZKP (3)

Data sent to verifier is compressed, and can be hidden

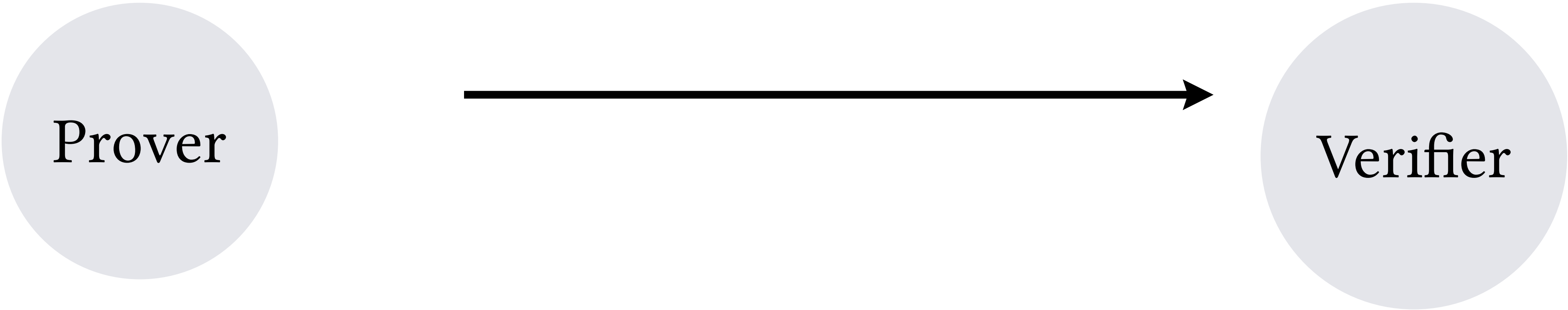


size: n

size: at least log n

ZKP (3)

Data sent to verifier is compressed, and can be hidden



size: n

size: at least log n

size: at least log n

Modular arithmetics (1)

It is about integers.

Modular arithmetics (1)

It is about integers.

Let’s assume we arithmetics **mod 8**. It means the possible values are 0,1,2,3,4,5,6,7. if we move below or above we need to wrap up.

Modular arithmetics (1)

$$3 + 3 \bmod 8 = 6 \bmod 8$$

$$10 \bmod 8 = 2 \bmod 8$$

$$5 + 5 \bmod 8 = 2 \bmod 8$$

$$5 \cdot 5 \bmod 8 = 25 \bmod 8 = (3 \cdot 8 + 1) \bmod 8 = 1 \bmod 8$$

Modular arithmetics (1)

$$3 + 3 \bmod 8 = 6 \bmod 8$$

$$10 \bmod 8 = 2 \bmod 8$$

$$5 + 5 \bmod 8 = 2 \bmod 8$$

$$5 \cdot 5 \bmod 8 = 25 \bmod 8 = (3 \cdot 8 + 1) \bmod 8 = 1 \bmod 8$$

congruent groups

Modular arithmetics (2)

```
addition mod 8      multiplication mod 8
0 1 2 3 4 5 6 7   1 2 3 4 5 6 7
1 2 3 4 5 6 7 0   2 4 6 0 2 4 6
2 3 4 5 6 7 0 1   3 6 1 4 7 2 5
3 4 5 6 7 0 1 2   4 0 4 0 4 0 4
4 5 6 7 0 1 2 3   5 2 7 4 1 6 3
5 6 7 0 1 2 3 4   6 4 2 0 6 4 2
6 7 0 1 2 3 4 5   7 6 5 4 3 2 1
7 0 1 2 3 4 5 6
```

Modular arithmetics (2)

```
addition mod 8      multiplication mod 8          Let's solve the eq in mod 8:
0 1 2 3 4 5 6 7     1 2 3 4 5 6 7                 9(2x + 7) − 6 ≡ 2x + 6
1 2 3 4 5 6 7 0     2 4 6 0 2 4 6
2 3 4 5 6 7 0 1     3 6 1 4 7 2 5
3 4 5 6 7 0 1 2     4 0 4 0 4 0 4
4 5 6 7 0 1 2 3     5 2 7 4 1 6 3
5 6 7 0 1 2 3 4     6 4 2 0 6 4 2
6 7 0 1 2 3 4 5     7 6 5 4 3 2 1
7 0 1 2 3 4 5 6
```

Modular arithmetics (2)

```
addition mod 8
0 1 2 3 4 5 6 7
1 2 3 4 5 6 7 0
2 3 4 5 6 7 0 1
3 4 5 6 7 0 1 2
4 5 6 7 0 1 2 3
5 6 7 0 1 2 3 4
6 7 0 1 2 3 4 5
7 0 1 2 3 4 5 6
```

```
multiplication mod 8
1 2 3 4 5 6 7
2 4 6 0 2 4 6
3 6 1 4 7 2 5
4 0 4 0 4 0 4
5 2 7 4 1 6 3
6 4 2 0 6 4 2
7 6 5 4 3 2 1
```

```
Let's solve the eq in mod 8:
9(2x + 7) − 6 ≡ 2x + 6
19*2x + 19*7 - 6 ≡ 2x + 6
```

Modular arithmetics (2)

```
addition mod 8      multiplication mod 8         Let's solve the eq in mod 8:
0 1 2 3 4 5 6 7     1 2 3 4 5 6 7               9(2x + 7) − 6 ≡ 2x + 6
1 2 3 4 5 6 7 0     2 4 6 0 2 4 6               19*2x + 19*7 - 6 ≡ 2x + 6
2 3 4 5 6 7 0 1     3 6 1 4 7 2 5               38x + 133 - 6 ≡ 2x + 6  # 133 mod 8 = 5
3 4 5 6 7 0 1 2     4 0 4 0 4 0 4
4 5 6 7 0 1 2 3     5 2 7 4 1 6 3
5 6 7 0 1 2 3 4     6 4 2 0 6 4 2
6 7 0 1 2 3 4 5     7 6 5 4 3 2 1
7 0 1 2 3 4 5 6
```

Modular arithmetics (2)

```
addition mod 8      multiplication mod 8          Let's solve the eq in mod 8:
0 1 2 3 4 5 6 7     1 2 3 4 5 6 7              9(2x + 7) − 6 ≡ 2x + 6
1 2 3 4 5 6 7 0     2 4 6 0 2 4 6              19*2x + 19*7 - 6 ≡ 2x + 6
2 3 4 5 6 7 0 1     3 6 1 4 7 2 5              38x + 133 - 6 ≡ 2x + 6  # 133 mod 8 = 5
3 4 5 6 7 0 1 2     4 0 4 0 4 0 4              6x + 5 - 6 ≡ 2x + 6
4 5 6 7 0 1 2 3     5 2 7 4 1 6 3
5 6 7 0 1 2 3 4     6 4 2 0 6 4 2
6 7 0 1 2 3 4 5     7 6 5 4 3 2 1
7 0 1 2 3 4 5 6
```

Modular arithmetics (2)

```
addition mod 8     multiplication mod 8          Let's solve the eq in mod 8:
0 1 2 3 4 5 6 7    1 2 3 4 5 6 7             9(2x + 7) − 6 ≡ 2x + 6
1 2 3 4 5 6 7 0    2 4 6 0 2 4 6             19*2x + 19*7 - 6 ≡ 2x + 6
2 3 4 5 6 7 0 1    3 6 1 4 7 2 5             38x + 133 - 6 ≡ 2x + 6  # 133 mod 8 = 5
3 4 5 6 7 0 1 2    4 0 4 0 4 0 4             6x + 5 - 6 ≡ 2x + 6
4 5 6 7 0 1 2 3    5 2 7 4 1 6 3             6x + 5 - 6 + 6 ≡ 2x + 6 + 6
5 6 7 0 1 2 3 4    6 4 2 0 6 4 2
6 7 0 1 2 3 4 5    7 6 5 4 3 2 1
7 0 1 2 3 4 5 6
```

Modular arithmetics (2)

```
addition mod 8     multiplication mod 8          Let's solve the eq in mod 8:
0 1 2 3 4 5 6 7    1 2 3 4 5 6 7                 9(2x + 7) − 6 ≡ 2x + 6
1 2 3 4 5 6 7 0    2 4 6 0 2 4 6                 19*2x + 19*7 - 6 ≡ 2x + 6
2 3 4 5 6 7 0 1    3 6 1 4 7 2 5                 38x + 133 - 6 ≡ 2x + 6  # 133 mod 8 = 5
3 4 5 6 7 0 1 2    4 0 4 0 4 0 4                 6x + 5 - 6 ≡ 2x + 6
4 5 6 7 0 1 2 3    5 2 7 4 1 6 3                 6x + 5 - 6 + 6 ≡ 2x + 6 + 6
5 6 7 0 1 2 3 4    6 4 2 0 6 4 2                 6x + 5 ≡ 2x + 4  # 12 mod 8 = 4
6 7 0 1 2 3 4 5    7 6 5 4 3 2 1                 6x - 2x + 5 - 5 ≡ 2x - 2x + 4 - 5
7 0 1 2 3 4 5 6
```

Modular arithmetics (2)

```
addition mod 8      multiplication mod 8        Let's solve the eq in mod 8:
0 1 2 3 4 5 6 7     1 2 3 4 5 6 7           9(2x + 7) − 6 ≡ 2x + 6
1 2 3 4 5 6 7 0     2 4 6 0 2 4 6           19*2x + 19*7 - 6 ≡ 2x + 6
2 3 4 5 6 7 0 1     3 6 1 4 7 2 5           38x + 133 - 6 ≡ 2x + 6  # 133 mod 8 = 5
3 4 5 6 7 0 1 2     4 0 4 0 4 0 4           6x + 5 - 6 ≡ 2x + 6
4 5 6 7 0 1 2 3     5 2 7 4 1 6 3           6x + 5 - 6 + 6 ≡ 2x + 6 + 6
5 6 7 0 1 2 3 4     6 4 2 0 6 4 2           6x + 5 ≡ 2x + 4  # 12 mod 8 = 4
6 7 0 1 2 3 4 5     7 6 5 4 3 2 1           6x - 2x + 5 - 5 ≡ 2x - 2x + 4 - 5
7 0 1 2 3 4 5 6                         4x ≡ 7  # -1 mod 8 = 7
```

Now we do **NOT have multiplication inverse** for 4, ie. we cannot divide by 4 in modulo 8, ie. solve this equation We have only multiplication inverse for 1 which is 1; 3 which is 3; 5 which is 5, and 7 which is 7.

Modular arithmetics (3)

addition mod 11

0	1	2	3	4	5	6	7	8	9	10
1	2	3	4	5	6	7	8	9	10	0
2	3	4	5	6	7	8	9	10	0	1
3	4	5	6	7	8	9	10	0	1	2
4	5	6	7	8	9	10	0	1	2	3
5	6	7	8	9	10	0	1	2	3	4
6	7	8	9	10	0	1	2	3	4	5
7	8	9	10	0	1	2	3	4	5	6
8	9	10	0	1	2	3	4	5	6	7
9	10	0	1	2	3	4	5	6	7	8
10	0	1	2	3	4	5	6	7	8	9

multiplication mod 11

1	2	3	4	5	6	7	8	9	10
2	4	6	8	10	1	3	5	7	9
3	6	9	1	4	7	10	2	5	8
4	8	1	5	9	2	6	10	3	7
5	10	4	9	3	8	2	7	1	6
6	1	7	2	8	3	9	4	10	5
7	3	10	6	2	9	5	1	8	4
8	5	2	10	7	4	1	9	6	3
9	7	5	3	1	10	8	6	4	2
10	9	8	7	6	5	4	3	2	1

Modular arithmetics (3)

addition mod 11

0	1	2	3	4	5	6	7	8	9	10
1	2	3	4	5	6	7	8	9	10	0
2	3	4	5	6	7	8	9	10	0	1
3	4	5	6	7	8	9	10	0	1	2
4	5	6	7	8	9	10	0	1	2	3
5	6	7	8	9	10	0	1	2	3	4
6	7	8	9	10	0	1	2	3	4	5
7	8	9	10	0	1	2	3	4	5	6
8	9	10	0	1	2	3	4	5	6	7
9	10	0	1	2	3	4	5	6	7	8
10	0	1	2	3	4	5	6	7	8	9

multiplication mod 11

1	2	3	4	5	6	7	8	9	10
2	4	6	8	10	1	3	5	7	9
3	6	9	1	4	7	10	2	5	8
4	8	1	5	9	2	6	10	3	7
5	10	4	9	3	8	2	7	1	6
6	1	7	2	8	3	9	4	10	5
7	3	10	6	2	9	5	1	8	4
8	5	2	10	7	4	1	9	6	3
9	7	5	3	1	10	8	6	4	2
10	9	8	7	6	5	4	3	2	1

Let's solve in mod 11:

$19(2x + 7) - 6 \equiv 2x + 6$

Modular arithmetics (3)

addition mod 11

0	1	2	3	4	5	6	7	8	9	10
1	2	3	4	5	6	7	8	9	10	0
2	3	4	5	6	7	8	9	10	0	1
3	4	5	6	7	8	9	10	0	1	2
4	5	6	7	8	9	10	0	1	2	3
5	6	7	8	9	10	0	1	2	3	4
6	7	8	9	10	0	1	2	3	4	5
7	8	9	10	0	1	2	3	4	5	6
8	9	10	0	1	2	3	4	5	6	7
9	10	0	1	2	3	4	5	6	7	8
10	0	1	2	3	4	5	6	7	8	9

multiplication mod 11

1	2	3	4	5	6	7	8	9	10
2	4	6	8	10	1	3	5	7	9
3	6	9	1	4	7	10	2	5	8
4	8	1	5	9	2	6	10	3	7
5	10	4	9	3	8	2	7	1	6
6	1	7	2	8	3	9	4	10	5
7	3	10	6	2	9	5	1	8	4
8	5	2	10	7	4	1	9	6	3
9	7	5	3	1	10	8	6	4	2
10	9	8	7	6	5	4	3	2	1

Let's solve in mod 11:

$19(2x + 7) - 6 \equiv 2x + 6$

$19{*}2x + 19{*}7 - 6 \equiv 2x + 6$

Modular arithmetics (3)

addition mod 11

0	1	2	3	4	5	6	7	8	9	10
1	2	3	4	5	6	7	8	9	10	0
2	3	4	5	6	7	8	9	10	0	1
3	4	5	6	7	8	9	10	0	1	2
4	5	6	7	8	9	10	0	1	2	3
5	6	7	8	9	10	0	1	2	3	4
6	7	8	9	10	0	1	2	3	4	5
7	8	9	10	0	1	2	3	4	5	6
8	9	10	0	1	2	3	4	5	6	7
9	10	0	1	2	3	4	5	6	7	8
10	0	1	2	3	4	5	6	7	8	9

multiplication mod 11

1	2	3	4	5	6	7	8	9	10
2	4	6	8	10	1	3	5	7	9
3	6	9	1	4	7	10	2	5	8
4	8	1	5	9	2	6	10	3	7
5	10	4	9	3	8	2	7	1	6
6	1	7	2	8	3	9	4	10	5
7	3	10	6	2	9	5	1	8	4
8	5	2	10	7	4	1	9	6	3
9	7	5	3	1	10	8	6	4	2
10	9	8	7	6	5	4	3	2	1

Let's solve in mod 11:

$19(2x + 7) - 6 \equiv 2x + 6$

$19*2x + 19*7 - 6 \equiv 2x + 6$

$5x + 1 - 6 \equiv 2x + 6$

Modular arithmetics (3)

addition mod 11

0	1	2	3	4	5	6	7	8	9	10
1	2	3	4	5	6	7	8	9	10	0
2	3	4	5	6	7	8	9	10	0	1
3	4	5	6	7	8	9	10	0	1	2
4	5	6	7	8	9	10	0	1	2	3
5	6	7	8	9	10	0	1	2	3	4
6	7	8	9	10	0	1	2	3	4	5
7	8	9	10	0	1	2	3	4	5	6
8	9	10	0	1	2	3	4	5	6	7
9	10	0	1	2	3	4	5	6	7	8
10	0	1	2	3	4	5	6	7	8	9

multiplication mod 11

1	2	3	4	5	6	7	8	9	10
2	4	6	8	10	1	3	5	7	9
3	6	9	1	4	7	10	2	5	8
4	8	1	5	9	2	6	10	3	7
5	10	4	9	3	8	2	7	1	6
6	1	7	2	8	3	9	4	10	5
7	3	10	6	2	9	5	1	8	4
8	5	2	10	7	4	1	9	6	3
9	7	5	3	1	10	8	6	4	2
10	9	8	7	6	5	4	3	2	1

Let's solve in mod 11:

$19(2x + 7) - 6 \equiv 2x + 6$

$19*2x + 19*7 - 6 \equiv 2x + 6$

$5x + 1 - 6 \equiv 2x + 6$

$5x + 1 - 6 + 6 \equiv 2x + 6 + 6$

Modular arithmetics (3)

addition mod 11

0	1	2	3	4	5	6	7	8	9	10
1	2	3	4	5	6	7	8	9	10	0
2	3	4	5	6	7	8	9	10	0	1
3	4	5	6	7	8	9	10	0	1	2
4	5	6	7	8	9	10	0	1	2	3
5	6	7	8	9	10	0	1	2	3	4
6	7	8	9	10	0	1	2	3	4	5
7	8	9	10	0	1	2	3	4	5	6
8	9	10	0	1	2	3	4	5	6	7
9	10	0	1	2	3	4	5	6	7	8
10	0	1	2	3	4	5	6	7	8	9

multiplication mod 11

1	2	3	4	5	6	7	8	9	10
2	4	6	8	10	1	3	5	7	9
3	6	9	1	4	7	10	2	5	8
4	8	1	5	9	2	6	10	3	7
5	10	4	9	3	8	2	7	1	6
6	1	7	2	8	3	9	4	10	5
7	3	10	6	2	9	5	1	8	4
8	5	2	10	7	4	1	9	6	3
9	7	5	3	1	10	8	6	4	2
10	9	8	7	6	5	4	3	2	1

Let's solve in mod 11:

$$19(2x + 7) - 6 \equiv 2x + 6$$

$$19*2x + 19*7 - 6 \equiv 2x + 6$$

$$5x + 1 - 6 \equiv 2x + 6$$

$$5x + 1 - 6 + 6 \equiv 2x + 6 + 6$$

$$5x + 1 \equiv 2x + 1$$

Modular arithmetics (3)

addition mod 11

0	1	2	3	4	5	6	7	8	9	10
1	2	3	4	5	6	7	8	9	10	0
2	3	4	5	6	7	8	9	10	0	1
3	4	5	6	7	8	9	10	0	1	2
4	5	6	7	8	9	10	0	1	2	3
5	6	7	8	9	10	0	1	2	3	4
6	7	8	9	10	0	1	2	3	4	5
7	8	9	10	0	1	2	3	4	5	6
8	9	10	0	1	2	3	4	5	6	7
9	10	0	1	2	3	4	5	6	7	8
10	0	1	2	3	4	5	6	7	8	9

multiplication mod 11

1	2	3	4	5	6	7	8	9	10
2	4	6	8	10	1	3	5	7	9
3	6	9	1	4	7	10	2	5	8
4	8	1	5	9	2	6	10	3	7
5	10	4	9	3	8	2	7	1	6
6	1	7	2	8	3	9	4	10	5
7	3	10	6	2	9	5	1	8	4
8	5	2	10	7	4	1	9	6	3
9	7	5	3	1	10	8	6	4	2
10	9	8	7	6	5	4	3	2	1

Let's solve in mod 11:

$19(2x + 7) - 6 \equiv 2x + 6$

$19*2x + 19*7 - 6 \equiv 2x + 6$

$5x + 1 - 6 \equiv 2x + 6$

$5x + 1 - 6 + 6 \equiv 2x + 6 + 6$

$5x + 1 \equiv 2x + 1$

$x \equiv 0$

We have solution: **{.., −22, −11, 0, 11, 22, ...}** As in each row of mult table there is 1 we have inverse for each congruence group!

Modular arithmetics (4)

addition mod 13

0	1	2	3	4	5	6	7	8	9	10	11	12
1	2	3	4	5	6	7	8	9	10	11	12	0
2	3	4	5	6	7	8	9	10	11	12	0	1
3	4	5	6	7	8	9	10	11	12	0	1	2
4	5	6	7	8	9	10	11	12	0	1	2	3
5	6	7	8	9	10	11	12	0	1	2	3	4
6	7	8	9	10	11	12	0	1	2	3	4	5
7	8	9	10	11	12	0	1	2	3	4	5	6
8	9	10	11	12	0	1	2	3	4	5	6	7
9	10	11	12	0	1	2	3	4	5	6	7	8
10	11	12	0	1	2	3	4	5	6	7	8	9
11	12	0	1	2	3	4	5	6	7	8	9	10
12	0	1	2	3	4	5	6	7	8	9	10	11

multiplication mod 13

1	2	3	4	5	6	7	8	9	10	11	12
2	4	6	8	10	12	1	3	5	7	9	11
3	6	9	12	2	5	8	11	1	4	7	10
4	8	12	3	7	11	2	6	10	1	5	9
5	10	2	7	12	4	9	1	6	11	3	8
6	12	5	11	4	10	3	9	2	8	1	7
7	1	8	2	9	3	10	4	11	5	12	6
8	3	11	6	1	9	4	12	7	2	10	5
9	5	1	10	6	2	11	7	3	12	8	4
10	7	4	1	11	8	5	2	12	9	6	3
11	9	7	5	3	1	12	10	8	6	4	2
12	11	10	9	8	7	6	5	4	3	2	1

As in each row of mult table there is 1 we have inverse for each congruence group! => we want to work with mod **PRIME** as we want inverses!

Modular arithmetics (4)

addition mod 13

0	1	2	3	4	5	6	7	8	9	10	11	12
1	2	3	4	5	6	7	8	9	10	11	12	0
2	3	4	5	6	7	8	9	10	11	12	0	1
3	4	5	6	7	8	9	10	11	12	0	1	2
4	5	6	7	8	9	10	11	12	0	1	2	3
5	6	7	8	9	10	11	12	0	1	2	3	4
6	7	8	9	10	11	12	0	1	2	3	4	5
7	8	9	10	11	12	0	1	2	3	4	5	6
8	9	10	11	12	0	1	2	3	4	5	6	7
9	10	11	12	0	1	2	3	4	5	6	7	8
10	11	12	0	1	2	3	4	5	6	7	8	9
11	12	0	1	2	3	4	5	6	7	8	9	10
12	0	1	2	3	4	5	6	7	8	9	10	11

multiplication mod 13

1	2	3	4	5	6	7	8	9	10	11	12
2	4	6	8	10	12	1	3	5	7	9	11
3	6	9	12	2	5	8	11	1	4	7	10
4	8	12	3	7	11	2	6	10	1	5	9
5	10	2	7	12	4	9	1	6	11	3	8
6	12	5	11	4	10	3	9	2	8	1	7
7	1	8	2	9	3	10	4	11	5	12	6
8	3	11	6	1	9	4	12	7	2	10	5
9	5	1	10	6	2	11	7	3	12	8	4
10	7	4	1	11	8	5	2	12	9	6	3
11	9	7	5	3	1	12	10	8	6	4	2
12	11	10	9	8	7	6	5	4	3	2	1

diagonal:	1	2	3	4	5	6	7	8	9	10	11	12
val:	1	4	9	3	12	10	10	12	3	9	4	1

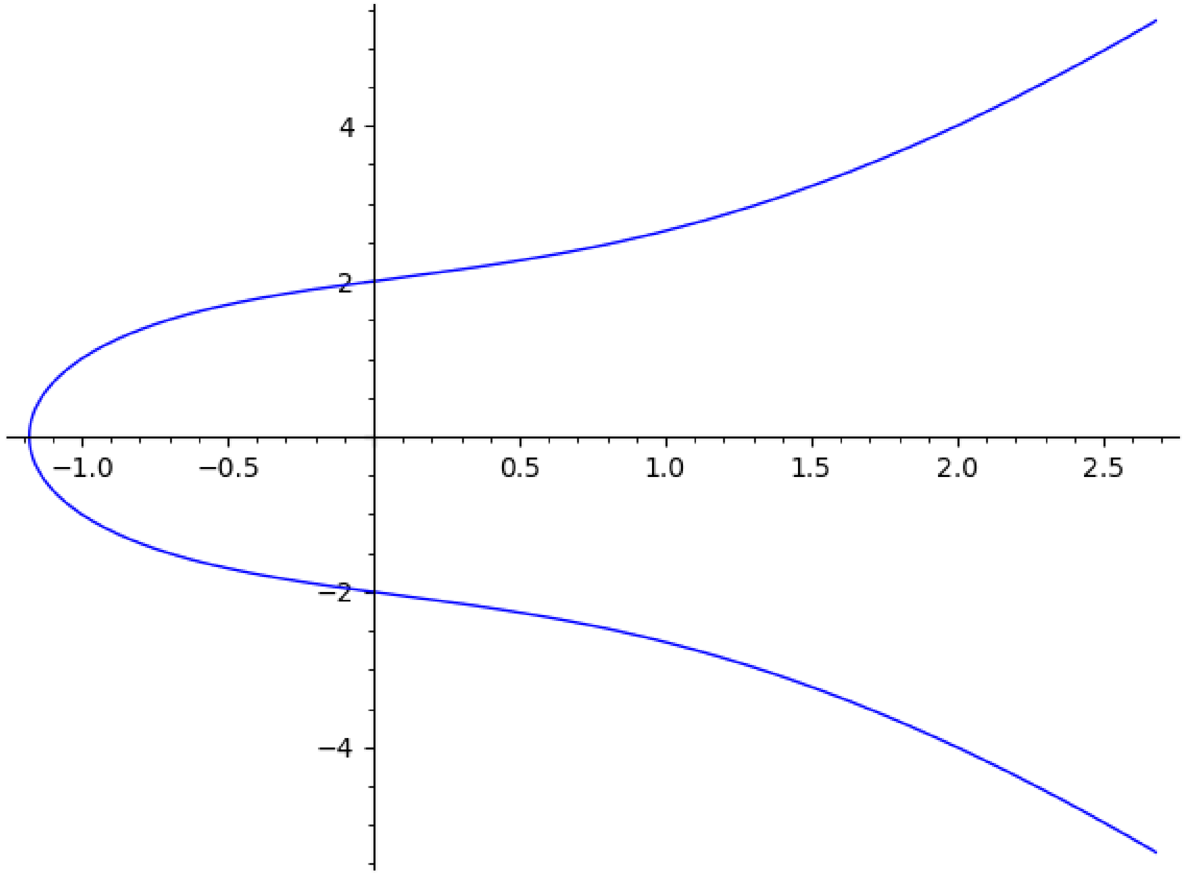
- not always square is possible within modulus
- 1, 3, 4, 9, 10, 12

  are **quadratic residues**

Elliptic curves (1)

```
sage: # Let's plot the following elliptic curve in R
sage: # y^2 == x^3 + 2x + 4 in R

sage: E = EllipticCurve([2,4]);
sage: P = E.plot()
sage: P.save("ellipticR.png")
```



Elliptic curves (2)

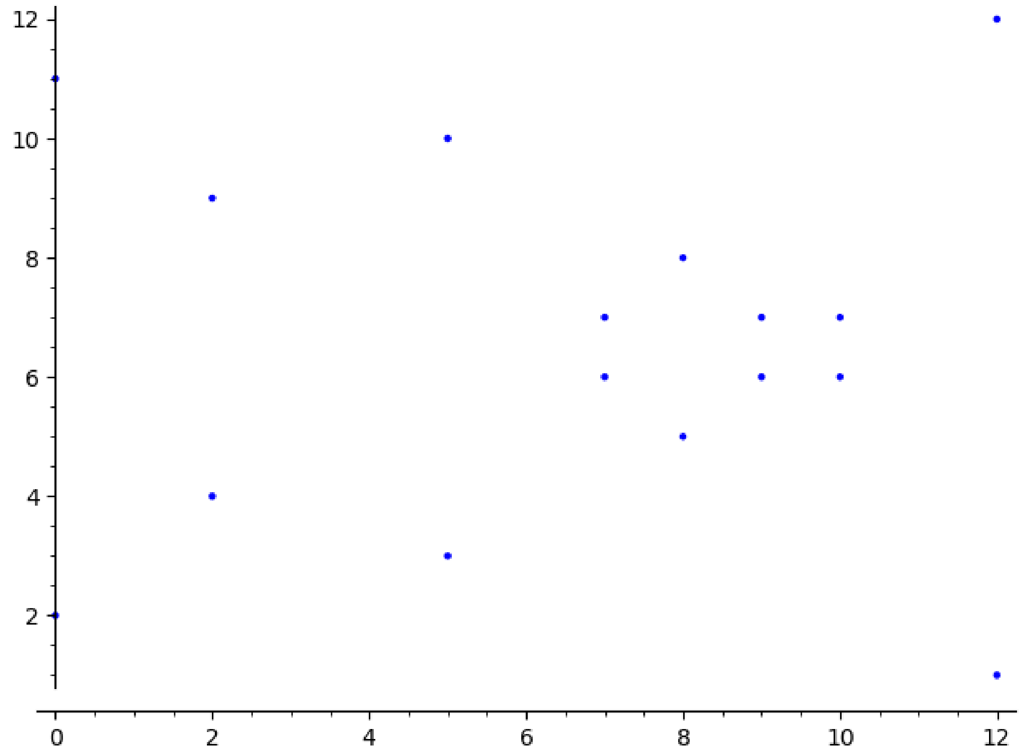
```
y^2 == x^3 + 2x + 4 in mod 13

x=0, y^2= 4 => (0,2) and (0,11) -> two points
x=1, y^2= 1 + 2 + 4 = 7 -> no point
x=2, y^2= 8 + 4 + 4 = 16 mod 13 = 3 -> (2,4) and (2,9)
x=3, y^2= 1 + 6 + 4 = 11 -> no point
x=4, y^2= 12 + 8 + 4 = 24 mod 13 = 11 -> no point
x=5, y^2= 8 + 10 + 4 = 9 -> (5,3) and (5,10)
x=6, y^2= 8 + 12 + 4 = 24 mod 13 = 11 -> no point
x=7, y^2= 5 + 14 + 4 = 23 mod 13 = 10 -> (7,6) and (7,7)
x=8, y^2= 5 + 16 + 4 = 25 mod 13 = 12 -> (8,5) and (8,8)
x=9, y^2= 1 + 18 + 4 = 23 mod 13 = 10 -> (9,6) and (9,7)
x=10, y^2= 12 + 20 + 4 = 36 mod 13 = 10 -> (10,6) and (10,7)
x=11, y^2= 5 + 22 + 4 = 31 mod 13 = 5 -> no point
x=12, y^2= 12 + 24 + 4 = 40 mod 13 = 1 -> (12,1) and (12,12)
```

For some x there are no solutions when we have mod 13, for the rest we have two!

Elliptic curves (3)

```
sage: F13=GF(13)
sage: a = F13(2)
sage: b = F13(4)
sage: # discriminant obeys condition
sage: F13(6)*(F13(4)*a^3+F13(27)*b^2) != F13(0)
True
sage: E = EllipticCurve(F13,[a,b]) # y^2 == x^3 + 2x + 4
sage: P = E(0,2) # 2^2 == 0^3 + 2*0 + 4 mod 13
sage: P.xy()
(0, 2)
sage: INF=E(0)
sage: try:
....:     INF.xy()
....: except ZeroDivisionError:
....:     pass
....:
sage: P = E.plot()
sage: P.save("elliptic13.png")
```



Modular arithmetics (5)

addition mod 13

0	1	2	3	4	5	6	7	8	9	10	11	12
1	2	3	4	5	6	7	8	9	10	11	12	0
2	3	4	5	6	7	8	9	10	11	12	0	1
3	4	5	6	7	8	9	10	11	12	0	1	2
4	5	6	7	8	9	10	11	12	0	1	2	3
5	6	7	8	9	10	11	12	0	1	2	3	4
6	7	8	9	10	11	12	0	1	2	3	4	5
7	8	9	10	11	12	0	1	2	3	4	5	6
8	9	10	11	12	0	1	2	3	4	5	6	7
9	10	11	12	0	1	2	3	4	5	6	7	8
10	11	12	0	1	2	3	4	5	6	7	8	9
11	12	0	1	2	3	4	5	6	7	8	9	10
12	0	1	2	3	4	5	6	7	8	9	10	11

Addition forms a **group** as

- 0 is identity element
- addition is associative op
- addition is closed op
- each element has the inverse

Modular arithmetics (5)

multiplication mod 13

1	2	3	4	5	6	7	8	9	10	11	12
2	4	6	8	10	12	1	3	5	7	9	11
3	6	9	12	2	5	8	11	1	4	7	10
4	8	12	3	7	11	2	6	10	1	5	9
5	10	2	7	12	4	9	1	6	11	3	8
6	12	5	11	4	10	3	9	2	8	1	7
7	1	8	2	9	3	10	4	11	5	12	6
8	3	11	6	1	9	4	12	7	2	10	5
9	5	1	10	6	2	11	7	3	12	8	4
10	7	4	1	11	8	5	2	12	9	6	3
11	9	7	5	3	1	12	10	8	6	4	2
12	11	10	9	8	7	6	5	4	3	2	1

Mult forms a **group** as

- 1 is identity element
- mult is associative op
- mult is closed op
- each element has the inverse (thanks to p being prime)

Modular arithmetics (5)

multiplication mod 13

1	2	3	4	5	6	7	8	9	10	11	12
2	4	6	8	10	12	1	3	5	7	9	11
3	6	9	12	2	5	8	11	1	4	7	10
4	8	12	3	7	11	2	6	10	1	5	9
5	10	2	7	12	4	9	1	6	11	3	8
6	12	5	11	4	10	3	9	2	8	1	7
7	1	8	2	9	3	10	4	11	5	12	6
8	3	11	6	1	9	4	12	7	2	10	5
9	5	1	10	6	2	11	7	3	12	8	4
10	7	4	1	11	8	5	2	12	9	6	3
11	9	7	5	3	1	12	10	8	6	4	2
12	11	10	9	8	7	6	5	4	3	2	1

- 3 . 3 = 9
- 3 . 3 . 3 = 1
- 3 . 3 . 3 . 3 = 3

we can NOT generate EACH element -> 3 is not generator

Modular arithmetics (5)

multiplication mod 13

1	2	3	4	5	6	7	8	9	10	11	12
2	4	6	8	10	12	1	3	5	7	9	11
3	6	9	12	2	5	8	11	1	4	7	10
4	8	12	3	7	11	2	6	10	1	5	9
5	10	2	7	12	4	9	1	6	11	3	8
6	12	5	11	4	10	3	9	2	8	1	7
7	1	8	2	9	3	10	4	11	5	12	6
8	3	11	6	1	9	4	12	7	2	10	5
9	5	1	10	6	2	11	7	3	12	8	4
10	7	4	1	11	8	5	2	12	9	6	3
11	9	7	5	3	1	12	10	8	6	4	2
12	11	10	9	8	7	6	5	4	3	2	1

- 8 . 8 = 12
- 8 . 8 . 8 = 5
- 8 . 8 . 8 . 8 = 1

we can NOT generate EACH element -> 8 is not generator

Modular arithmetics (5)

multiplication mod 13

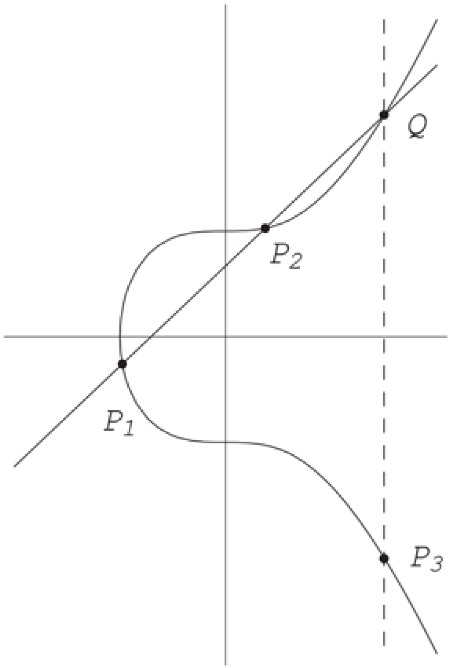
1	2	3	4	5	6	7	8	9	10	11	12
2	4	6	8	10	12	1	3	5	7	9	11
3	6	9	12	2	5	8	11	1	4	7	10
4	8	12	3	7	11	2	6	10	1	5	9
5	10	2	7	12	4	9	1	6	11	3	8
6	12	5	11	4	10	3	9	2	8	1	7
7	1	8	2	9	3	10	4	11	5	12	6
8	3	11	6	1	9	4	12	7	2	10	5
9	5	1	10	6	2	11	7	3	12	8	4
10	7	4	1	11	8	5	2	12	9	6	3
11	9	7	5	3	1	12	10	8	6	4	2
12	11	10	9	8	7	6	5	4	3	2	1

- 2 . 2 = 4
- 2 . 2 . 2 = 8
- 2 . 2 . 2 . 2 = 3
- 2 . 2 . 2 . 2 . 2 = 6
- 2 . 2 . 2 . 2 . 2 . 2 = 12
- 2 . 2 . 2 . 2 . 2 . 2 . 2 = 11
- 2 . 2 . 2 . 2 . 2 . 2 . 2 . 2 = 9
- 2 . 2 . 2 . 2 . 2 . 2 . 2 . 2 . 2 = 5
- 2 . 2 . 2 . 2 . 2 . 2 . 2 . 2 . 2 . 2 = 10
- 2 . 2 . 2 . 2 . 2 . 2 . 2 . 2 . 2 . 2 . 2 = 7
- 2 . 2 . 2 . 2 . 2 . 2 . 2 . 2 . 2 . 2 . 2 . 2 = 1

we can generate EVERY element -> 2 is **generator**, group is **cyclic**

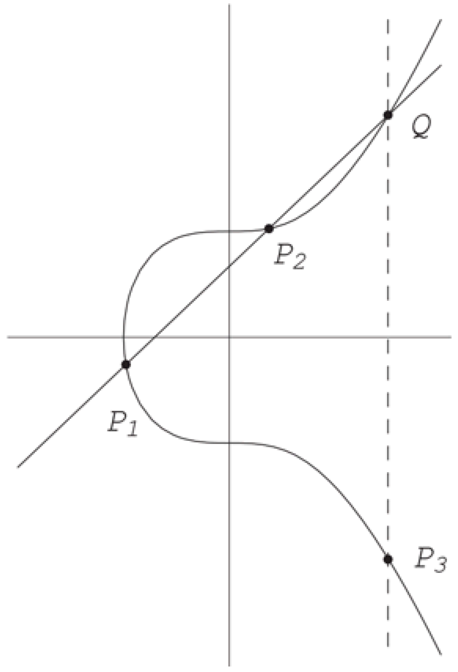
Elliptic curves (4)

Visually addition looks like here for elliptic curves with one remark: it is modulo PRIME



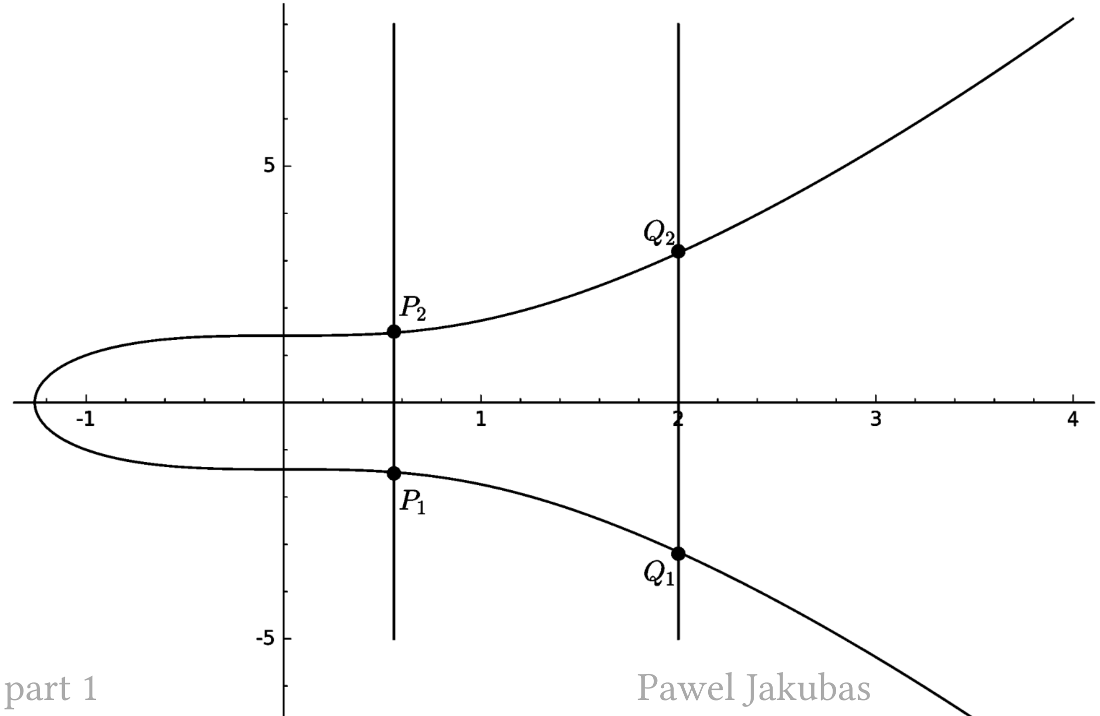
Elliptic curves (4)

It is called **chord-and-tangent** rule and visually is looks like below $P_1 + P_2 = P_3$



Elliptic curves (4)

y=INF is identity element, and because of that $P_1^{-1} = P_2$ $Q_1^{-1} = Q_2$



Elliptic curves (4)

chord-and-tangent rule algebraically is following

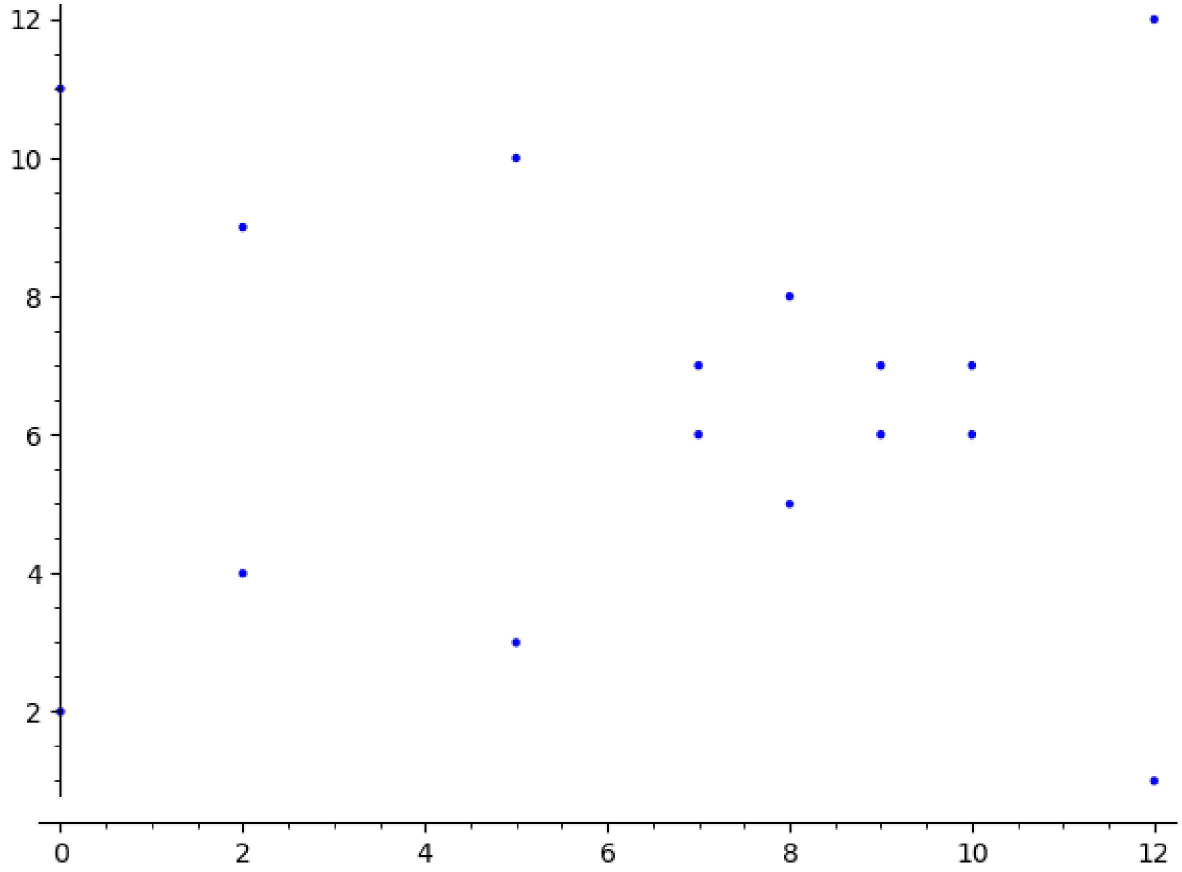
(Tangent Rule) If $P = (x, y)$ with $y \neq 0$, the group law $P \oplus P = (x', y')$ is defined as follows:

$$x' = \left(\frac{3x^2+a}{2y}\right)^2 - 2x \quad , \quad y' = \left(\frac{3x^2+a}{2y}\right)(x - x') - y$$

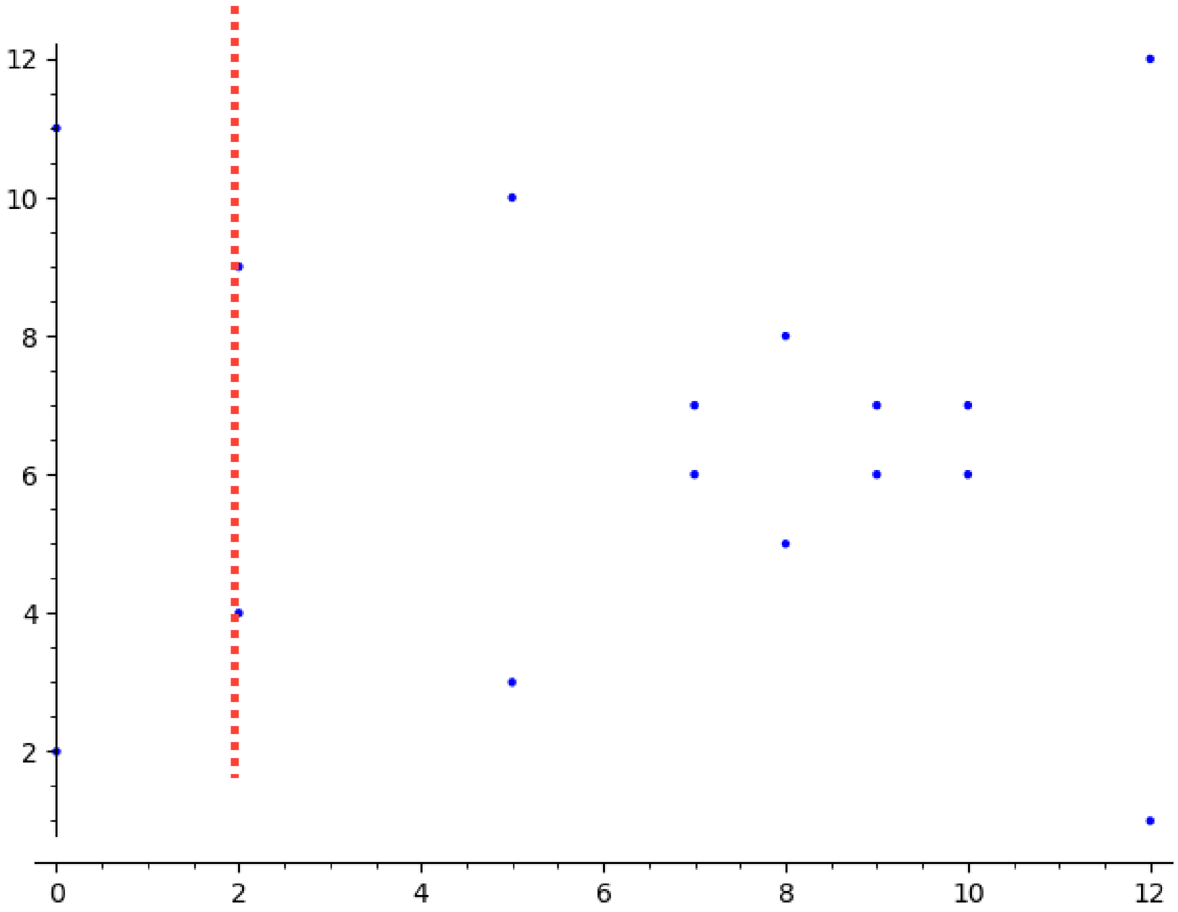
(Chord Rule) If $P = (x_1, y_1)$ and $Q = (x_2, y_2)$ such that $x_1 \neq x_2$, the group law $R = P \oplus Q$ with $R = (x_3, y_3)$ is defined as follows:

$$x_3 = \left(\frac{y_2-y_1}{x_2-x_1}\right)^2 - x_1 - x_2 \quad , \quad y_3 = \left(\frac{y_2-y_1}{x_2-x_1}\right)(x_1 - x_3) - y_1$$

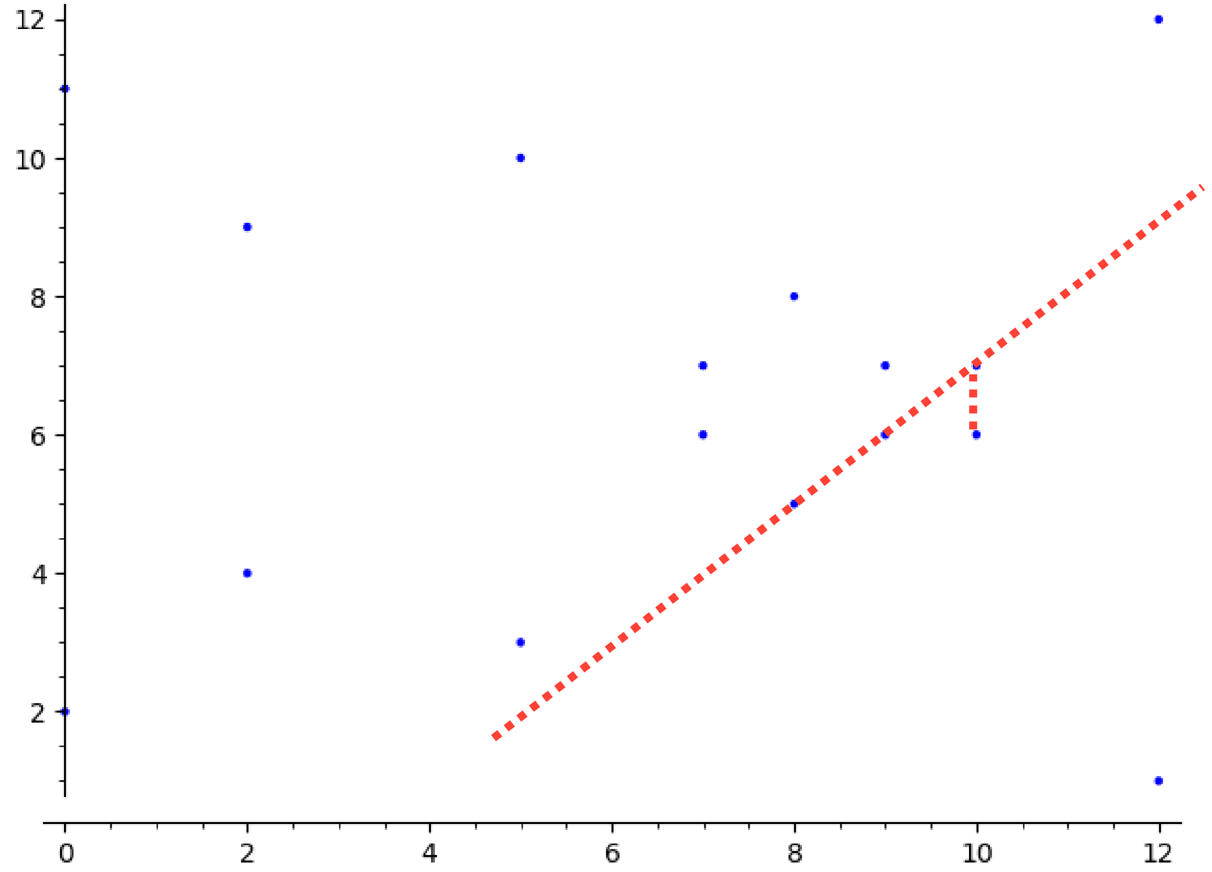
Elliptic curves (5) (mod 13)



Elliptic curves (5) (mod 13)



Elliptic curves (5) (mod 13)



Elliptic curves (6) (mod 13)

```
sage: F13 = GF(13)
sage: a = F13(2)
sage: b = F13(4)
sage: E = EllipticCurve(F13,[a,b]) # y^2 == x^3 + 2x + 4
sage: INF=E(0)
sage: E(2,4) + E(2,9) == INF
True
sage: E(8,5) + E(9,6) == E(10,7)
False
sage: E(8,5) + E(9,6) == E(10,6)
True
```

Elliptic curves (7)

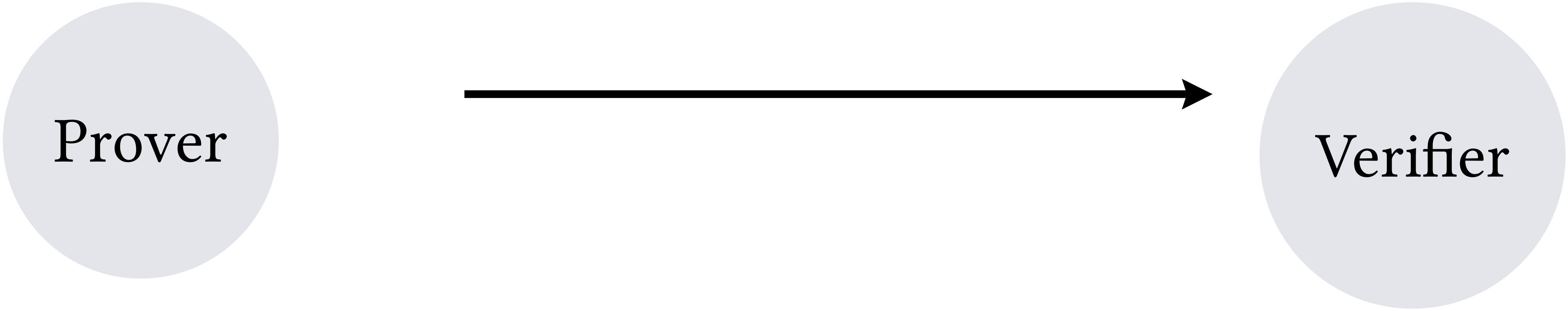
```
sage: # Bitcoin’s secp256k1 curve
sage: # p = 2^256-2^32-977
sage: p =115792089237316195423570985008687907853269984665640564039457584007908834671663
sage: p.is_prime()
True
sage: p.nbits()
256
sage: Fp = GF(p)
sage: secp256k1 = EllipticCurve(Fp,[0,7])
sage: # Base point
sage: gx= 55066263022277343669578718895168534326250603453777594175500187360389116729240L
sage: gy= 32670510020758816978083085130507043184471273380659243275938904335757337482424L
sage: G = secp256k1(Fp(gx), Fp(gy))
```

Elliptic curves (8)

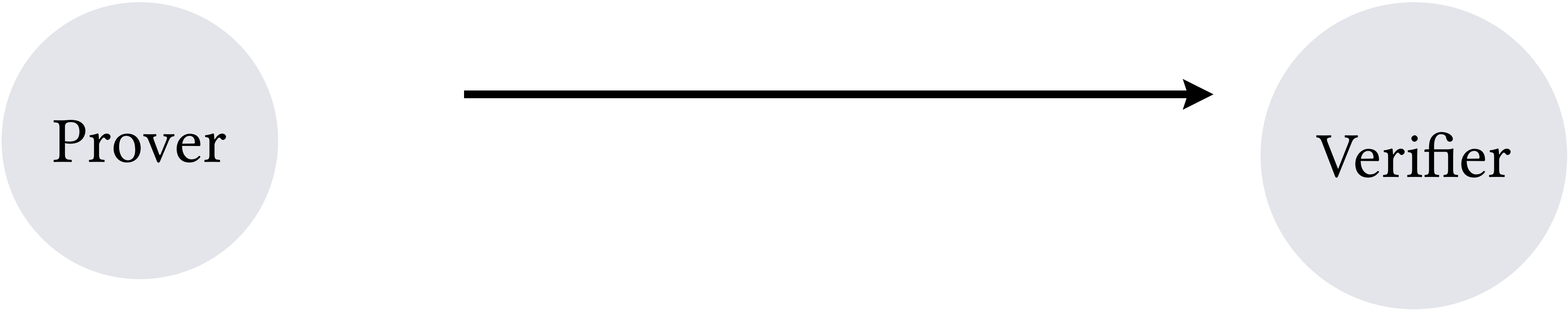
```
sage: # we have x + y = 9 to solve
sage: # PROVER provided a solution (x=2, y=7) and has the proof for it
sage: #
sage: xHidden = 2*G
sage: yHidden = 7*G
sage:
sage: # VERIFIER knows 9 which is public knowledge and gets solution hidden in POINTS
sage: rhsPoint = 9*G
sage: rhsPoint == xHidden + yHidden
True

sage: xHidden
(89565891926547004231252920425935692360644145829622209833684329913297188986597 :
12158399299693830322967808612713398636155367887041628176798871954788371653930 : 1)
```

Elliptic curves (8)



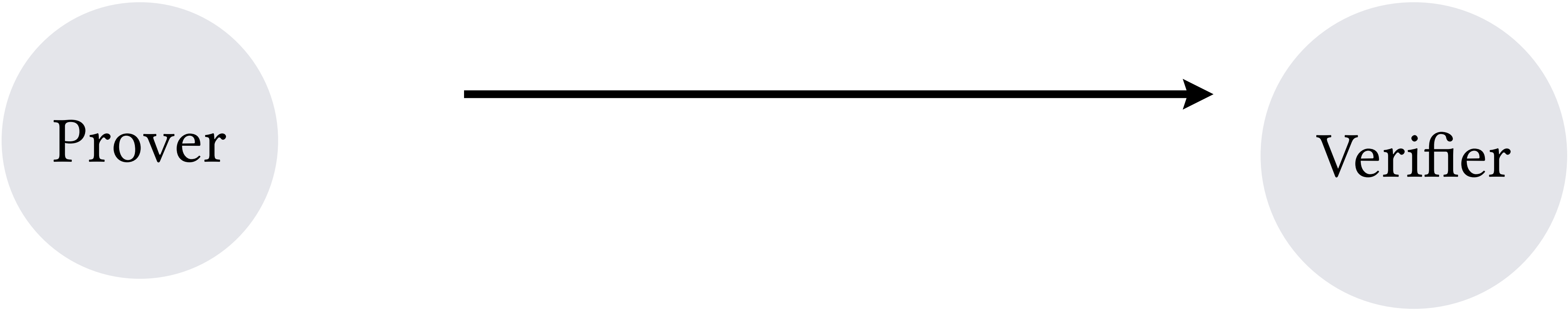
Elliptic curves (8)



x=2, y=7 and sends xG and yG

can check **xG + yG = 9G**, but cannot retrieve x and y

Elliptic curves (8)



x=2, y=7 and sends xG and yG

can check **xG + yG = 9G**, but cannot retrieve x and y

!homomorphic encryption preserves operations!

Elliptic curves (9)

At this moment we can solve problems that are linear, meaning can be expressed as set of linear expressions:

$$a_{11} * x_1 + a_{12} * x_2 + \ldots = b_1$$

$$a_{21} * x_1 + a_{22} * x_2 + \ldots = b_2$$

$$\ldots$$

$$a_{\mathrm{n}1} * x_1 + a_{\mathrm{n}2} * x_2 + \ldots = b_{\mathrm{n}}$$

but we cannot solve:

$$\mathbf{xy = 9}$$

Elliptic curves (9)

At this moment we can solve problems that are linear, meaning can be expressed as set of linear expressions:

$$a_{11} * x_1 + a_{12} * x_2 + \ldots = b_1$$

$$a_{21} * x_1 + a_{22} * x_2 + \ldots = b_2$$

$$\ldots$$

$$a_{\mathrm{n}1} * x_1 + a_{\mathrm{n}2} * x_2 + \ldots = b_{\mathrm{n}}$$

but we cannot solve:

xy = 9

=> pairings

Pairings of elliptic curves (1)

We are going to use TWO groups together
$(G_1, G_2) = G_T$
G_1 elliptic curve point => 2 numbers
G_2 elliptic curve point over an extended field in the form of polynomials **aw + b** and **a'w + b'** => 4 numbers

For $A \in G_1$, $B \in G_2$ and $C \in G_T$

G_T is multiplicative
e(A,B)=C => e(A^x, B^y)= C^{xy}

Pairings of elliptic curves (2)

G_1 has generator A
G_2 has generator B
C is pairing e(G_1, G_2)=g
xy=12
x=4 and y=3
e(4A, 3B)=C^{12}

G_1 and G_2 are elliptic groups
G_T is multiplicative group of an extensive field

$G_1 = G_2$ symmetric
$G_1 \neq G_2$ antisymmetric (used in production due to performance)

Pairings of elliptic curves (3)

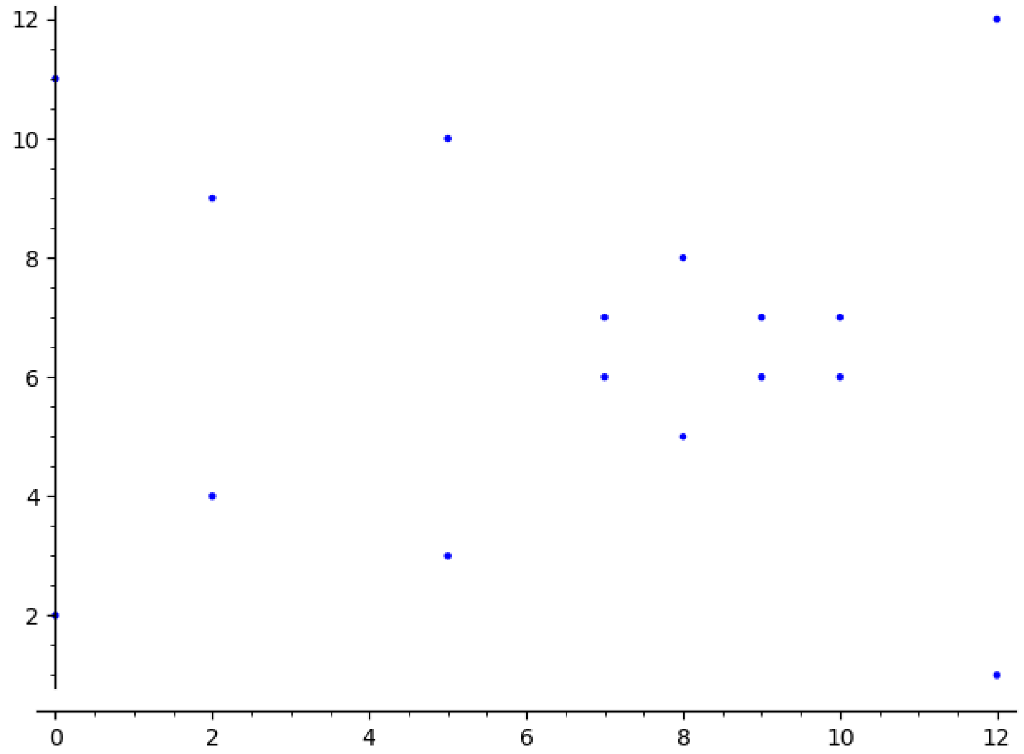
Every elliptic curve gives rise to a pairing map

BUT

not every such pairing can be efficiently computed

=>

embedding degree of a curve



Pairings of elliptic curves (3)

embedding degree of a curve **k**

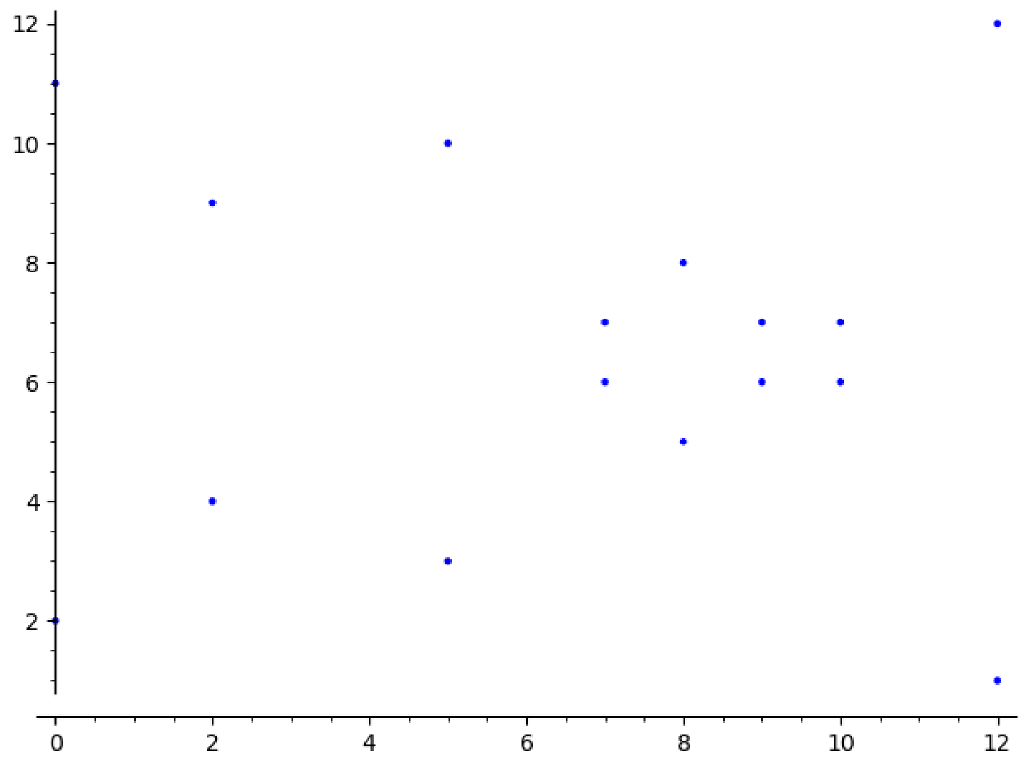
8 pairs + identity (INF) = 17

=>

order r = 17

p = 13

$$r \mid p^k - 1$$



Pairings of elliptic curves (3)

embedding degree of a curve **k**

8 pairs + identity (INF) = 17

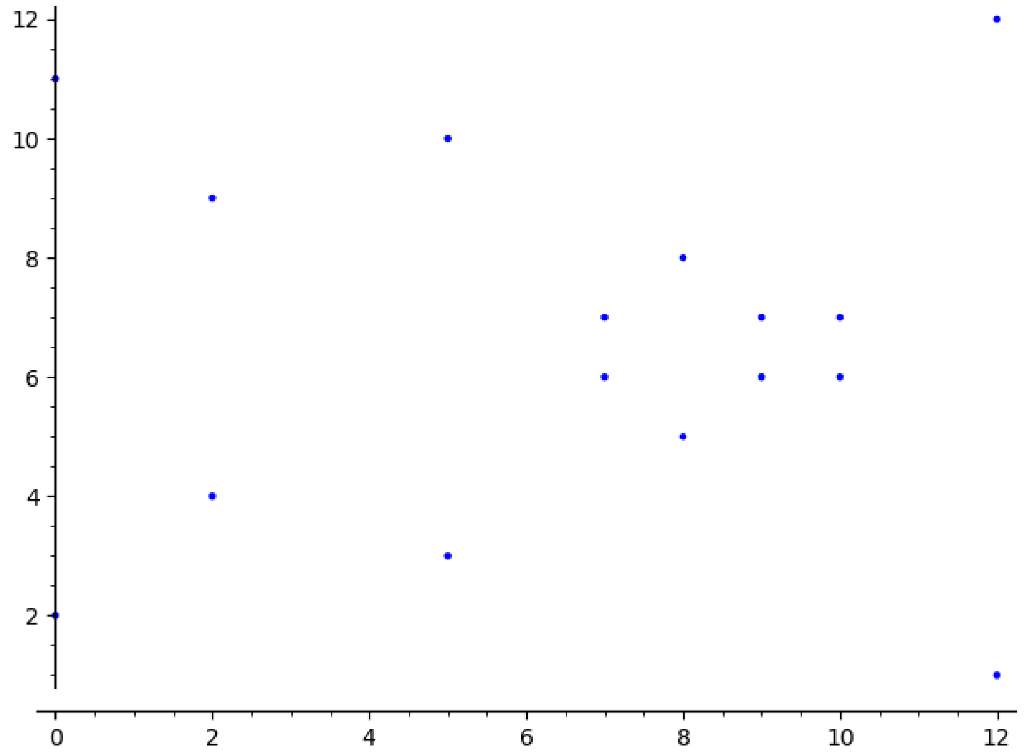
=>

order r = 17

p = 13

$r \mid p^k - 1$

$17 \mid 13^1 - 1$ <=> 17|13 NO



Pairings of elliptic curves (3)

embedding degree of a curve **k**

8 pairs + identity (INF) = 17

=>

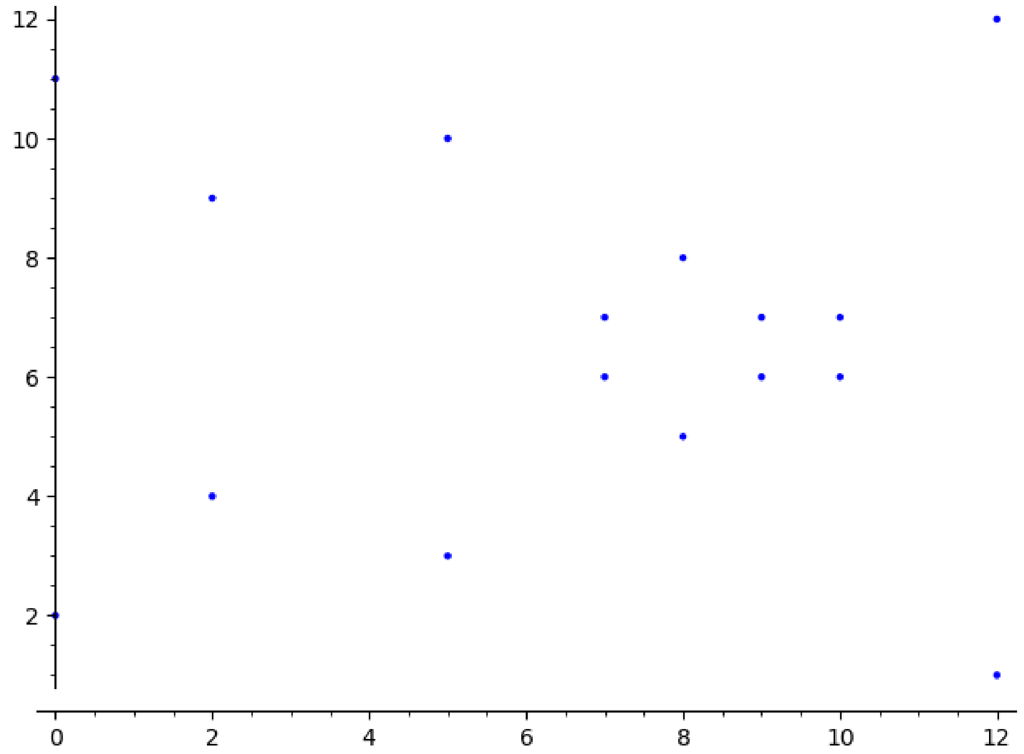
order r = 17

p = 13

$r \mid p^k - 1$

$17 \mid 13^1 - 1$ <=> 17|13 NO

$17 \mid 13^2 - 1$ <=> 17|168 NO



Pairings of elliptic curves (3)

embedding degree of a curve **k**

8 pairs + identity (INF) = 17

=>

order r = 17

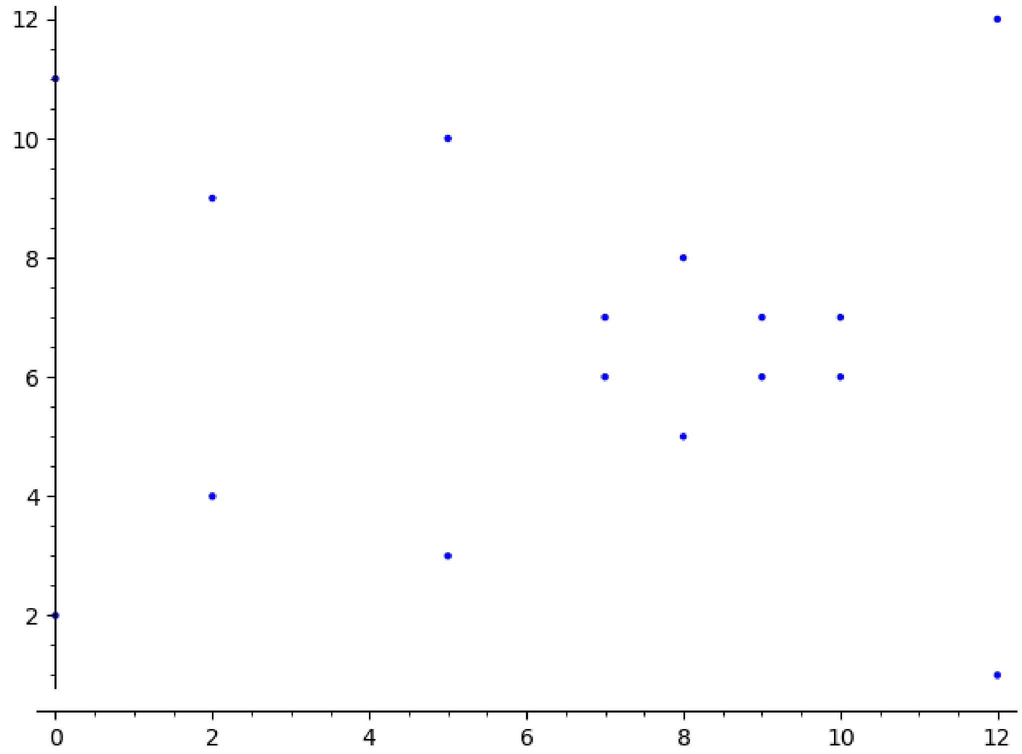
p = 13

$r \mid p^k - 1$

$17 \mid 13^1 - 1$ <=> 17|13 NO

$17 \mid 13^2 - 1$ <=> 17|168 NO

$17 \mid 13^3 - 1$ <=> 17|2196 NO



Pairings of elliptic curves (3)

embedding degree of a curve **k**

8 pairs + identity (INF) = 17

=>

order r = 17

p = 13

$r \mid p^k - 1$

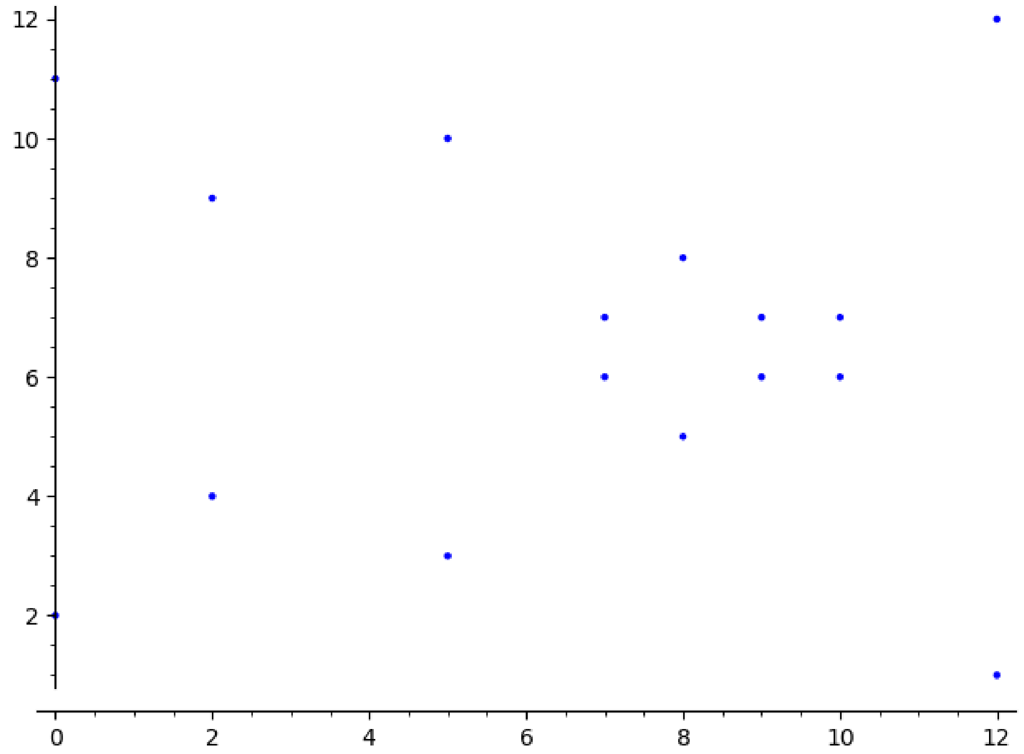
$17 \mid 13^1 - 1$ <=> 17|13 NO

$17 \mid 13^2 - 1$ <=> 17|168 NO

$17 \mid 13^3 - 1$ <=> 17|2196 NO

$17 \mid 13^4 - 1$ <=> 17|28560 YES

k=4



Pairings of elliptic curves (3)

```
sage: p=13
sage: F13=GF(p)
sage: a = F13(2)
sage: b = F13(4)
sage: E = EllipticCurve(F13,[a,b])
sage: r= E.order()
17
sage: k = 1
sage: while True:
....:     # Check if the order r divides (p^k - 1)
....:     if (p**k - 1) % r == 0:
....:         print(f"The embedding degree k is: {k}")
....:         break
....:     k += 1
....:
The embedding degree k is: 4
```

Pairings of elliptic curves (4)

```
sage: # Now again secp256k1
sage: p =115792089237316195423570985008687907853269984665640564039457584007908834671663
sage: Fp = GF(p)
sage: secp256k1 = EllipticCurve(Fp,[0,7])
sage: r= secp256k1.order()
sage: r
115792089237316195423570985008687907852837564279074904382605163141518161494337
sage: k = 1
sage: while k < 1000:
....:     if (p^k-1)%r == 0:
....:         break
....:     k=k+1
....:
sage: k
1000
```

```
sage: # in fact it very large
sage: # k =19298681539552699237261830834781317547292737984581739710086052358636024 9056
```

Pairings of elliptic curves (5)

secp256k1 is a curve that has large k

As extension field is built over p^k it means the extension field is here extremely big. =>

secp256k1 NEEDS k-many entries, each of them 256 bits

=> well not enough atoms in the observable universe out there

Pairings of elliptic curves (5)

secp256k1 is a curve that has large k

As extension field is built over p^k it means the extension field is here extremely big. =>

secp256k1 NEEDS k-many entries, each of them 256 bits

=> well not enough atoms in the observable universe out there

secp256k1 is not pairing-friendly

we want curves with low embedding degree:

BN128 -> 12

Jubjub

BLS

That’s it! More to come in the future

Get in touch 👋

⇧My notes on GitHub