Bitcoin:
Programming the Future of Money

Topics in Computer Science - ITCS 4010/5010, Spring 2025

Dr. Christian Kümmerle

Lecture 12

Elliptic Curve-Based Digital Signature Schemes (II)

Main Reference:
- “Programming Bitcoin: Learn How to Program Bitcoin from Scratch”, Jimmy Song, 1st Edition, O’Reilly, 2019, Chapters 3
- “Mastering Bitcoin”, Harding & Antonopoulos, Chapter 8 on “Digital Signatures”

UNIVERSITY OF NORTH CAROLINA
CHARLOTTE

Quiz Time

SOME QUIZ QUESTIONS

For the elliptic curve $S_{5,7} = \{(x, y) : y^2 = x^3 + 5x + 7\}$ over the real numbers, if

$$B = (0,y_2) \in S_{5,7},$$

what is y_2?

SOME QUIZ QUESTIONS

For the elliptic curve $S_{5,7} = \{(x, y) : y^2 = x^3 + 5x + 7\}$ over the real numbers, if

$$B = (0,y_2) \in S_{5,7},$$

what is y_2?

Answer: $y_2 = \sqrt{7}$ or $y_2 = -\sqrt{7}$

Elliptic Curve-Based Digital Signature Schemes

We create now secure identity and signature schemes based on the finite field arithmetic of points $P = (x, y) \in S_{a,b}$ where $x, y \in \mathbb{F}_p$ for suitable a, b and very large p.

We choose:

- ▷ Finite field $\mathbb{F}_p = \{0, 1, \dots, p-1\}$ with $p = 2^{256} - 2^{32} - 977$.
- ▷ $a = 0$, $b = 7$ (thus, $S_{a,b} = S_{0,7} = \{(x, y) \in \mathbb{F}_p \times \mathbb{F}_p : y^2 = x^3 + 7\}$)
- ▷ Generator point $G = (G_x, G_y)$, with $G_x, G_y \in \mathbb{F}_p$ specific, well-chosen numbers
- ▷ $n := \underbrace{|\{f \cdot G : f \in \mathbb{N}\}|}$

this quantity is called (group) order of the (Abelian) group $\{f \cdot G : f \in \mathbb{N}\} \subset S_{a,b}$.

these numbers can be looked up

Notation:
If S is a set, $|S|$ counts how many distinct elements are in set S.

ELLIPTIC-CURVE BASED DIGITAL SIGNATURE SCHEMES

- Since $p < 2^{256}$, but $p \simeq 2^{256}$ (same order of magnitude), it is convenient to represent $G_x, G_y \in \mathbb{F}_p$ as 256-bit integers.
- Note: $2^{256} > 10^{77}$

  Nr. of atoms in universe: $\approx 10^{80}$ (according to estimates)

  $\Rightarrow$ 2^{256} is HUGE!

DIGITAL SIGNATURE SCHEMES USED IN BITCOIN

- **Elliptic Curve Digital Signature Algorithm (ECDSA)**
  - Concept proposed by Neal Koblitz and Victor S. Miller in 1985
  - Standardized in 2000 by NIST
  - Used in Bitcoin since 2009, was freely available
  - Used by all address formats before Taproot upgrade

- **Schnorr Signatures:**
  - Proposed and **patented** by Claus-Peter Schnorr in 1990
  - Has certain advantages over ECDSA (will see later) and simpler
  - Patent expired in 2010, so not available at inception of Bitcoin
  - Implemented in address format introduced by 2021 Taproot upgrade

ELLIPTIC CURVE DIGITAL SIGNATURE ALGORITHM

ECDSAsign(e, k, m)

▷ $R = k \cdot G$ $[\in S_{0,7}]$

▷ Define r as x-coordinate of $R = (r, R_y)$

▷ Compute $z = hash(m)$

▷ Compute $s = (z + r \cdot e)/k$

[this is done in F_n]

where n is group order of generator group

return (r, s)

use SHA256(SHA256(·))

makes sure that z is 256-bit integer

Glossary:

▷ $e \in F_p$: private key

▷ $G \in S_{0,7}$: Generator point

▷ $P \in S_{0,7}$: Public key (satisfies $P = e \cdot G$)

▷ $k \in F_p$: random (private) nonce

▷ $r \in F_p$: public nonce (derived from private nonce)

▷ m message to be signed

▷ s : signature

ECDSAverify(P, m, r, s)

▷ Compute $z = hash(m)$

▷ Compute $u = z/s$ [in F_n]

▷ Compute $v = r/s$ [in F_n]

▷ testval $= u \cdot G + v \cdot P$

▷ If (x-coordinate of testval) $== r$
return True

Else
return False

$$s = (z + re) \cdot k^{-1} \qquad [\text{in } \mathbb{F}_n]$$

hash(m)

x-coordinate of some R

private key

256-bit random number

$z = z \bmod n$

$r = r \bmod n$

$e = e \bmod n$

$k = k \bmod n$

$n < p < 2^{256}$

ELLIPTIC CURVE DIGITAL SIGNATURE ALGORITHM

Intuition: The formulas of ECDSAsign and ECDSAverify are derived from following 3 equations:

$$u \cdot G + v \cdot P = R \quad \text{(I) (from ECDSA verify)}$$

$$k \cdot G = R \quad \text{(II) (from — " — sign)}$$

$$e\, G = P \quad \text{(III) (equation relating public \& private key)}$$

(II) in (I) $\Longrightarrow$

$$u \cdot G + v\, e \cdot G = k \cdot G$$

This equation is satisfied if scalars in front of generator point G are matching, i.e., if:

$$u + ve = k \quad \text{in finite field } \mathbb{F}_n$$

inserting (*) for k $\Longrightarrow$

$$\underbrace{u + ve = \frac{z}{s} + \frac{r}{s} \cdot e}$$

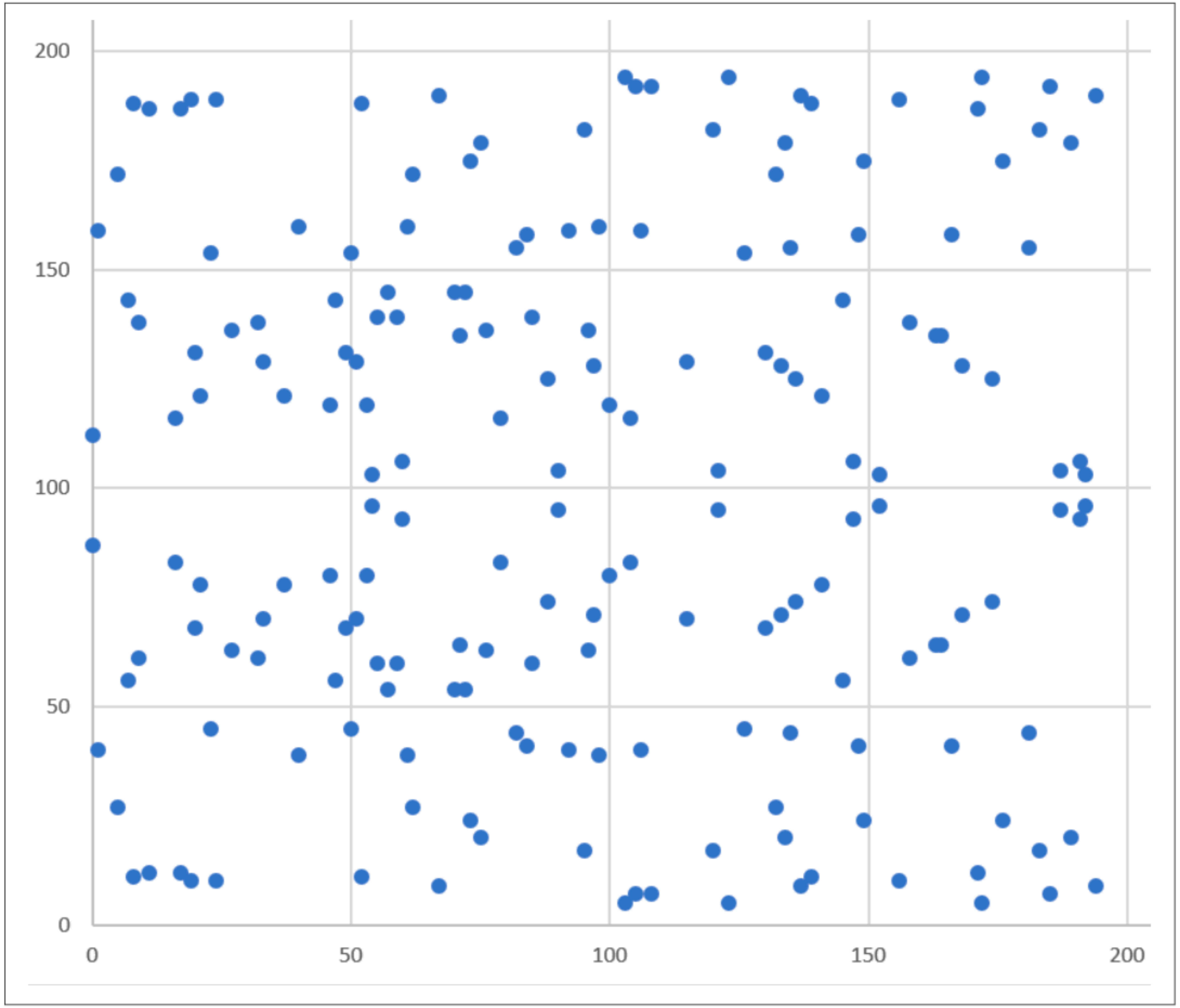
this equality holds if $u = \frac{z}{s}$ and $v = \frac{r}{s}$ in $\mathbb{F}_n$

from step 4 of ECDSAsign:

$$s = (z + re)/k$$

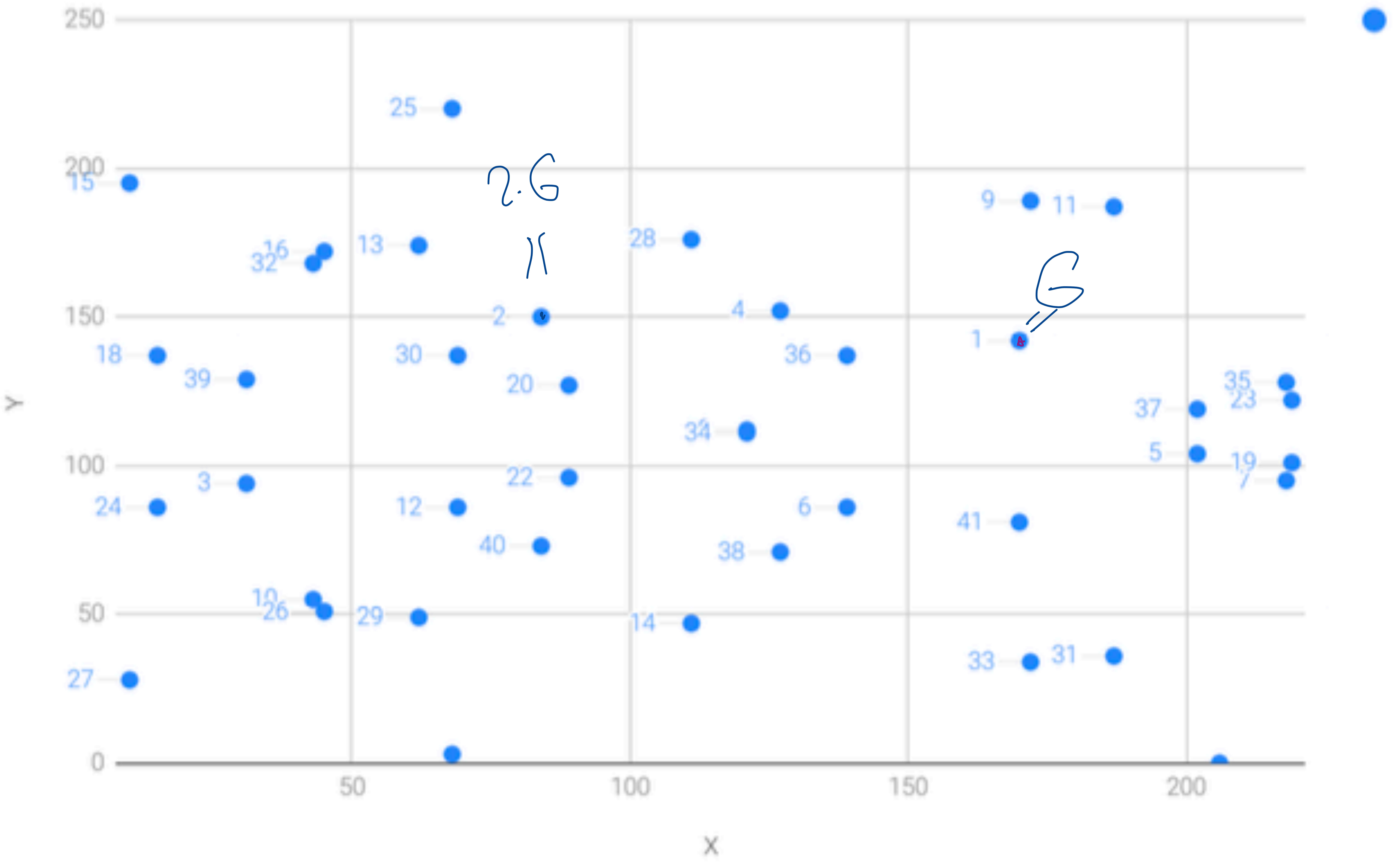
$$\Longleftrightarrow \underset{(*)}{k =} \frac{z + re}{s}$$

AN ELLIPTIC CURVE ON A FINITE FIELD



Example of elliptic curve on F_{191}

SCALAR MULTIPLICATION ON ELLIPTIC CURVES OVER FINITE FIELDS



EC equation:

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

Finite field:

$$F_{223}$$

Generator point:

$$G = (170,142)$$

CALCULATING WITH FINITE-FIELD VALUED POINTS ON ELLIPTIC CURVES

Elliptic Curve $S_{2,2} = \{(x,y) \in F_p \times F_p : y^2 = x^3 + 2x + 2\}$, $p = 17$.

$x^3 + ax + b$

Generator Point $G = (G_x, G_y)$ with $x = G_x = 5$

$\cancel{(4,4)}$

$$x^3 + 2x + 2 \mod 17 = (5^3 + 2\cdot 5 + 2) \mod 17$$
$$= (125 + 10 + 2) \mod 17$$
$$= 137 \mod 17 = 1$$

We need $y = G_y$ to satisfy: $y^2 = 1$

$\Rightarrow$ $y = G_y = 1$ satisfies this

$\Rightarrow$ $G = (5, 1)$

Q: What is the group order n of group $\{k \cdot G : k \in \mathbb{N}\}$ generated by G?

Idea: Compute $2\cdot G$, $3 \cdot G$, $4 \cdot G$, ... until $n \cdot G = \mathcal{O}$

k	$k \cdot 17$
1	17
2	34
3	51
4	68
5	85
6	102
7	119
8	136

CALCULATING WITH FINITE-FIELD VALUED POINTS ON ELLIPTIC CURVES

Compute $2G$. [Recall $G=(5,1)$]

Enter: Point addition of EC point $\underset{(x_1,y_1)}{A=G}$, $\underset{(x_2,y_2)}{B=G}$ (since $2G=G+G$)

Here: $x_1=x_2=5$, $y_1=y_2=1\neq 0$

$\hookrightarrow$ Case 3.

Revisit formulas

$a=2$

1) $s=\frac{3x_1^2+a}{2y_1}=(3x_1^2+a)\cdot(2\cdot y_1)^{-1}$

2) $x_3=s^2-2x_1$

3) $y_3=s(x_1-x_3)-y_1$

1) $(3\cdot 5^2+2) \bmod 17 =$
$=(75+2) \bmod 17 = 77 \bmod 17$
$=9$

$(2y_1)^{-1}=(2\cdot 1)^{-1}=2^{-1} \bmod 17$
$=1\cdot 2^{-1} \bmod 17$

Fermat

$=2^{p-1}\cdot 2^{-1} \bmod 17$

$p=17$

$=2^{15} \bmod 17$

$2\cdot x=1 \bmod 17$

$2\cdot 9=18 \bmod 17$
$=1$

$=9$

CALCULATING WITH FINITE-FIELD VALUED POINTS ON ELLIPTIC CURVES

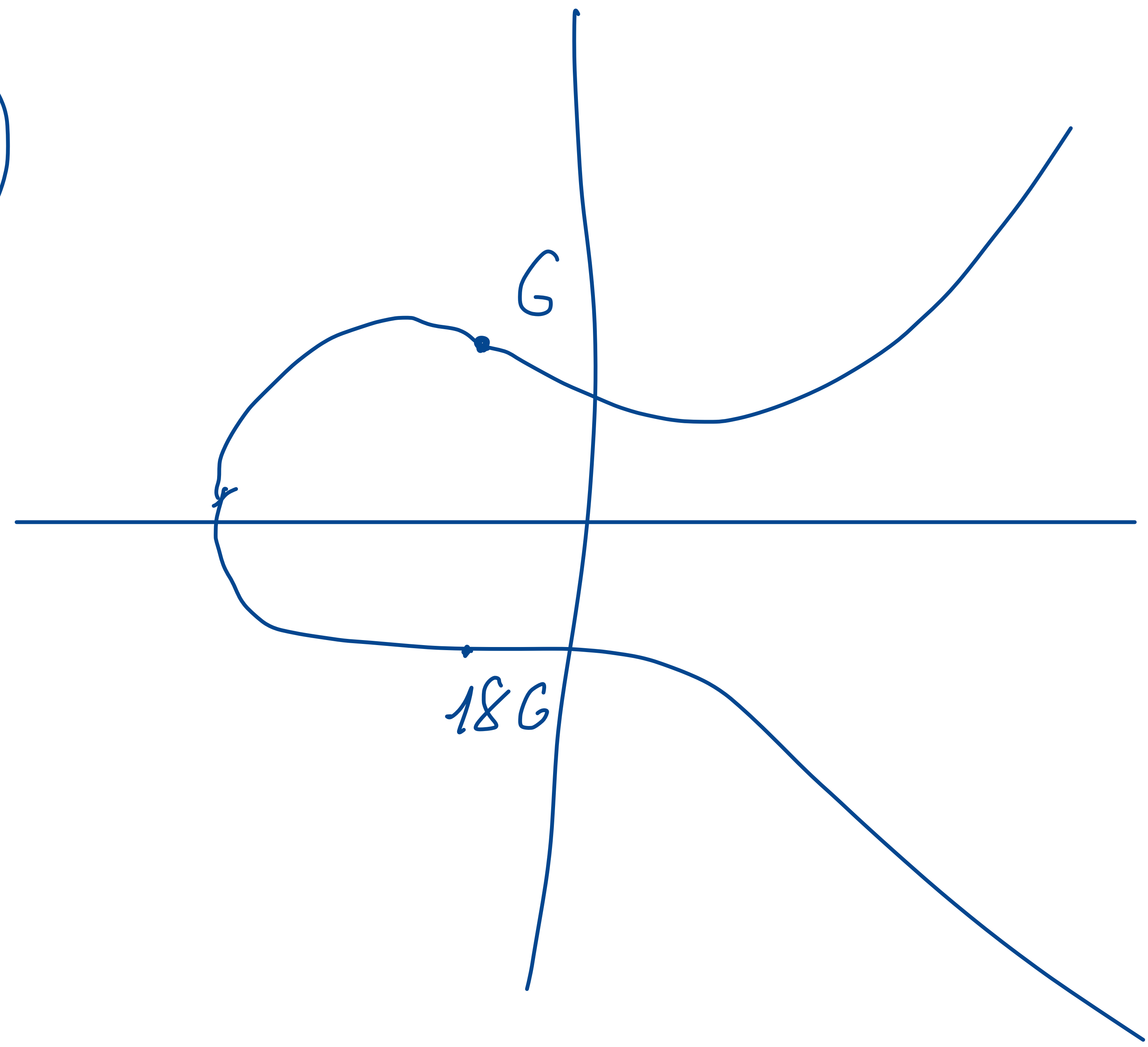
..... repeat for $3G$, $4G$, ...

$19 \cdot G = \mathcal{O}$ ← point at infinity

$p = 17$

$18G = (5, \underset{\substack{\| \\ -1 \bmod p}}{p-1})$

$(G = (5, 1))$



$\longrightarrow \quad n = 19$

CALCULATING WITH FINITE-FIELD VALUED POINTS ON ELLIPTIC CURVES

$$\Rightarrow \quad s = (3 \cdot 5^2 + 2) \cdot (2y_1)^{-1} \bmod 17 = 9 \cdot 9 \bmod 17 =$$

$$= 81 \bmod 17$$

$$= (81 - 68) \bmod 17 = \underline{\underline{13}}$$

2) $x_3 = s^2 - 2x_1 \bmod 17$ (x - coordinate of $G+G$).

$$= 13^2 - 2 \cdot 5 \bmod 17$$

$$= 169 - 10 \bmod 17 =$$

$$= 169 \bmod 17 - 10 \bmod 17$$

$$= (-1) \bmod 17 - 10 \bmod 17$$

$$= -11 \bmod 17 = \underline{\underline{6}}$$

3) $y_3 = s(x_1 - x_3) - y_1$

$$= 13 \cdot (5 - 6) - 1 = -13 - 1 \bmod 17 = -14 \bmod 17 = \underline{\underline{3}}$$

$$\Rightarrow \quad 2 \cdot G = (6, 3$$

SIMPLIFIED SCHNORR IDENTITY PROTOCOL

Schnorr Signature Scheme

- Uses also secp256k1 with same p, n, and generator point G as ECDSA.
- Unlike ECDSA, security of signature/verification algorithms can be proven under two assumptions

(A1) • It is too costly to find $s \in \mathbb{F}_n$ such that $s \cdot G = T$ for any given $T \in S_{0,7}$

(A2) • Cryptographic hash function hash(·) satisfies random oracle model:

Assumes that for each unseen input x, hash(x) is (uniformly) random among range R of hash(·)

- Signatures s are linear in input parameters
  - ⟶ makes batch verification of signatures possible
- Slightly shorter serialized representation

SIMPLIFIED SCHNORR IDENTITY PROTOCOL

SchnorrIdentity(e , k)

(interactive version)

- $\boxed{R = k \cdot G}$
- Send R to counterparty.
- Bob (counterparty) choose challenge target z,
  send back to Alice
- $S = k + z \cdot e$
- Send s to Bob/counterparty.

SchnorrVerify(z, s, P, R) :

- Compute $s \cdot G$
- Compute $z \cdot P$
- Compute $testval = R + z \cdot P$
- If $sG == testval$
  return True
  Else return False

$e \cdot G = P$

e: private key

k: private random nonce

P: public key

R: public nonce

"Private side":

$S = k + z \cdot e$

"Public side":

$\underline{sG} = k \cdot G + z \cdot e \cdot G$

$= R + \underline{z \cdot P}$

SIMPLIFIED SCHNORR IDENTITY PROTOCOL

SchnorrIdentity(e , k)

NON-
(interactive version)

$e \cdot G = P$

e: private key
k: private random nonce
P: public key
R: public nonce

- $\boxed{R = k \cdot G}$
- Compute $z = hash(R)$
-
- $s = k + z \cdot e$
- Return (R, s)

Idea: Use hash function to avoid inter-activity

SchnorrVerify(s, P, R) :

- Compute $s \cdot G$
- $z = hash(R)$
- Compute $z \cdot P$
- Compute testval $= R + z \cdot P$
- If $sG == testval$
  return True
  Else return False

"Private side":

$s = k + z \cdot e$

"Public side":

$\underline{sG} = k \cdot G + z \cdot e \cdot G$

$= R + \underline{z \cdot P}$

SCHNORR SIGNATURES

SchnorrSign (e, k, m) (non-interactive) $e \cdot G = P$

- $\boxed{R = k \cdot G}$
- Compute $z = hash(R \| e \| m)$
- $s = k + z \cdot e$
- Return (R, s)

concatenate bit strings

e: private key
k: private random nonce
P: public key
R: public nonc
m: message

SchnorrVerify (s, P, R, m):

- Compute $s \cdot G$
- $z = hash(R \| P \| m)$
- Compute $z \cdot P$
- Compute $testval = R + z \cdot P$
- If $sG == testval$ return True
  Else return False

"Private side":

$s = k + z \cdot e$

"Public side":

$\underline{sG} = k \cdot G + z \cdot e \cdot G$

$= R + \underline{z \cdot P}$

$=$