POLITECNICO
MILANO 1863

An Advanced Signature Scheme:

Schnorr Algorithm and its Benefits to the Bitcoin Ecosystem

Author:
Giona Soldati

Supervisors:
Daniele Marazzina
Ferdinando M. Ametrano

School of Industrial and Information Engineering
Master of Science in Mathematical Engineering

20$^{\text{th}}$ December 2018

Introduction

The Elliptic Curve Digital Signature Algorithm (ECDSA) is used in the Bitcoin protocol as signature scheme, but it has some problems:

Introduction

The Elliptic Curve Digital Signature Algorithm (ECDSA) is used in the Bitcoin protocol as signature scheme, but it has some problems:

1. Efficiency (DER encoding, no batch validation, modular inversion);

Introduction

The Elliptic Curve Digital Signature Algorithm (ECDSA) is used in the Bitcoin protocol as signature scheme, but it has some problems:

1. Efficiency (DER encoding, no batch validation, modular inversion);
2. Poor implementation of higher level constructions (low privacy and fungibility, scales badly);

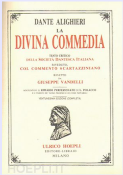
Introduction

The Elliptic Curve Digital Signature Algorithm (ECDSA) is used in the Bitcoin protocol as signature scheme, but it has some problems:

1. Efficiency (DER encoding, no batch validation, modular inversion);
2. Poor implementation of higher level constructions (low privacy and fungibility, scales badly);
3. Not provably secure (malleable).

Outline

Hash functions ($\simeq$ Random functions)

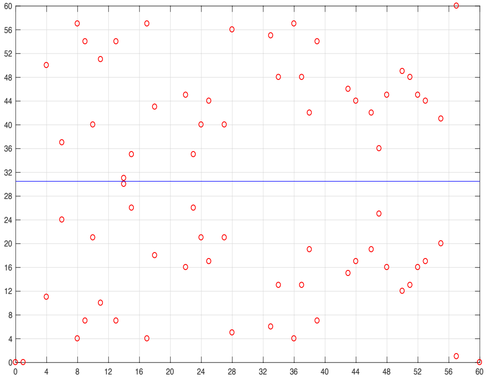
Input		Output	
"Politecnico di Milano"	$\xrightarrow{\text{SHA-256}}$ / $\not\leftarrow$	52c61456bd3d60c08599a8c61e113e7b c39118b75d06b2643c55957ece91269b	
	$\xrightarrow{\text{SHA-256}}$ / $\not\leftarrow$	1f12f8384d5941e8e273926635be646a f786405a8662e0ba6dd1fd0526ec0528	
	$\xrightarrow{\text{SHA-256}}$ / $\not\leftarrow$	721a14d89cb463b51cf20ecc707aa290 5b7c8f32d9114dc19fe353e611494bf1	

Elliptic curve cryptography

An elliptic curve over a finite field is defined by:

$$E(\mathbb{F}_p):\; y^2 = x^3 + ax + b \pmod p.$$

It is possible to define:



The curve $y^2 = x^3 - x$ over $\mathbb{F}_{61}$.

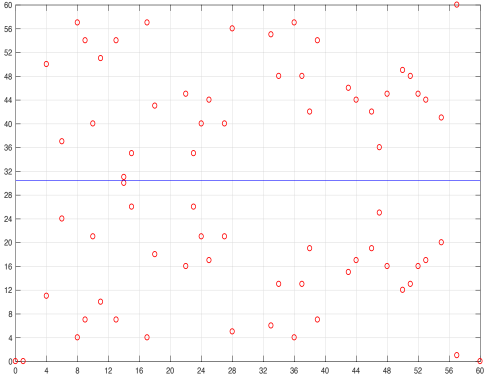
Elliptic curve cryptography

An elliptic curve over a finite field is defined by:

$$E(\mathbb{F}_p): \; y^2 = x^3 + ax + b \pmod{p}.$$

It is possible to define:

- Addition: $Q_3 := Q_1 + Q_2, \; \forall Q_1, Q_2 \in E(\mathbb{F}_p)$;



The curve $y^2 = x^3 - x$ over $\mathbb{F}_{61}$.

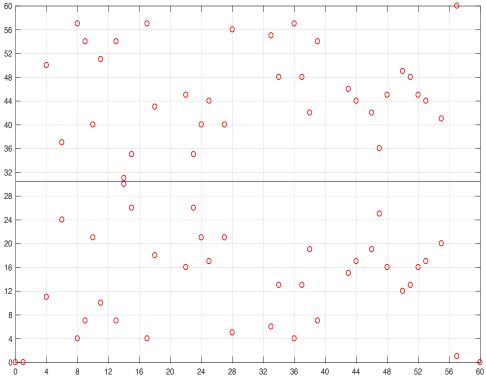
Elliptic curve cryptography

An elliptic curve over a finite field is defined by:

$$E(\mathbb{F}_p):\ y^2 = x^3 + ax + b \,(\mathrm{mod}\ p).$$

It is possible to define:

- Addition: $Q_3 := Q_1 + Q_2,\ \forall Q_1, Q_2 \in E(\mathbb{F}_p)$;
- Scalar multiplication: $nG = G + \ldots + G$, $\forall G \in E(\mathbb{F}_p), \forall n \in \mathbb{N}$.



The curve $y^2 = x^3 - x$ over $\mathbb{F}_{61}$.

Elliptic curve cryptography

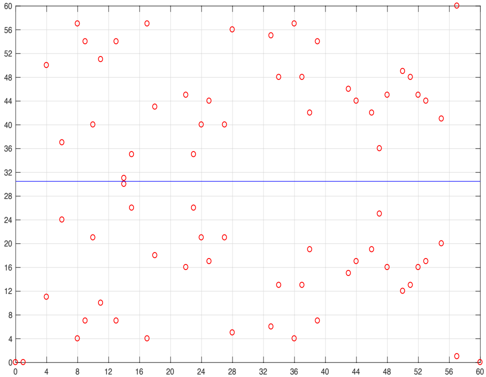
An elliptic curve over a finite field is defined by:

$$E(\mathbb{F}_p):\ y^2 = x^3 + ax + b \pmod{p}.$$

It is possible to define:

- Addition: $Q_3 := Q_1 + Q_2,\ \forall Q_1, Q_2 \in E(\mathbb{F}_p)$;
- Scalar multiplication: $nG = G + \ldots + G$, $\forall G \in E(\mathbb{F}_p), \forall n \in \mathbb{N}$.

Multiplication's computational asymmetry is the core of ECC.



The curve $y^2 = x^3 - x$ over $\mathbb{F}_{61}$.

Discrete logarithm problem

Fixed $G \in E(\mathbb{F}_p)$, we can define $Q = qG \quad \forall q \in [1, ..., n-1]$:

Discrete logarithm problem

Fixed $G \in E(\mathbb{F}_p)$, we can define $Q = qG \quad \forall q \in [1, ..., n-1]$:

- The direct operation $q \mapsto Q$ is efficient;

Discrete logarithm problem

Fixed $G \in E(\mathbb{F}_p)$, we can define $Q = qG \quad \forall q \in [1, ..., n-1]$:

- The direct operation $q \mapsto Q$ is efficient;
- The inverse operation $Q \mapsto q$ is computationally infeasible for certain groups.

Discrete logarithm problem

Fixed $G \in E(\mathbb{F}_p)$, we can define $Q = qG \quad \forall q \in [1, ..., n-1]$:

- The direct operation $q \mapsto Q$ is efficient;
- The inverse operation $Q \mapsto q$ is computationally infeasible for certain groups.

Asymmetric cryptography: $\{q, Q\}$ is a key pair whose elements have complementary roles.

Discrete logarithm problem

Fixed $G \in E(\mathbb{F}_p)$, we can define $Q = qG \ \ \forall q \in [1, ..., n-1]$:

- The direct operation $q \mapsto Q$ is efficient;
- The inverse operation $Q \mapsto q$ is computationally infeasible for certain groups.

Asymmetric cryptography: $\{q, Q\}$ is a key pair whose elements have complementary roles.

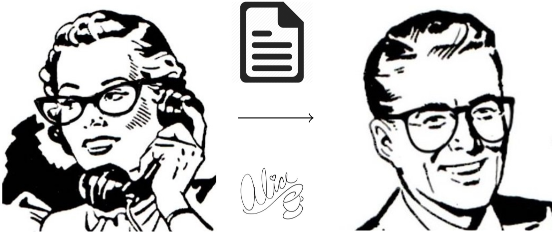
Double and add algorithm: $q = 41$

$$41 = 1 + 8 + 32 \implies 41G = G + 8G + 32G.$$

5 point doubling and 2 additions vs. 40 additions.

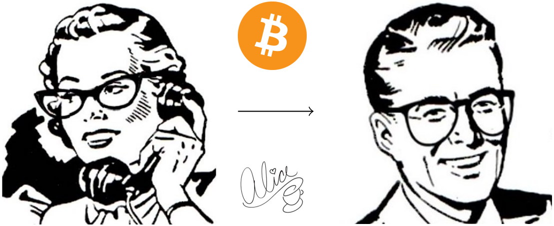
Outline

Digital signature



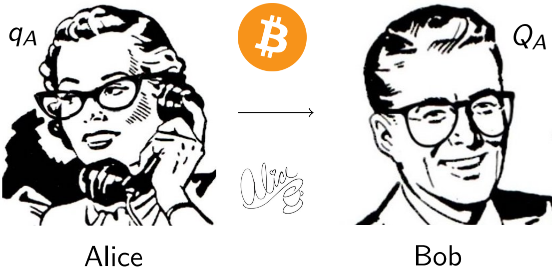
Alice Bob

Digital signature



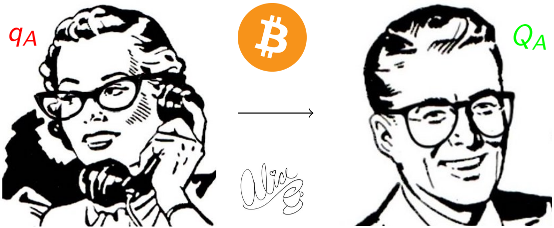
Alice Bob

Digital signature



- Authentication: the recipient is confident that the message comes from the alleged sender;

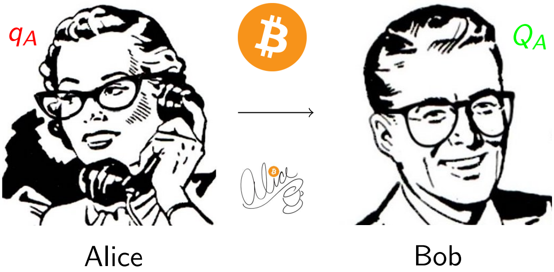
Digital signature



Alice Bob

- Authentication: the recipient is confident that the message comes from the alleged sender;
- Non repudiation: the sender cannot deny having sent the message;

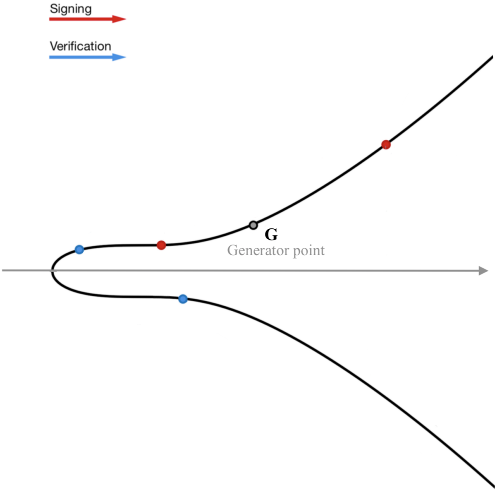
Digital signature



- Authentication: the recipient is confident that the message comes from the alleged sender;
- Non repudiation: the sender cannot deny having sent the message;
- Integrity: ensures that the message has not been altered during transmission.

Elliptic curve digital signature algorithm

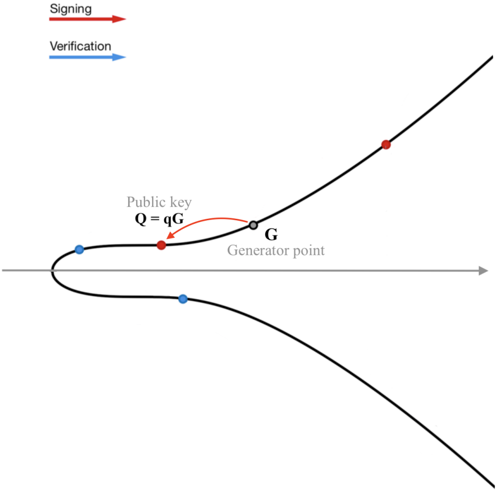
ECDSA_SIG(m, q):



Adapted from:
`https://medium.com/cryptoadvance/how-schnorr-signatures-may-improve-bitcoin-91655bcb4744`

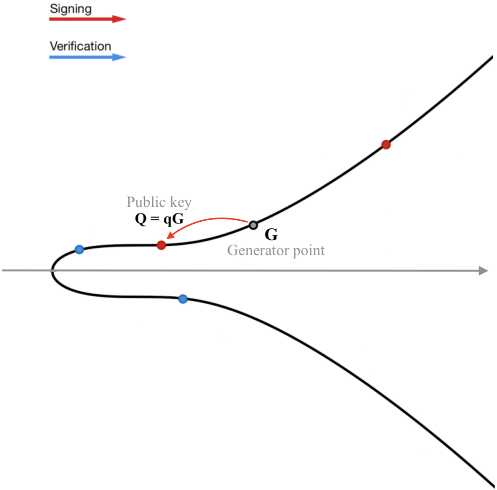
Elliptic curve digital signature algorithm

ECDSA_SIG(m, q):



Adapted from:
`https://medium.com/cryptoadvance/how-schnorr-signatures-may-improve-bitcoin-91655bcb4744`

Elliptic curve digital signature algorithm

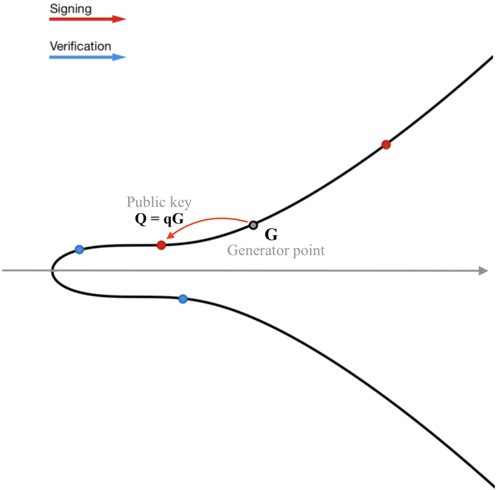


ECDSA_SIG(m, q):

1. $z \leftarrow \text{hash}(m)$;

Adapted from:
`https://medium.com/cryptoadvance/how-schnorr-signatures-may-improve-bitcoin-91655bcb4744`

Elliptic curve digital signature algorithm



ECDSA_SIG(m, q):

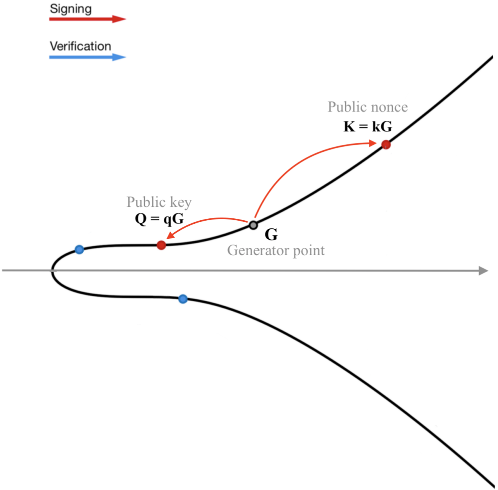
1. $z \leftarrow \text{hash}(m)$;
2. $k \xleftarrow{\$} \{1, ..., n-1\}$;

Adapted from:
`https://medium.com/cryptoadvance/how-schnorr-signatures-may-improve-bitcoin-91655bcb4744`

Elliptic curve digital signature algorithm

ECDSA_SIG(m, q):

1. $z \leftarrow \text{hash}(m)$;
2. $k \xleftarrow{\$} \{1, \ldots, n-1\}$;
3. $K \leftarrow kG$;

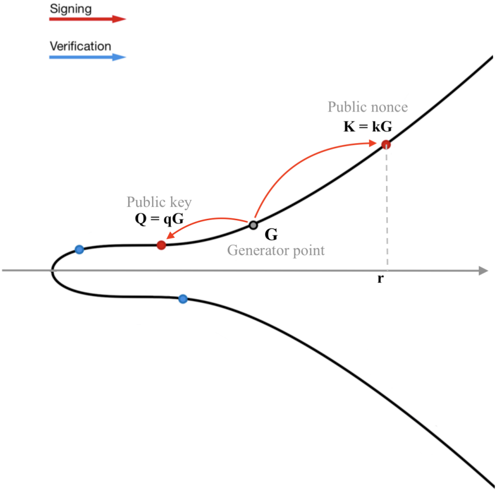


Adapted from:
`https://medium.com/cryptoadvance/how-schnorr-signatures-may-improve-bitcoin-91655bcb4744`

Elliptic curve digital signature algorithm

ECDSA_SIG(m, q):

1. $z \leftarrow \text{hash}(m)$;
2. $k \xleftarrow{\$} \{1, \ldots, n-1\}$;
3. $K \leftarrow kG$;
4. $r \leftarrow x_K \pmod{n}$;

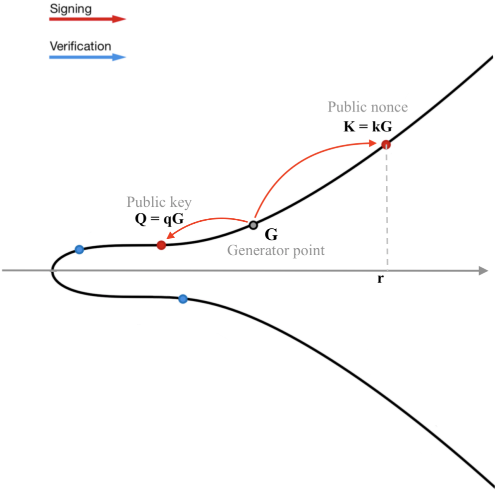


Adapted from:
`https://medium.com/cryptoadvance/how-schnorr-signatures-may-improve-bitcoin-91655bcb4744`

Elliptic curve digital signature algorithm

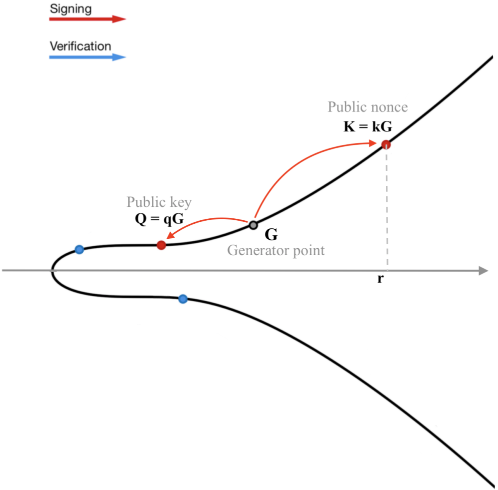
ECDSA_SIG(m, q):

1. $z \leftarrow \text{hash}(m)$;
2. $k \xleftarrow{\$} \{1, ..., n-1\}$;
3. $K \leftarrow kG$;
4. $r \leftarrow x_K \pmod{n}$;
5. $s \leftarrow k^{-1}(z + rq) \pmod{n}$;



Adapted from:
`https://medium.com/cryptoadvance/how-schnorr-signatures-may-improve-bitcoin-91655bcb4744`

Elliptic curve digital signature algorithm



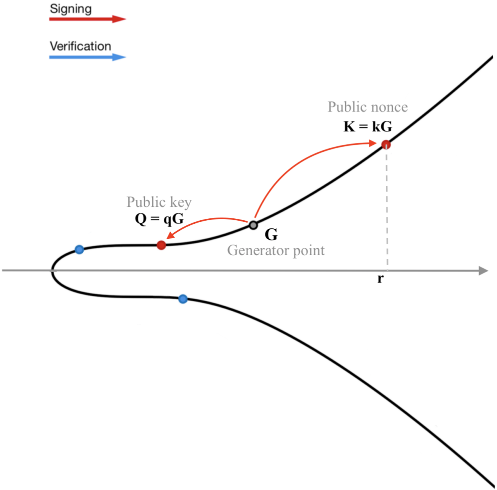
ECDSA_SIG(m, q):

1. $z \leftarrow \text{hash}(m)$;
2. $k \xleftarrow{\$} \{1, \ldots, n-1\}$;
3. $K \leftarrow kG$;
4. $r \leftarrow x_K \pmod{n}$;
5. $s \leftarrow k^{-1}(z + rq) \pmod{n}$;
6. **return** (r, s).

Adapted from:
`https://medium.com/cryptoadvance/how-schnorr-signatures-may-improve-bitcoin-91655bcb4744`

Elliptic curve digital signature algorithm

ECDSA_VER$((r, s), m, Q)$:

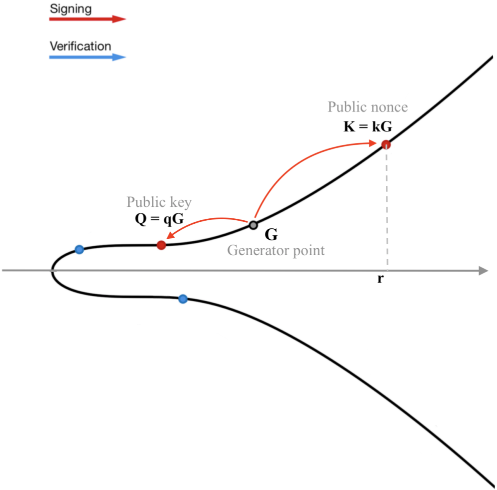


Adapted from:
`https://medium.com/cryptoadvance/how-schnorr-signatures-may-improve-bitcoin-91655bcb4744`

Elliptic curve digital signature algorithm

ECDSA_VER$((r, s), m, Q)$:

1. **If** $r \notin \{1, \ldots, n-1\}$ **or** $s \notin \{1, \ldots, n-1\}$:
   return False;

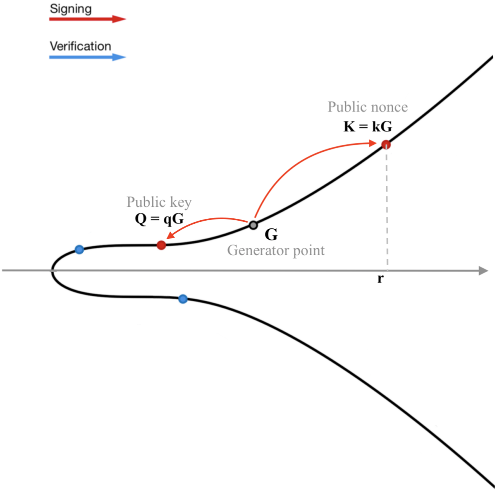


Adapted from:
`https://medium.com/cryptoadvance/how-schnorr-signatures-may-improve-bitcoin-91655bcb4744`

Elliptic curve digital signature algorithm

ECDSA_VER((r, s), m, Q):

1. **If** $r \notin \{1, \ldots, n-1\}$ **or** $s \notin \{1, \ldots, n-1\}$:
   return False;
2. $z \leftarrow \text{hash}(m)$;

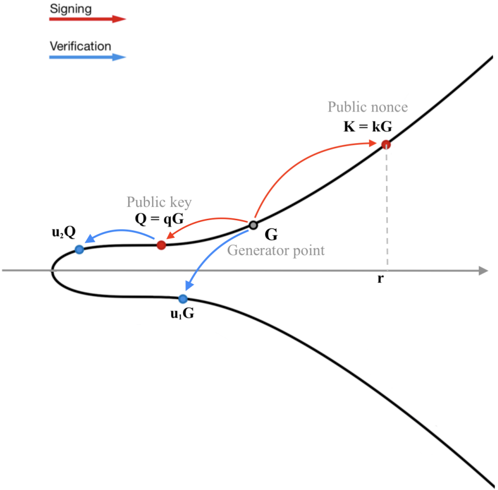


Adapted from:
`https://medium.com/cryptoadvance/how-schnorr-signatures-may-improve-bitcoin-91655bcb4744`

Elliptic curve digital signature algorithm

ECDSA_VER$((r, s), m, Q)$:

1. **If** $r \notin \{1, \ldots, n-1\}$ **or** $s \notin \{1, \ldots, n-1\}$:
   return False;
2. $z \leftarrow \text{hash}(m)$;
3. $u_1 \leftarrow zs^{-1} \pmod{n}$, $u_2 \leftarrow rs^{-1} \pmod{n}$;

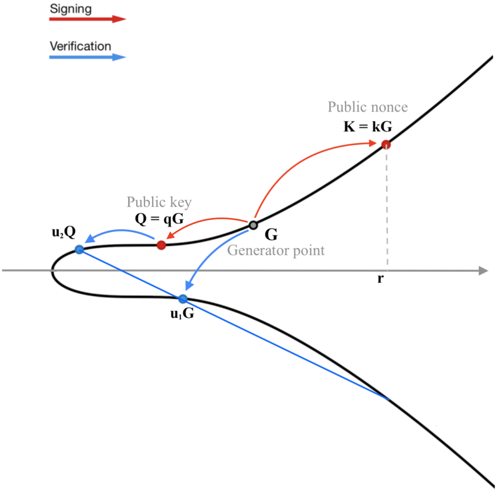


Adapted from:
`https://medium.com/cryptoadvance/how-schnorr-signatures-may-improve-bitcoin-91655bcb4744`

Elliptic curve digital signature algorithm

ECDSA_VER$((r, s), m, Q)$:

1. **If** $r \notin \{1, \ldots, n-1\}$ **or** $s \notin \{1, \ldots, n-1\}$:
   return False;
2. $z \leftarrow \text{hash}(m)$;
3. $u_1 \leftarrow zs^{-1} \pmod{n}$,
   $u_2 \leftarrow rs^{-1} \pmod{n}$;
4. $K \leftarrow u_1 G + u_2 Q$;

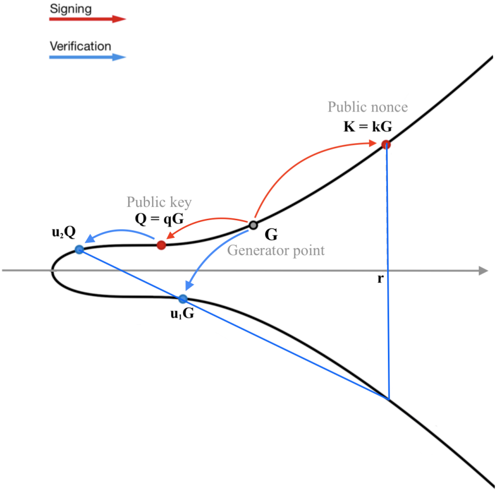


Adapted from:
https://medium.com/cryptoadvance/how-schnorr-signatures-may-improve-bitcoin-91655bcb4744

Elliptic curve digital signature algorithm

ECDSA_VER$((r, s), m, Q)$:

1. **If** $r \notin \{1, ..., n-1\}$ **or** $s \notin \{1, ..., n-1\}$:
   return False;
2. $z \leftarrow \text{hash}(m)$;
3. $u_1 \leftarrow zs^{-1} \pmod{n}$, $u_2 \leftarrow rs^{-1} \pmod{n}$;
4. $K \leftarrow u_1 G + u_2 Q$;

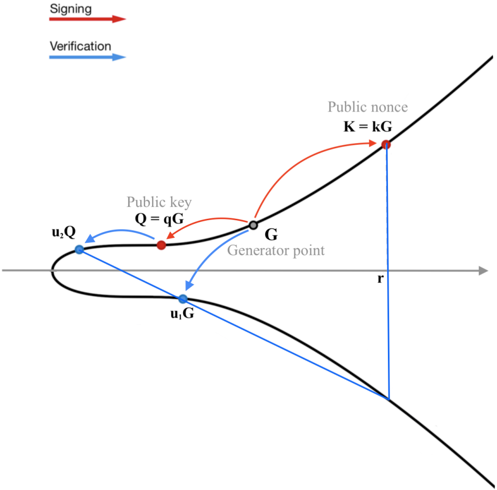


Adapted from:
`https://medium.com/cryptoadvance/how-schnorr-signatures-may-improve-bitcoin-91655bcb4744`

Elliptic curve digital signature algorithm

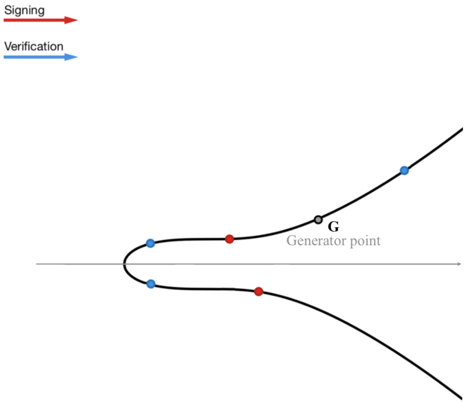
ECDSA_VER$((r, s), m, Q)$:

1. **If** $r \notin \{1, \ldots, n-1\}$ **or** $s \notin \{1, \ldots, n-1\}$:
   return False;
2. $z \leftarrow \mathrm{hash}(m)$;
3. $u_1 \leftarrow zs^{-1} \pmod{n}$,
   $u_2 \leftarrow rs^{-1} \pmod{n}$;
4. $K \leftarrow u_1 G + u_2 Q$;
5. **return** $r = x_K$ **(mod** n**)**.



Adapted from:
`https://medium.com/cryptoadvance/how-schnorr-signatures-may-improve-bitcoin-91655bcb4744`

Elliptic curve Schnorr signature algorithm

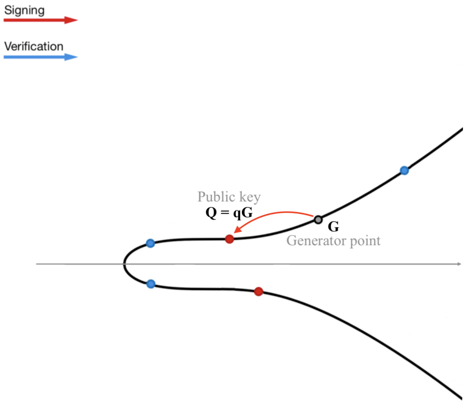


ECSSA_SIG(m, q):

Adapted from:
`https://medium.com/cryptoadvance/how-schnorr-signatures-may-improve-bitcoin-91655bcb4744`

Elliptic curve Schnorr signature algorithm

ECSSA_SIG(m, q):

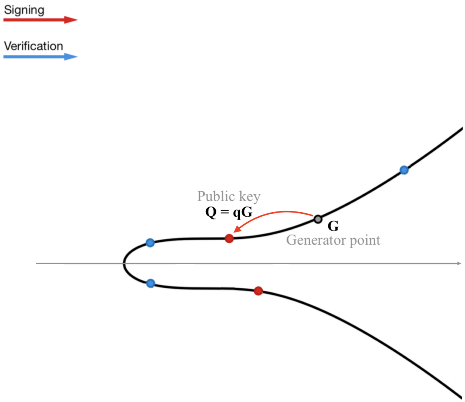


Adapted from:
`https://medium.com/cryptoadvance/how-schnorr-signatures-may-improve-bitcoin-91655bcb4744`

Elliptic curve Schnorr signature algorithm

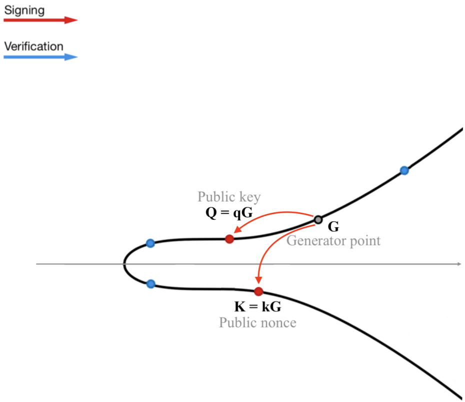
ECSSA_SIG(m, q):

1. $k \xleftarrow{\$} \{1, ..., n-1\}$;



Adapted from:
`https://medium.com/cryptoadvance/how-schnorr-signatures-may-improve-bitcoin-91655bcb4744`

Elliptic curve Schnorr signature algorithm



ECSSA_SIG(m, q):

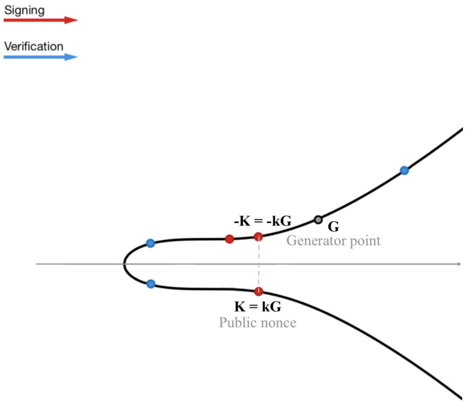
1. $k \xleftarrow{\$} \{1, ..., n-1\}$;
2. $K \leftarrow kG$;

Adapted from:
https://medium.com/cryptoadvance/how-schnorr-signatures-may-improve-bitcoin-91655bcb4744

Elliptic curve Schnorr signature algorithm

ECSSA_SIG(m, q):

1. $k \xleftarrow{\$} \{1, ..., n-1\}$;
2. $K \leftarrow kG$;
3. **If** jacobi$(y_K) \neq 1$:
   $k \leftarrow n - k$;

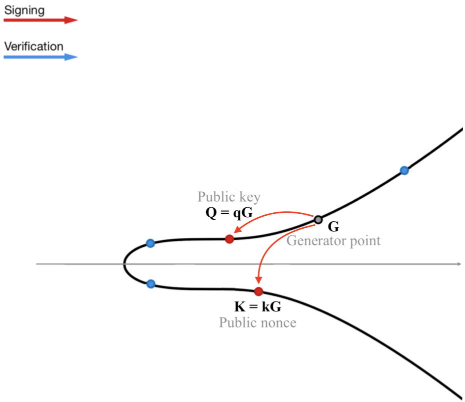


Adapted from:
https://medium.com/cryptoadvance/how-schnorr-signatures-may-improve-bitcoin-91655bcb4744

Elliptic curve Schnorr signature algorithm

ECSSA_SIG(m, q):

1. $k \xleftarrow{\$} \{1, \ldots, n-1\}$;
2. $K \leftarrow kG$;
3. **If** $\mathrm{jacobi}(y_K) \neq 1$:
   $k \leftarrow n - k$;
4. $e \leftarrow \mathrm{hash}(x_K || qG || m) \pmod{n}$;

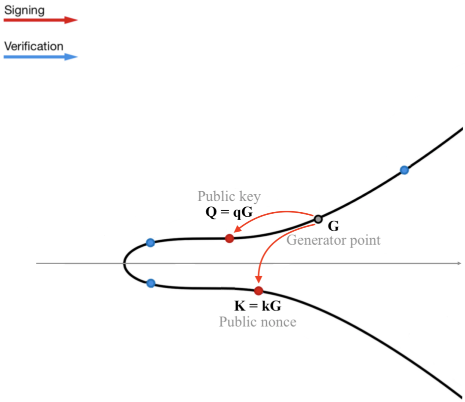


Adapted from:
https://medium.com/cryptoadvance/how-schnorr-signatures-may-improve-bitcoin-91655bcb4744

Elliptic curve Schnorr signature algorithm

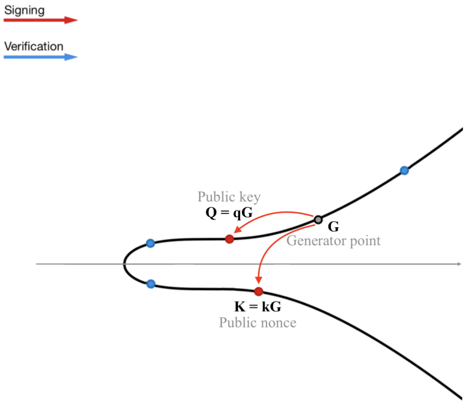
ECSSA_SIG(m, q):

1. $k \xleftarrow{\$} \{1, ..., n-1\}$;
2. $K \leftarrow kG$;
3. **If** $\text{jacobi}(y_K) \neq 1$:
   $k \leftarrow n - k$;
4. $e \leftarrow \text{hash}(x_K || qG || m) \pmod{n}$;
5. $s \leftarrow k + eq \pmod{n}$;



Adapted from:
`https://medium.com/cryptoadvance/how-schnorr-signatures-may-improve-bitcoin-91655bcb4744`

Elliptic curve Schnorr signature algorithm



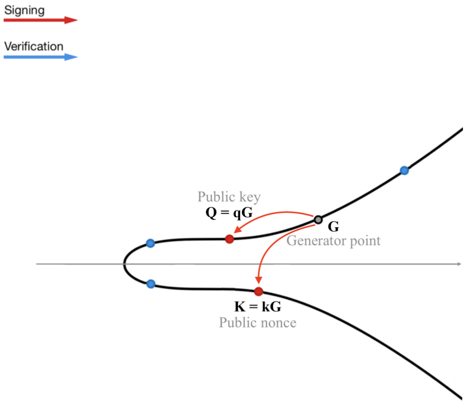
ECSSA_SIG(m, q):

1. $k \xleftarrow{\$} \{1, ..., n-1\}$;
2. $K \leftarrow kG$;
3. **If** jacobi$(y_K) \neq 1$:
   $k \leftarrow n - k$;
4. $e \leftarrow \text{hash}(x_K || qG || m) \pmod n$;
5. $s \leftarrow k + eq \pmod n$;
6. **return** (x_K, s).

Adapted from:
`https://medium.com/cryptoadvance/how-schnorr-signatures-may-improve-bitcoin-91655bcb4744`

Elliptic curve Schnorr signature algorithm

ECSSA_VER($(r, s), m, Q$):

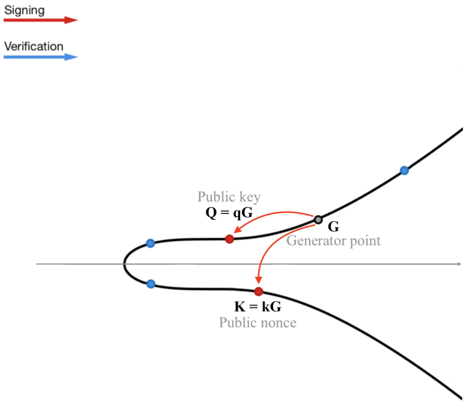


Adapted from:
`https://medium.com/cryptoadvance/how-schnorr-signatures-may-improve-bitcoin-91655bcb4744`

Elliptic curve Schnorr signature algorithm

ECSSA_VER($(r, s), m, Q$):

1. **If** $r \notin \{1, \ldots, p-1\}$ or $s \notin \{1, \ldots, n-1\}$:
   return False;

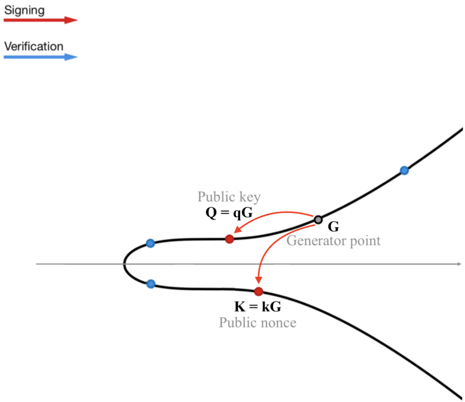


Adapted from:
`https://medium.com/cryptoadvance/how-schnorr-signatures-may-improve-bitcoin-91655bcb4744`

Elliptic curve Schnorr signature algorithm

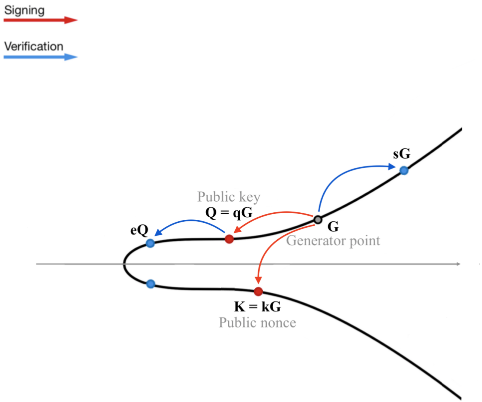
ECSSA_VER$((r, s), m, Q)$:

1. **If** $r \notin \{1, \ldots, p-1\}$ or $s \notin \{1, \ldots, n-1\}$:
   return False;
2. $e \leftarrow \mathrm{hash}(r||Q||m) \pmod{n}$;



Adapted from:
`https://medium.com/cryptoadvance/how-schnorr-signatures-may-improve-bitcoin-91655bcb4744`

Elliptic curve Schnorr signature algorithm



ECSSA_VER$((r, s), m, Q)$:

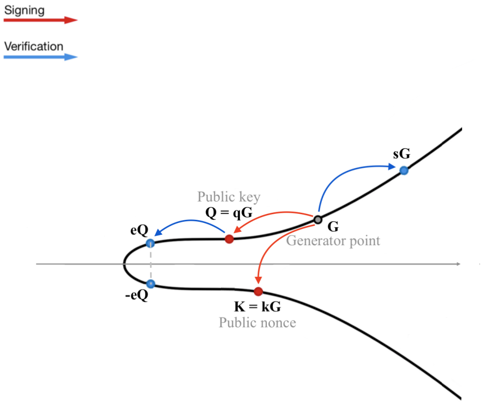
1. **If** $r \notin \{1, ..., p-1\}$ or $s \notin \{1, ..., n-1\}$:
   return False;
2. $e \leftarrow \text{hash}(r||Q||m) \pmod{n}$;
3. $K \leftarrow sG - eQ$;

Adapted from:
https://medium.com/cryptoadvance/how-schnorr-signatures-may-improve-bitcoin-91655bcb4744

Elliptic curve Schnorr signature algorithm

ECSSA_VER($(r, s), m, Q$):

1. **If** $r \notin \{1, \ldots, p-1\}$ or
   $s \notin \{1, \ldots, n-1\}$:
   return False;
2. $e \leftarrow \text{hash}(r||Q||m) \pmod{n}$;
3. $K \leftarrow sG - eQ$;

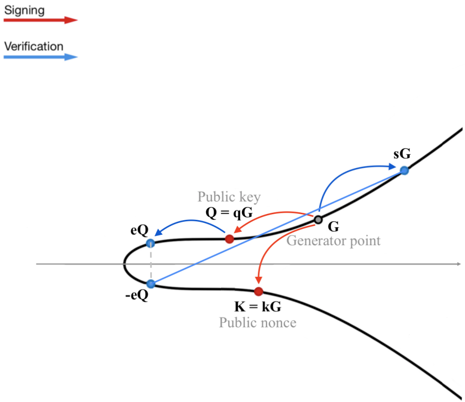


Adapted from:
`https://medium.com/cryptoadvance/how-schnorr-signatures-may-improve-bitcoin-91655bcb4744`

Elliptic curve Schnorr signature algorithm

ECSSA_VER$((r, s), m, Q)$:

1. **If** $r \notin \{1, \ldots, p-1\}$ or $s \notin \{1, \ldots, n-1\}$:
   return False;
2. $e \leftarrow \text{hash}(r||Q||m) \pmod{n}$;
3. $K \leftarrow sG - eQ$;

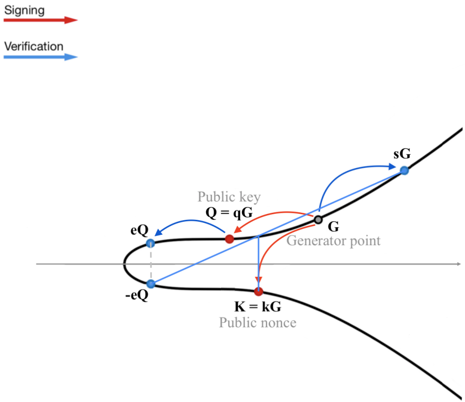


Adapted from:
`https://medium.com/cryptoadvance/how-schnorr-signatures-may-improve-bitcoin-91655bcb4744`

Elliptic curve Schnorr signature algorithm

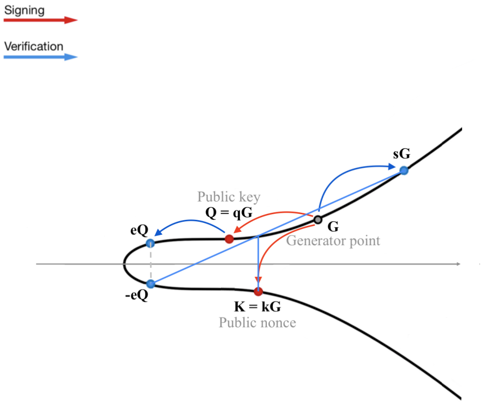
ECSSA_VER$((r, s), m, Q)$:

1. **If** $r \notin \{1, ..., p-1\}$ or $s \notin \{1, ..., n-1\}$:
   return False;
2. $e \leftarrow \text{hash}(r||Q||m) \pmod{n}$;
3. $K \leftarrow sG - eQ$;



Adapted from:
`https://medium.com/cryptoadvance/how-schnorr-signatures-may-improve-bitcoin-91655bcb4744`

Elliptic curve Schnorr signature algorithm

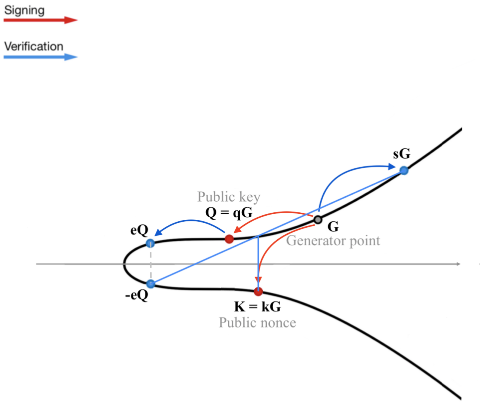


ECSSA_VER($(r, s), m, Q$):

1. **If** $r \notin \{1, ..., p-1\}$ or $s \notin \{1, ..., n-1\}$:
   return False;
2. $e \leftarrow \text{hash}(r||Q||m) \pmod{n}$;
3. $K \leftarrow sG - eQ$;
4. **If** $\text{jacobi}(y_K) \neq 1$ **or** $x_K \neq r$:
   return False;

Adapted from:
`https://medium.com/cryptoadvance/how-schnorr-signatures-may-improve-bitcoin-91655bcb4744`

Elliptic curve Schnorr signature algorithm



ECSSA_VER$((r, s), m, Q)$:

1. **If** $r \notin \{1, ..., p-1\}$ or $s \notin \{1, ..., n-1\}$:
   return False;
2. $e \leftarrow \text{hash}(r||Q||m) \pmod{n}$;
3. $K \leftarrow sG - eQ$;
4. **If** $\text{jacobi}(y_K) \neq 1$ **or** $x_K \neq r$:
   return False;
5. **return True**.

Adapted from:
`https://medium.com/cryptoadvance/how-schnorr-signatures-may-improve-bitcoin-91655bcb4744`

ECDSA vs. ECSSA

ECDSA:

ECSSA:

ECDSA vs. ECSSA

ECDSA:

- Malleable: given (r, s) also $(r, -s \pmod{n})$ is a valid signature for same message and public key;

ECSSA:

- Provably secure (SUF-CMA) in the Random Oracle Model assuming the ECDLP is hard $\implies$ not malleable;

ECDSA vs. ECSSA

ECDSA:

- Malleable: given (r, s) also $(r, -s \pmod{n})$ is a valid signature for same message and public key;
- DER encoding: variable length, up to 73 bytes;

ECSSA:

- Provably secure (SUF-CMA) in the Random Oracle Model assuming the ECDLP is hard $\Longrightarrow$ not malleable;
- New encoding: fixed length, always 64 bytes;

ECDSA vs. ECSSA

ECDSA:

- Malleable: given (r, s) also $(r, -s \pmod{n})$ is a valid signature for same message and public key;
- DER encoding: variable length, up to 73 bytes;
- Cannot be validate faster in batch;

ECSSA:

- Provably secure (SUF-CMA) in the Random Oracle Model assuming the ECDLP is hard $\implies$ not malleable;
- New encoding: fixed length, always 64 bytes;
- Batch validation scales logarithmically;

ECDSA vs. ECSSA

ECDSA:

- Malleable: given (r, s) also $(r, -s \pmod{n})$ is a valid signature for same message and public key;
- DER encoding: variable length, up to 73 bytes;
- Cannot be validate faster in batch;
- Requires the calculation of modular inverses;

ECSSA:

- Provably secure (SUF-CMA) in the Random Oracle Model assuming the ECDLP is hard $\implies$ not malleable;
- New encoding: fixed length, always 64 bytes;
- Batch validation scales logarithmically;
- No computational heavy operations involved;

ECDSA vs. ECSSA

ECDSA:

- Malleable: given (r, s) also $(r, -s \pmod{n})$ is a valid signature for same message and public key;
- DER encoding: variable length, up to 73 bytes;
- Cannot be validate faster in batch;
- Requires the calculation of modular inverses;
- Not linear: very complex higher level constructions.

ECSSA:

- Provably secure (SUF-CMA) in the Random Oracle Model assuming the ECDLP is hard $\implies$ not malleable;
- New encoding: fixed length, always 64 bytes;
- Batch validation scales logarithmically;
- No computational heavy operations involved;
- Linear: easier higher level constructions.

Batch validation

A signature (K, s) is valid if $K = sG - \text{hash}(x_K \mid\mid Q \mid\mid m)Q$.
Thus, two valid signatures (K_0, s_0) and (K_1, s_1) satisfies:

$$K_0+K_1 = (s_0+s_1)G-\text{hash}(x_{K_0} \mid\mid Q_0 \mid\mid m_0)Q_0-\text{hash}(x_{K_1} \mid\mid Q_1 \mid\mid m_1)Q_1.$$

Insecure: introduction of random factors.

$$a_0K_0 + a_1K_1 =$$

$$= (a_0s_0+a_1s_1)G-a_0\text{hash}(x_{K_0} \mid\mid Q_0 \mid\mid m_0)Q_0-a_1\text{hash}(x_{K_1} \mid\mid Q_1 \mid\mid m_1)Q_1.$$

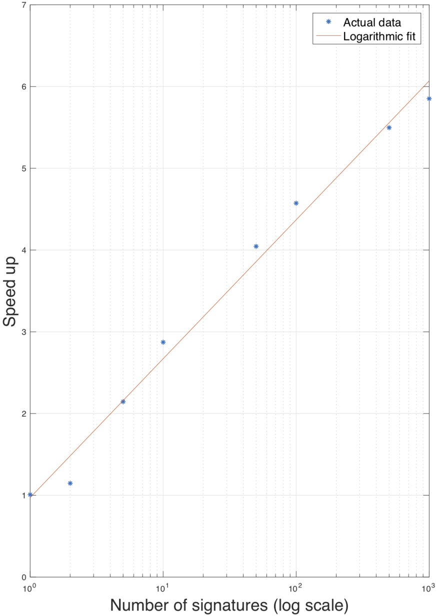
Batch validation - Bos-Coster's algorithm

$$a_0 K_0 + a_1 K_1 =$$
$$= (a_0 - a_1) K_0 + a_1 (K_0 + K_1).$$

Batch validation - Bos-Coster's algorithm

$$a_0 K_0 + a_1 K_1 =$$
$$= (a_0 - a_1)K_0 + a_1(K_0 + K_1).$$

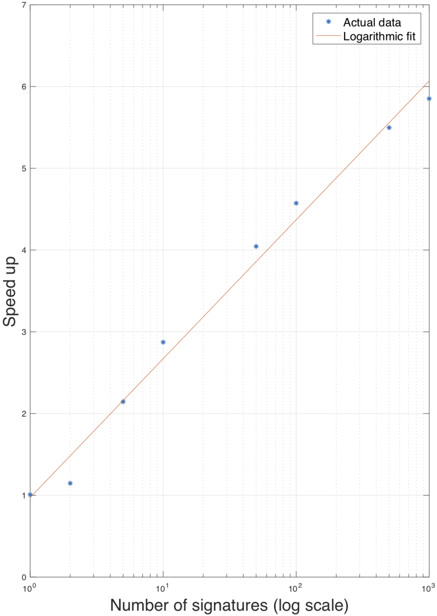
- Sort the tuples according to a_i in descending order;



Batch validation - Bos-Coster's algorithm

$$a_0K_0 + a_1K_1 =$$
$$= (a_0 - a_1)K_0 + a_1(K_0 + K_1).$$

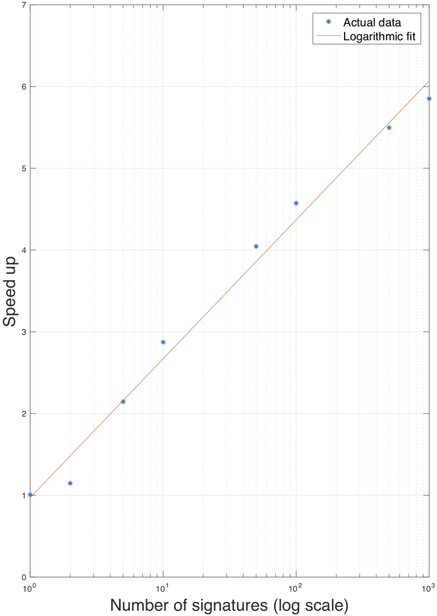
- Sort the tuples according to a_i in descending order;
- While the list has length larger than one:



Batch validation - Bos-Coster's algorithm

$$a_0 K_0 + a_1 K_1 =$$
$$= (a_0 - a_1) K_0 + a_1 (K_0 + K_1).$$

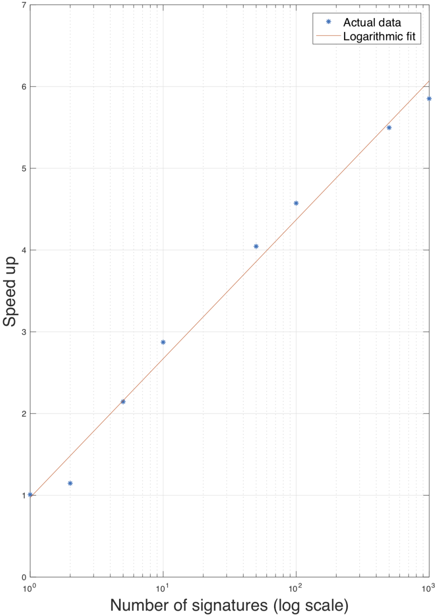
- Sort the tuples according to a_i in descending order;
- While the list has length larger than one:
  - Substitute (a_0, K_0) and (a_1, K_1) with $(a_0 - a_1, K_0)$ and $(a_1, K_0 + K_1)$;



Batch validation - Bos-Coster's algorithm

$$a_0 K_0 + a_1 K_1 =$$
$$= (a_0 - a_1)K_0 + a_1(K_0 + K_1).$$

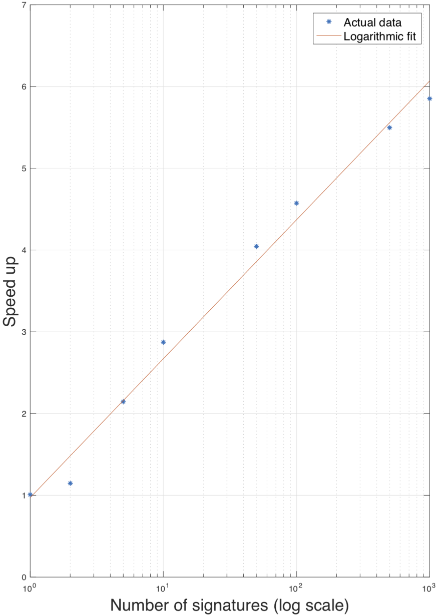
- Sort the tuples according to a_i in descending order;
- While the list has length larger than one:
  - Substitute (a_0, K_0) and (a_1, K_1) with $(a_0 - a_1, K_0)$ and $(a_1, K_0 + K_1)$;
  - Sort the list again;



Batch validation - Bos-Coster's algorithm

$$a_0 K_0 + a_1 K_1 =$$
$$= (a_0 - a_1) K_0 + a_1 (K_0 + K_1).$$

- Sort the tuples according to a_i in descending order;
- While the list has length larger than one:
  - Substitute (a_0, K_0) and (a_1, K_1) with $(a_0 - a_1, K_0)$ and $(a_1, K_0 + K_1)$;
  - Sort the list again;
- When only one element remains, with very large probability it will be of the form $(1, K)$, otherwise it will be of the form (a, K).



Outline

Multi-signature schemes

Multi-signature schemes allow a group of users to cooperate to sign a single message: they are fundamental in real life applications.

Multi-signature schemes

Multi-signature schemes allow a group of users to cooperate to sign a single message: they are fundamental in real life applications.

Bitcoin multi-signature (t-of-m) is implemented naively:

Multi-signature schemes

Multi-signature schemes allow a group of users to cooperate to sign a single message: they are fundamental in real life applications.

Bitcoin multi-signature (t-of-m) is implemented naively:

- Locking script: t <pubKey1> <pubKey2> ... <pubKeym>
  m OP_CHECKMULTISIG

Multi-signature schemes

Multi-signature schemes allow a group of users to cooperate to sign a single message: they are fundamental in real life applications.

Bitcoin multi-signature (t-of-m) is implemented naively:

- Locking script: t <pubKey1> <pubKey2> ... <pubKeym> m OP_CHECKMULTISIG
- Unlocking script: 0 <sig1> <sig2> ... <sigt>

Multi-signature schemes

Multi-signature schemes allow a group of users to cooperate to sign a single message: they are fundamental in real life applications.

Bitcoin multi-signature (t-of-m) is implemented naively:

- Locking script: t <pubKey1> <pubKey2> ... <pubKeym>
  m OP_CHECKMULTISIG
- Unlocking script: 0 <sig1> <sig2> ... <sigt>

Schnorr multi-signature (2-of-2) implemented naively:

Multi-signature schemes

Multi-signature schemes allow a group of users to cooperate to sign a single message: they are fundamental in real life applications.

Bitcoin multi-signature (t-of-m) is implemented naively:

- Locking script: t <pubKey1> <pubKey2> ... <pubKeym> m OP_CHECKMULTISIG
- Unlocking script: 0 <sig1> <sig2> ... <sigt>

Schnorr multi-signature (2-of-2) implemented naively:

- Alice ($\{q_A, Q_A\}$) and Bob ($\{q_B, Q_B\}$) generates K_A and K_B;

Multi-signature schemes

Multi-signature schemes allow a group of users to cooperate to sign a single message: they are fundamental in real life applications.

Bitcoin multi-signature (t-of-m) is implemented naively:

- Locking script: t <pubKey1> <pubKey2> ... <pubKeym> m OP_CHECKMULTISIG
- Unlocking script: 0 <sig1> <sig2> ... <sigt>

Schnorr multi-signature (2-of-2) implemented naively:

- Alice ($\{q_A, Q_A\}$) and Bob ($\{q_B, Q_B\}$) generates K_A and K_B;
- They exchange them and set the public nonce at $K = K_A + K_B$. The joint public key is set at $Q = Q_A + Q_B$;

Multi-signature schemes

Multi-signature schemes allow a group of users to cooperate to sign a single message: they are fundamental in real life applications.

Bitcoin multi-signature (t-of-m) is implemented naively:

- Locking script: t <pubKey1> <pubKey2> ... <pubKeym> m OP_CHECKMULTISIG
- Unlocking script: 0 <sig1> <sig2> ... <sigt>

Schnorr multi-signature (2-of-2) implemented naively:

- Alice ($\{q_A, Q_A\}$) and Bob ($\{q_B, Q_B\}$) generates K_A and K_B;
- They exchange them and set the public nonce at $K = K_A + K_B$. The joint public key is set at $Q = Q_A + Q_B$;
- Their partial signatures are: $s_i = k_i + \mathrm{hash}(x_K||Q||m)q_i, \; i \in \{A, B\}$;

Multi-signature schemes

Multi-signature schemes allow a group of users to cooperate to sign a single message: they are fundamental in real life applications.

Bitcoin multi-signature (t-of-m) is implemented naively:

- Locking script: t <pubKey1> <pubKey2> ... <pubKeym> m OP_CHECKMULTISIG
- Unlocking script: 0 <sig1> <sig2> ... <sigt>

Schnorr multi-signature (2-of-2) implemented naively:

- Alice ($\{q_A, Q_A\}$) and Bob ($\{q_B, Q_B\}$) generates K_A and K_B;
- They exchange them and set the public nonce at $K = K_A + K_B$. The joint public key is set at $Q = Q_A + Q_B$;
- Their partial signatures are: $s_i = k_i + \text{hash}(x_K||Q||m)q_i,\ i \in \{A, B\}$;
- The signature $(x_K, s_A + s_B)$ is valid on m for public key Q.

Multi-signature schemes

Multi-signature schemes allow a group of users to cooperate to sign a single message: they are fundamental in real life applications.

Bitcoin multi-signature (t-of-m) is implemented naively:

- Locking script: t <pubKey1> <pubKey2> ... <pubKeym> m OP_CHECKMULTISIG
- Unlocking script: 0 <sig1> <sig2> ... <sigt>

Schnorr multi-signature (2-of-2) implemented naively:

- Alice ($\{q_A, Q_A\}$) and Bob ($\{q_B, Q_B\}$) generates K_A and K_B;
- They exchange them and set the public nonce at $K = K_A + K_B$. The joint public key is set at $Q = Q_A + Q_B$;
- Their partial signatures are: $s_i = k_i + \text{hash}(x_K||Q||m)q_i,\ i \in \{A, B\}$;
- The signature $(x_K, s_A + s_B)$ is valid on m for public key Q.

INSECURE: rogue key attack!

MuSig: compact m-of-m signature scheme

$\text{MuSig}(m, q_1, \langle L \rangle)$:

1: Alice

2: Bob

3: Charlotte

MuSig: compact m-of-m signature scheme

MuSig($m, q_1, \langle L \rangle$):

1. **for** $i \leftarrow 1, m$ **do**:
    1.1 $a_i \leftarrow \text{hash}(\langle L \rangle || Q_i)$;

1: Alice

a_1, a_2

2: Bob

3: Charlotte

MuSig: compact m-of-m signature scheme

MuSig$(m, q_1, \langle L \rangle)$:

1. **for** $i \leftarrow 1, m$ **do**:
   1.1 $a_i \leftarrow \text{hash}(\langle L \rangle || Q_i)$;
2. $Q \leftarrow \sum_{i=1}^{m} a_i Q_i$;

Q

1: Alice

2: Bob

3: Charlotte

MuSig: compact m-of-m signature scheme

MuSig$(m, q_1, \langle L \rangle)$:

1. **for** $i \leftarrow 1, m$ **do**:
    1.1 $a_i \leftarrow \text{hash}(\langle L \rangle || Q_i)$;
2. $Q \leftarrow \sum_{i=1}^{m} a_i Q_i$;
3. $k_1 \xleftarrow{\$} \{1, ..., n-1\}$;

k_1

1: Alice

2: Bob

3: Charlotte

MuSig: compact m-of-m signature scheme

MuSig$(m, q_1, \langle L \rangle)$:

1. **for** $i \leftarrow 1, m$ **do**:
   1.1 $a_i \leftarrow \text{hash}(\langle L \rangle || Q_i)$;
2. $Q \leftarrow \sum_{i=1}^{m} a_i Q_i$;
3. $k_1 \xleftarrow{\$} \{1, ..., n-1\}$;
4. $K_1 \leftarrow k_1 G,\ \ t_1 \leftarrow \text{hash}(K_1)$;

K_1, t_1

1: Alice

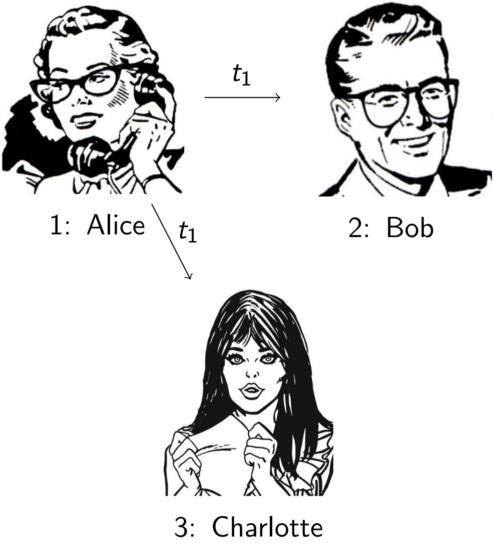
2: Bob

3: Charlotte

MuSig: compact m-of-m signature scheme

MuSig$(m, q_1, \langle L \rangle)$:

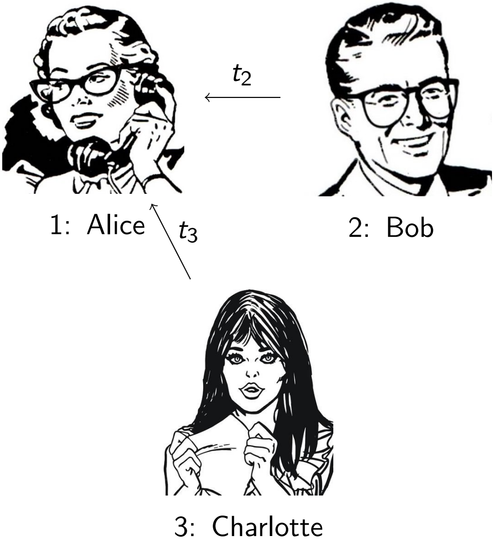
1. **for** $i \leftarrow 1, m$ **do**:
    1.1 $a_i \leftarrow \text{hash}(\langle L \rangle || Q_i)$;
2. $Q \leftarrow \sum_{i=1}^{m} a_i Q_i$;
3. $k_1 \xleftarrow{\$} \{1, ..., n-1\}$;
4. $K_1 \leftarrow k_1 G,\ \ t_1 \leftarrow \text{hash}(K_1)$;
5. **send** t_1, K_1;



MuSig: compact m-of-m signature scheme

MuSig$(m, q_1, \langle L \rangle)$:

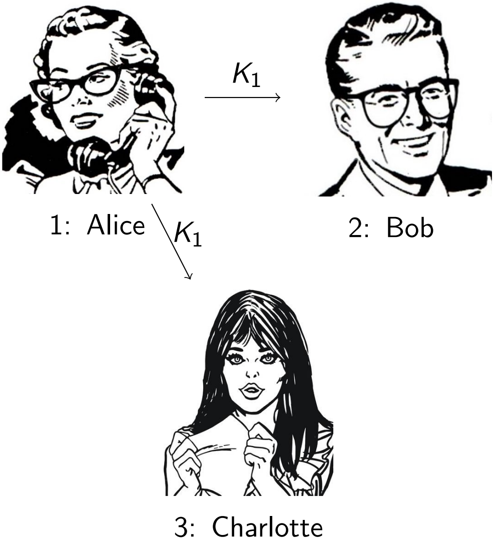
1. **for** $i \leftarrow 1, m$ **do**:
   - 1.1 $a_i \leftarrow \text{hash}(\langle L \rangle || Q_i)$;
2. $Q \leftarrow \sum_{i=1}^{m} a_i Q_i$;
3. $k_1 \xleftarrow{\$} \{1, \ldots, n-1\}$;
4. $K_1 \leftarrow k_1 G,\ \ t_1 \leftarrow \text{hash}(K_1)$;
5. **send** t_1, K_1;



MuSig: compact *m*-of-*m* signature scheme

MuSig$(m, q_1, \langle L \rangle)$:

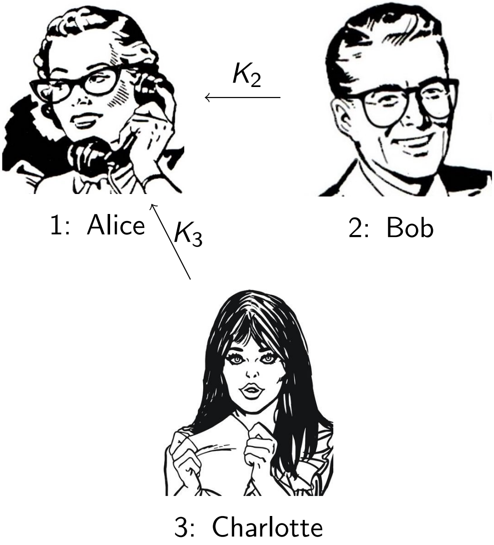
1. **for** $i \leftarrow 1, m$ **do**:
   1.1 $a_i \leftarrow \text{hash}(\langle L \rangle || Q_i)$;
2. $Q \leftarrow \sum_{i=1}^{m} a_i Q_i$;
3. $k_1 \xleftarrow{\$} \{1, \ldots, n-1\}$;
4. $K_1 \leftarrow k_1 G$, $t_1 \leftarrow \text{hash}(K_1)$;
5. **send** t_1, K_1;



MuSig: compact m-of-m signature scheme

MuSig$(m, q_1, \langle L \rangle)$:

1. **for** $i \leftarrow 1, m$ **do**:
   1.1 $a_i \leftarrow \text{hash}(\langle L \rangle || Q_i)$;
2. $Q \leftarrow \sum_{i=1}^{m} a_i Q_i$;
3. $k_1 \xleftarrow{\$} \{1, \ldots, n-1\}$;
4. $K_1 \leftarrow k_1 G,\ \ t_1 \leftarrow \text{hash}(K_1)$;
5. **send** t_1, K_1;



MuSig: compact *m*-of-*m* signature scheme

MuSig$(m, q_1, \langle L \rangle)$:

1. **for** $i \leftarrow 1, m$ **do**:
   1.1 $a_i \leftarrow \text{hash}(\langle L \rangle || Q_i)$;
2. $Q \leftarrow \sum_{i=1}^{m} a_i Q_i$;
3. $k_1 \xleftarrow{\$} \{1, \ldots, n-1\}$;
4. $K_1 \leftarrow k_1 G,\ \ t_1 \leftarrow \text{hash}(K_1)$;
5. **send** t_1, K_1;

1: Alice

$t_2 \stackrel{?}{=} \text{hash}(K_2)$

$t_3 \stackrel{?}{=} \text{hash}(K_3)$

2: Bob

3: Charlotte

MuSig: compact m-of-m signature scheme

MuSig($m, q_1, \langle L \rangle$):

1. **for** $i \leftarrow 1, m$ **do**:
   1.1 $a_i \leftarrow \text{hash}(\langle L \rangle || Q_i)$;
2. $Q \leftarrow \sum_{i=1}^{m} a_i Q_i$;
3. $k_1 \xleftarrow{\$} \{1, ..., n-1\}$;
4. $K_1 \leftarrow k_1 G,\ \ t_1 \leftarrow \text{hash}(K_1)$;
5. **send** t_1, K_1;
6. $K \leftarrow \sum_{i=1}^{m} K_i$;

K

1: Alice

2: Bob

3: Charlotte

MuSig: compact m-of-m signature scheme

MuSig$(m, q_1, \langle L \rangle)$:

1. **for** $i \leftarrow 1, m$ **do**:
   1.1 $a_i \leftarrow \text{hash}(\langle L \rangle || Q_i)$;
2. $Q \leftarrow \sum_{i=1}^{m} a_i Q_i$;
3. $k_1 \xleftarrow{\$} \{1, \ldots, n-1\}$;
4. $K_1 \leftarrow k_1 G, \;\; t_1 \leftarrow \text{hash}(K_1)$;
5. **send** t_1, K_1;
6. $K \leftarrow \sum_{i=1}^{m} K_i$;

if jacobi$(y_K) \neq 1$:
$k_1 \leftarrow n - k_1$

1: Alice

2: Bob

3: Charlotte

MuSig: compact m-of-m signature scheme

MuSig$(m, q_1, \langle L \rangle)$:

1. **for** $i \leftarrow 1, m$ **do**:
   1.1 $a_i \leftarrow \text{hash}(\langle L \rangle || Q_i)$;
2. $Q \leftarrow \sum_{i=1}^{m} a_i Q_i$;
3. $k_1 \xleftarrow{\$} \{1, \ldots, n-1\}$;
4. $K_1 \leftarrow k_1 G$, $t_1 \leftarrow \text{hash}(K_1)$;
5. **send** t_1, K_1;
6. $K \leftarrow \sum_{i=1}^{m} K_i$;
7. $c \leftarrow \text{hash}(x_K || Q || m)$;

c

1: Alice

2: Bob

3: Charlotte

MuSig: compact *m*-of-*m* signature scheme

MuSig$(m, q_1, \langle L \rangle)$:

1. **for** $i \leftarrow 1, m$ **do**:
    1.1 $a_i \leftarrow \text{hash}(\langle L \rangle || Q_i)$;
2. $Q \leftarrow \sum_{i=1}^{m} a_i Q_i$;
3. $k_1 \xleftarrow{\$} \{1, \ldots, n-1\}$;
4. $K_1 \leftarrow k_1 G,\ \ t_1 \leftarrow \text{hash}(K_1)$;
5. **send** t_1, K_1;
6. $K \leftarrow \sum_{i=1}^{m} K_i$;
7. $c \leftarrow \text{hash}(x_K || Q || m)$;
8. $s_1 \leftarrow k_1 + c a_1 q_1 \pmod{n}$;

1: Alice

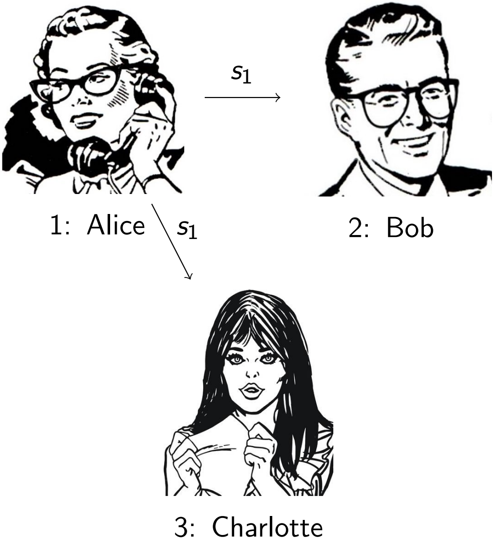
2: Bob

3: Charlotte

MuSig: compact m-of-m signature scheme

MuSig($m, q_1, \langle L \rangle$):

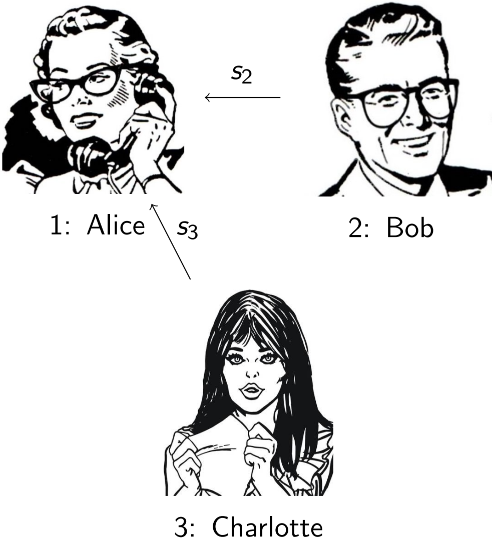
1. **for** $i \leftarrow 1, m$ **do**:
   1.1 $a_i \leftarrow \text{hash}(\langle L \rangle || Q_i)$;
2. $Q \leftarrow \sum_{i=1}^{m} a_i Q_i$;
3. $k_1 \xleftarrow{\$} \{1, \ldots, n-1\}$;
4. $K_1 \leftarrow k_1 G, \;\; t_1 \leftarrow \text{hash}(K_1)$;
5. **send** t_1, K_1;
6. $K \leftarrow \sum_{i=1}^{m} K_i$;
7. $c \leftarrow \text{hash}(x_K || Q || m)$;
8. $s_1 \leftarrow k_1 + c a_1 q_1 \pmod{n}$;
9. **send** s_1;



MuSig: compact m-of-m signature scheme

MuSig$(m, q_1, \langle L \rangle)$:

1. **for** $i \leftarrow 1, m$ **do**:
   1.1 $a_i \leftarrow \text{hash}(\langle L \rangle || Q_i)$;
2. $Q \leftarrow \sum_{i=1}^{m} a_i Q_i$;
3. $k_1 \xleftarrow{\$} \{1, \ldots, n-1\}$;
4. $K_1 \leftarrow k_1 G, \;\; t_1 \leftarrow \text{hash}(K_1)$;
5. **send** t_1, K_1;
6. $K \leftarrow \sum_{i=1}^{m} K_i$;
7. $c \leftarrow \text{hash}(x_K || Q || m)$;
8. $s_1 \leftarrow k_1 + c a_1 q_1 \pmod{n}$;
9. **send** s_1;



MuSig: compact m-of-m signature scheme

MuSig$(m, q_1, \langle L \rangle)$:

1. **for** $i \leftarrow 1, m$ **do**:
   1.1 $a_i \leftarrow \text{hash}(\langle L \rangle || Q_i)$;
2. $Q \leftarrow \sum_{i=1}^{m} a_i Q_i$;
3. $k_1 \xleftarrow{\$} \{1, ..., n-1\}$;
4. $K_1 \leftarrow k_1 G$, $t_1 \leftarrow \text{hash}(K_1)$;
5. **send** t_1, K_1;
6. $K \leftarrow \sum_{i=1}^{m} K_i$;
7. $c \leftarrow \text{hash}(x_K || Q || m)$;
8. $s_1 \leftarrow k_1 + c a_1 q_1 \pmod{n}$;
9. **send** s_1;
10. $s \leftarrow \sum_{i=1}^{m} s_i \pmod{n}$;

1: Alice

2: Bob

3: Charlotte

MuSig: compact m-of-m signature scheme

MuSig$(m, q_1, \langle L\rangle)$:

1. **for** $i \leftarrow 1, m$ **do**:
   1.1 $a_i \leftarrow \text{hash}(\langle L\rangle || Q_i)$;
2. $Q \leftarrow \sum_{i=1}^{m} a_i Q_i$;
3. $k_1 \xleftarrow{\$} \{1, \ldots, n-1\}$;
4. $K_1 \leftarrow k_1 G,\ \ t_1 \leftarrow \text{hash}(K_1)$;
5. **send** t_1, K_1;
6. $K \leftarrow \sum_{i=1}^{m} K_i$;
7. $c \leftarrow \text{hash}(x_K || Q || m)$;
8. $s_1 \leftarrow k_1 + c a_1 q_1 \pmod{n}$;
9. **send** s_1;
10. $s \leftarrow \sum_{i=1}^{m} s_i \pmod{n}$;
11. **return** (x_K, s).

1: Alice

(x_K, s)

2: Bob

3: Charlotte

MuSig: compact m-of-m signature scheme

MuSig ($\mu\Sigma$):

- Compact: same size as the single user case;

1: Alice

2: Bob

3: Charlotte

MuSig: compact m-of-m signature scheme

MuSig ($\mu\Sigma$):

- Compact: same size as the single user case;
- Secure in the plain public key model: cross input aggregation at transaction level;

1: Alice

2: Bob

3: Charlotte

MuSig: compact m-of-m signature scheme

MuSig ($\mu\Sigma$):

- Compact: same size as the single user case;
- Secure in the plain public key model: cross input aggregation at transaction level;
- Key aggregation: signature indistinguishable from the single user case.

1: Alice

2: Bob

3: Charlotte

Threshold signature scheme (t-of-m)

Verifiable secret sharing scheme

Alice

Bob

Charlotte

Threshold signature scheme (t-of-m)

Verifiable secret sharing scheme

↓

Protocol for the generation of a random shared secret

Alice

Bob

Charlotte

Threshold signature scheme (t-of-m)

Verifiable secret sharing scheme

↓

Protocol for the generation of a random shared secret

↓

Threshold scheme

Alice

Bob

Charlotte

Threshold signature scheme (t-of-m)

Verifiable secret sharing scheme

The dealer:

Dealer: Alice

1: Bob

2: Charlotte

Threshold signature scheme (t-of-m)

Verifiable secret sharing scheme

The dealer:

- generates secret s and $s' \xleftarrow{\$} \{1, ..., n-1\}$;

Dealer: Alice

s, s'

1: Bob

2: Charlotte

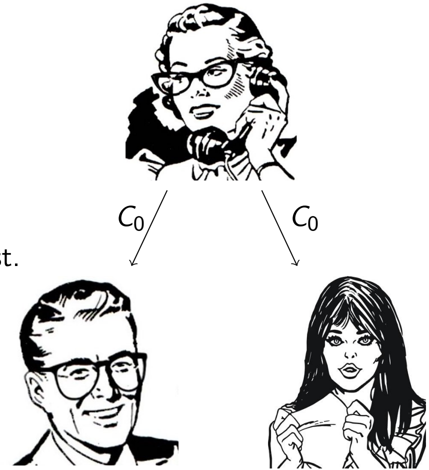
Threshold signature scheme (t-of-m)

Verifiable secret sharing scheme

The dealer:

- generates secret s and $s' \xleftarrow{\$} \{1, \ldots, n-1\}$;
- commits to them through the Pedersen commitment $C_0 = sG + s'H$: C_0 is broadcast.

Dealer: Alice



1: Bob

2: Charlotte

Threshold signature scheme (t-of-m)

Verifiable secret sharing scheme

The dealer:

- chooses random polynomials:
  $f(u) = s + f_1 u + ... + f_{t-1} u^{t-1}$,
  $f'(u) = s' + f'_1 u + ... + f'_{t-1} u^{t-1}$,
  $f_j,\ f'_j \xleftarrow{\$} \{1, ..., n-1\}$;

Dealer: Alice

f, f'

1: Bob

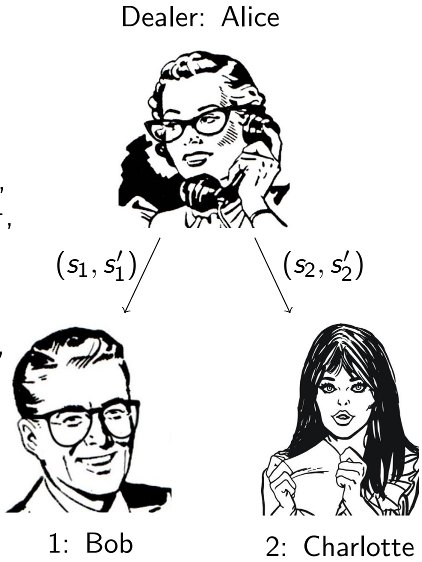
2: Charlotte

Threshold signature scheme (t-of-m)

Verifiable secret sharing scheme

The dealer:

- chooses random polynomials:
  $f(u) = s + f_1 u + ... + f_{t-1} u^{t-1}$,
  $f'(u) = s' + f'_1 u + ... + f'_{t-1} u^{t-1}$,
  $f_j,\ f'_j \xleftarrow{\$} \{1, ..., n-1\}$;
- computes $(s_i, s'_i) = (f(i) \pmod{n}, f'(i) \pmod{n})$, $i \in \{1, ..., m\}$ and sends them secretly to P_i;



Threshold signature scheme (t-of-m)

Verifiable secret sharing scheme

The dealer:

- chooses random polynomials:
  $f(u) = s + f_1 u + ... + f_{t-1} u^{t-1}$,
  $f'(u) = s' + f'_1 u + ... + f'_{t-1} u^{t-1}$,
  $f_j,\ f'_j \xleftarrow{\$} \{1, ..., n-1\}$;
- computes $(s_i, s'_i) = (f(i) \pmod{n}, f'(i) \pmod{n})$, $i \in \{1, ..., m\}$ and sends them secretly to P_i;
- broadcasts the commitment to the sharing polynomials:
  $C_j = f_j G + f'_j H$,
  $j \in \{1, ..., t-1\}$.

Dealer: Alice

1: Bob

2: Charlotte

Threshold signature scheme (t-of-m)

Verifiable secret sharing scheme

The participants:

Dealer: Alice

1: Bob

2: Charlotte

Threshold signature scheme (t-of-m)

Verifiable secret sharing scheme

The participants:

- verify the consistency of their shares of secret: $s_i G + s_i' H = \sum_{j=0}^{t-1} i^j C_j$;

Dealer: Alice

1: Bob

2: Charlotte

Threshold signature scheme (t-of-m)

Verifiable secret sharing scheme

The participants:

- verify the consistency of their shares of secret: $s_i G + s_i' H = \sum_{j=0}^{t-1} i^j C_j$;
- to reconstruct the secret they rely on Lagrange's interpolation formula:
  $f(u) = \sum_i f(i)\omega_i(u)$, where
  $\omega_i(u) = \prod_{j \neq i} \frac{u-j}{i-j} \pmod{n}$.
  $s = f(0) = \sum_i s_i \omega_i$, with
  $\omega_i = \omega_i(0) = \prod_{j \neq i} \frac{j}{j-i} \pmod{n}$.

Dealer: Alice

$\overset{s}{\longleftrightarrow}$

1: Bob

2: Charlotte

Threshold signature scheme (t-of-m)

Protocol for the generation of a random shared secret

Each participant:

1: Alice

2: Bob

3: Charlotte

Threshold signature scheme (t-of-m)

Protocol for the generation of a random shared secret

Each participant:

- acts as the dealer in the previous protocol ($f_i(u) = \sum_{j=0}^{t-1} a_{ij} u^j$, $a_{i0} = r_i$);

1: Alice

r_1, f_1

r_2, f_2

2: Bob

r_3, f_3

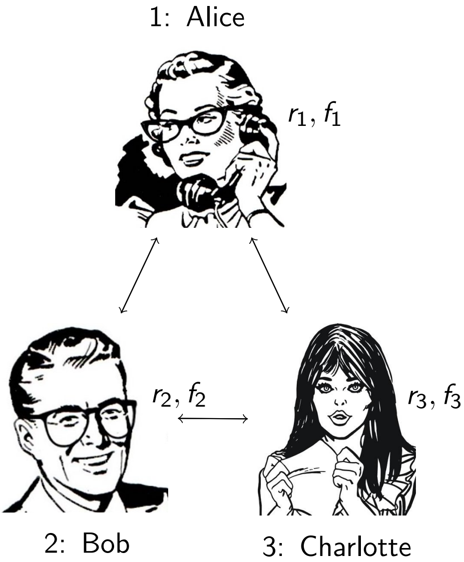
3: Charlotte

Threshold signature scheme (t-of-m)

Protocol for the generation of a random shared secret

Each participant:

- acts as the dealer in the previous protocol ($f_i(u) = \sum_{j=0}^{t-1} a_{ij} u^j$, $a_{i0} = r_i$);



Threshold signature scheme (t-of-m)

Protocol for the generation of a random shared secret

Each participant:

- acts as the dealer in the previous protocol ($f_i(u) = \sum_{j=0}^{t-1} a_{ij} u^j$, $a_{i0} = r_i$);
- the shared secret is $r = \sum_{i=1}^{m} r_i \pmod{n}$ with shares $s_i = \sum_{j=1}^{m} f_j(i) \pmod{n}$;

1: Alice

s_1

s_2

2: Bob

s_3

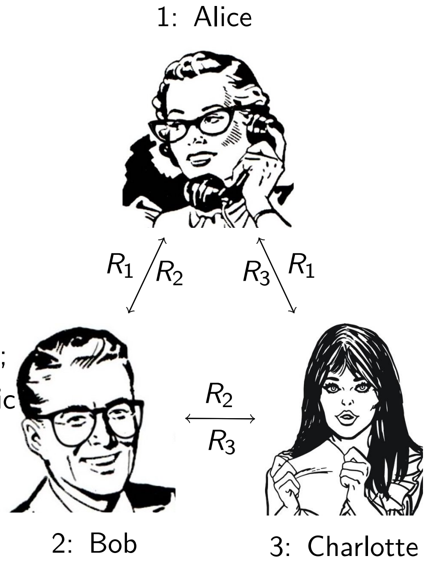
3: Charlotte

Threshold signature scheme (t-of-m)

Protocol for the generation of a random shared secret

Each participant:

- acts as the dealer in the previous protocol ($f_i(u) = \sum_{j=0}^{t-1} a_{ij} u^j$, $a_{i0} = r_i$);
- the shared secret is $r = \sum_{i=1}^{m} r_i \pmod{n}$ with shares $s_i = \sum_{j=1}^{m} f_j(i) \pmod{n}$;
- broadcast his share of the public key $R_j = r_j G$ ($R = \sum_{j=1}^{m} R_j = \sum_{j=1}^{m} r_j G = rG$).



Threshold signature scheme (t-of-m)

Threshold scheme

After having established a distributed key pair $(\alpha_1, \ldots, \alpha_m) \xleftrightarrow{\text{(t, m)}} (q|Q)$ through the protocol for the generation of a random shared secret (that acts as key generation protocol) the signers:

1: Alice

α_1

$\updownarrow$

α_3

3: Charlotte

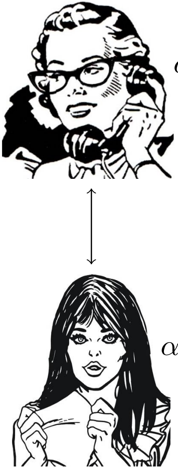
Threshold signature scheme (t-of-m)

Threshold scheme

After having established a distributed key pair $(\alpha_1, ..., \alpha_m) \xleftrightarrow{\text{(t, m)}} (q|Q)$ through the protocol for the generation of a random shared secret (that acts as key generation protocol) the signers:

- run again the same protocol to produce a nonces pair: $(\beta_1, ..., \beta_m) \xleftrightarrow{\text{(t, m)}} (k|K)$.

1: Alice



α_1, β_1

α_3, β_3

3: Charlotte

Threshold signature scheme (t-of-m)

Threshold scheme

Then each signer i:

- checks whether jacobi$(y_K) \neq 1$; if it is the case she sets $\beta_i = n - \beta_i$;

1: Alice

3: Charlotte

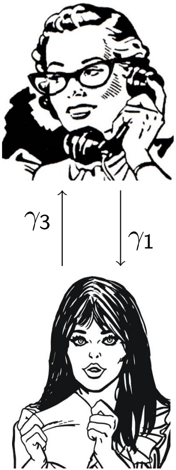
Threshold signature scheme (t-of-m)

Threshold scheme

Then each signer i:

- checks whether $\mathrm{jacobi}(y_K) \neq 1$;
  if it is the case she sets
  $\beta_i = n - \beta_i$;
- reveals her partial signature:
  $\gamma_i = \beta_i + e\alpha_i \pmod{n}$, with
  $e = \mathrm{hash}(x_K||Q||msg)$;

1: Alice



3: Charlotte

Threshold signature scheme (t-of-m)

Threshold scheme

Then each signer i:

- checks whether $\text{jacobi}(y_K) \neq 1$; if it is the case she sets $\beta_i = n - \beta_i$;
- reveals her partial signature: $\gamma_i = \beta_i + e\alpha_i \pmod{n}$, with $e = \text{hash}(x_K || Q || msg)$;
- computes $\sigma = \sum_{j=1}^{t} \gamma_j \omega_j \pmod{n}$, with $\omega_j = \prod_{h \neq j} \frac{h}{h-j} \pmod{n}$: σ is such that $\sigma = k + eq \pmod{n}$;

1: Alice

σ

σ

3: Charlotte

Threshold signature scheme (t-of-m)

Threshold scheme

Then each signer i:

- checks whether jacobi$(y_K) \neq 1$; if it is the case she sets $\beta_i = n - \beta_i$;
- reveals her partial signature: $\gamma_i = \beta_i + e\alpha_i \pmod{n}$, with $e = \text{hash}(x_K||Q||msg)$;
- computes $\sigma = \sum_{j=1}^{t} \gamma_j \omega_j \pmod{n}$, with $\omega_j = \prod_{h \neq j} \frac{h}{h-j} \pmod{n}$: σ is such that $\sigma = k + eq \pmod{n}$;
- the signature is (x_K, σ).

1: Alice

(x_K, σ)

(x_K, σ)

3: Charlotte

ECDSA vs. ECSSA (multi-signature)

ECDSA:

- Locking script: t <pubKey1> <pubKey2> ... <pubKeym> m OP_CHECKMULTISIG
- Unlocking script: 0 <sig1> <sig2> ... <sigt>

ECDSA vs. ECSSA (multi-signature)

ECDSA:

- Locking script: t <pubKey1> <pubKey2> ... <pubKeym> m OP_CHECKMULTISIG
- Unlocking script: 0 <sig1> <sig2> ... <sigt>

ECSSA:

- Locking script: <jointPubKey> OP_SCHNORR
- Unlocking script: <jointSig>

Adaptor signatures

Building block for *scriptless script*: aim at encapsulating the flexibility of script semantics in fixed size signatures.

Adaptor signatures

Building block for *scriptless script*: aim at encapsulating the flexibility of script semantics in fixed size signatures.

How?

Adaptor signatures

Building block for *scriptless script*: aim at encapsulating the flexibility of script semantics in fixed size signatures.

How? The idea is to add to the public nonce K a random $T = tG$ but still consider k as private nonce: this results in an invalid signature, however learning t is equivalent to learning a valid signature.

Adaptor signatures

Building block for *scriptless script*: aim at encapsulating the flexibility of script semantics in fixed size signatures.

How? The idea is to add to the public nonce K a random $T = tG$ but still consider k as private nonce: this results in an invalid signature, however learning t is equivalent to learning a valid signature.

If t is some necessary data for the execution of a separate protocol, arbitrary steps of arbitrary protocols can be made equivalent to signature production.

Cross-chain atomic swaps

Exchange of different crypto-currencies among two distrustful users in an atomic and decentralized way.

Cross-chain atomic swaps

Exchange of different crypto-currencies among two distrustful users in an atomic and decentralized way.

This is done via Hashed TimeLock Contract (HTLC), special locking scripts that ensures the atomicity of the transactions on both blockchains.

Cross-chain atomic swaps

Exchange of different crypto-currencies among two distrustful users in an atomic and decentralized way.

This is done via Hashed TimeLock Contract (HTLC), special locking scripts that ensures the atomicity of the transactions on both blockchains.

```
 OP_IF
    OP_HASH256 <digest> OP_EQUALVERIFY OP_DUP
    OP_HASH160 <Bob address>
OP_ELSE
    <num> OP_CHECKSEQUENCEVERIFY OP_DROP
    OP_DUP OP_HASH160 <Alice addres>
OP_ENDIF
OP_EQUALVERIFY OP_CHECKSIG
```

Cross-chain atomic swaps

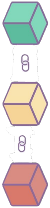
Exchange of different crypto-currencies among two distrustful users in an atomic and decentralized way.

This is done via Hashed TimeLock Contract (HTLC), special locking scripts that ensures the atomicity of the transactions on both blockchains.

```
 OP_IF
    OP_HASH256 <digest> OP_EQUALVERIFY OP_DUP
    OP_HASH160 <Bob address>
OP_ELSE
    <num> OP_CHECKSEQUENCEVERIFY OP_DROP
    OP_DUP OP_HASH160 <Alice addres>
OP_ENDIF
OP_EQUALVERIFY OP_CHECKSIG
```

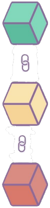
Easily identifiable (lack of privacy) and cumbersome (high fees).

Cross-chain atomic swaps via adaptor signatures

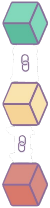


Alice

Bob



Cross-chain atomic swaps via adaptor signatures

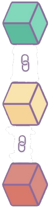


$Q_A^{฿}, Q_A^{\Xi}$

Alice

$Q_B^{฿}, Q_B^{\Xi}$

Bob



Cross-chain atomic swaps via adaptor signatures