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^{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:

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

Outline

Hash functions ($\simeq$ Random Oracle)

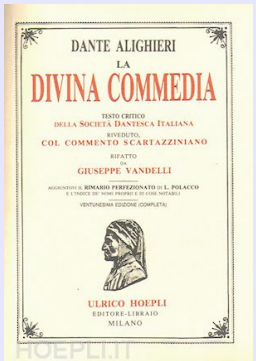
Input

Output

"Politecnico di Milano"

SHA-256

52c61456bd3d60c08599a8c61e113e7b
c39118b75d06b2643c55957ece91269b



SHA-256

1f12f8384d5941e8e273926635be646a
f786405a8662e0ba6dd1fd0526ec0528

SHA-256

721a14d89cb463b51cf20ecc707aa290
5b7c8f32d9114dc19fe353e611494bf1

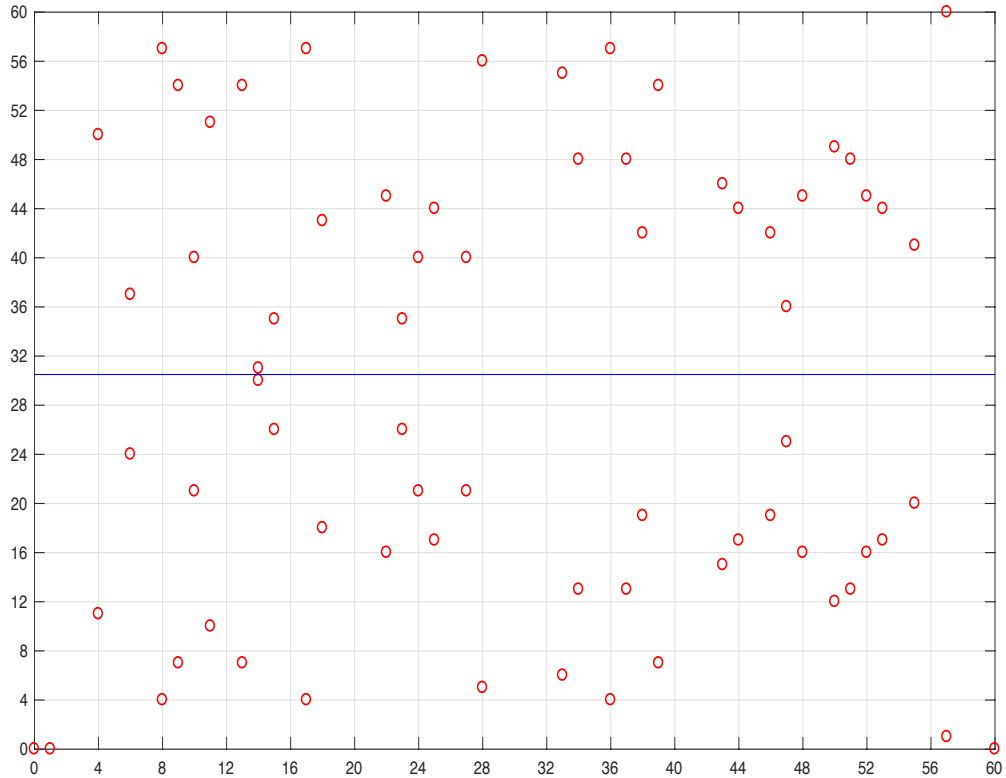
Elliptic curve cryptography

An elliptic curve over a finite field is defined by the equation:

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

It is possible to define:

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



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

Discrete logarithm problem

Fixed $G \in E(\mathbb{F}_p)$, we can define $Q = qG \quad \forall q \in \{1, ..., n-1\}$, where n is the smallest integer such that $nG = \infty$, the identity element of the sum:

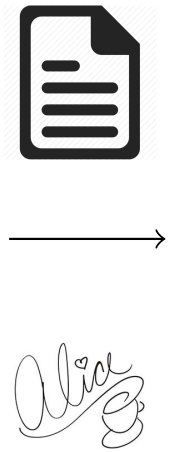
- The direct operation $q \mapsto Q$ is efficient (double and add algorithm);
- The inverse operation $Q \mapsto q$ is computationally infeasible for certain groups.

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

- q: private key (signature);
- Q: public key (verification).

Outline

Digital signature



Alice

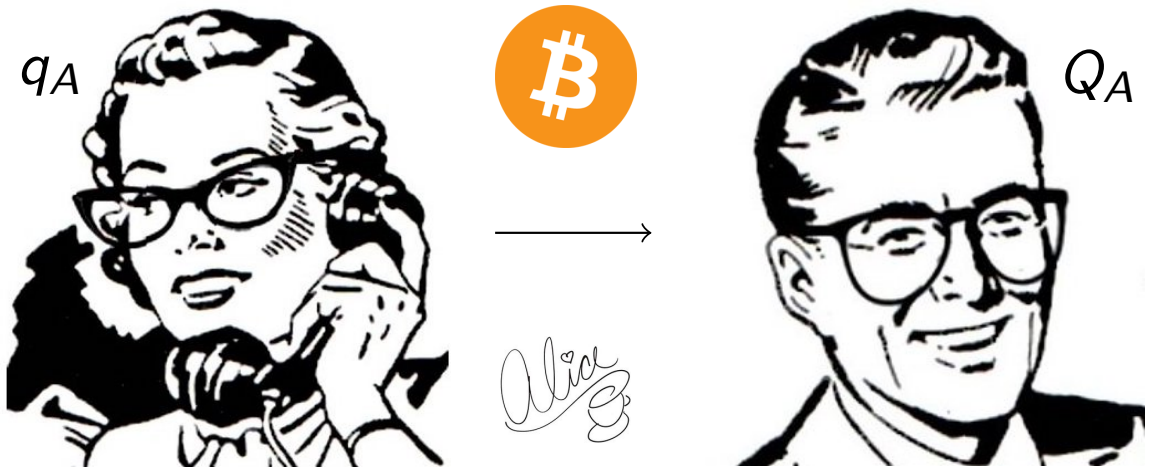
Bob

Digital signature

Alice

Bob

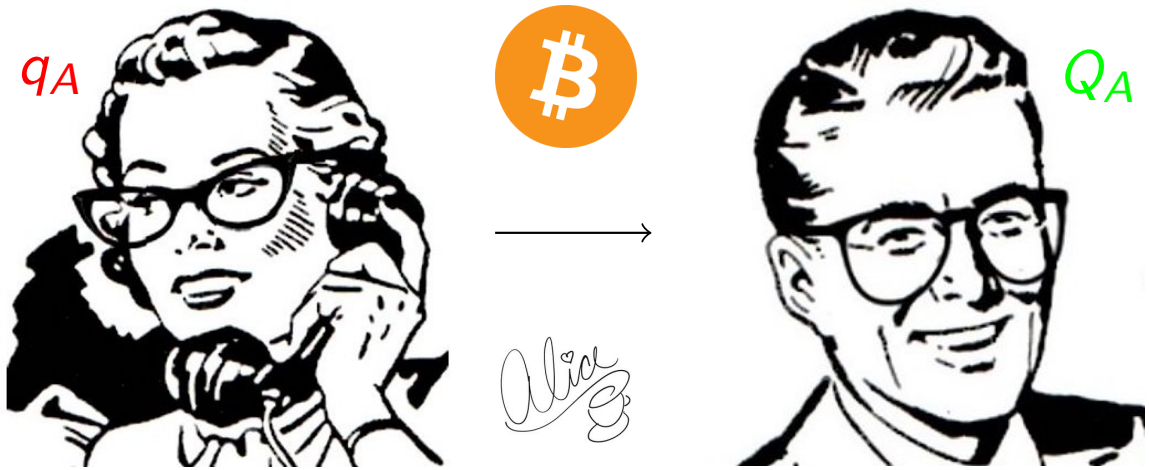
Digital signature



Alice Bob

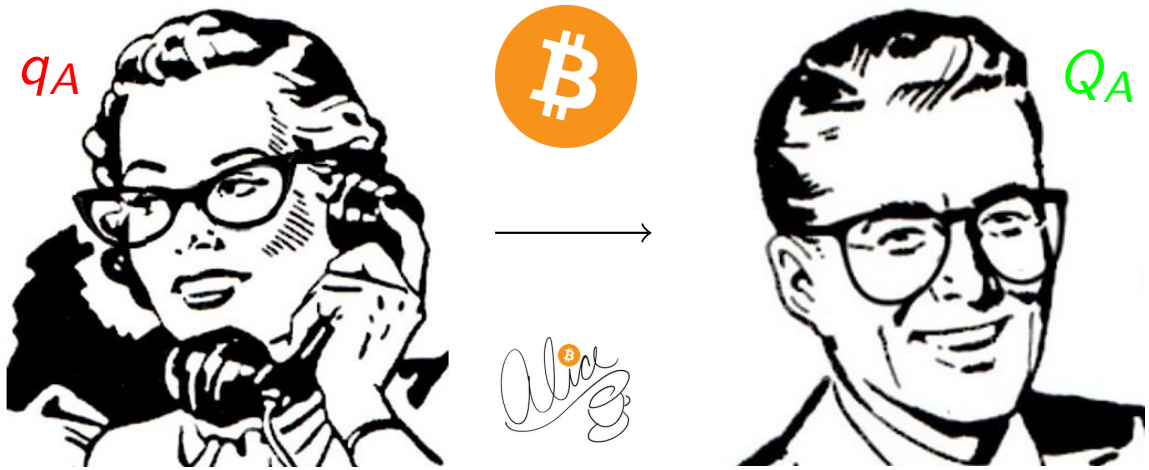
- Authentication: the recipient is confident that the message comes from the alleged sender;

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;

Digital signature

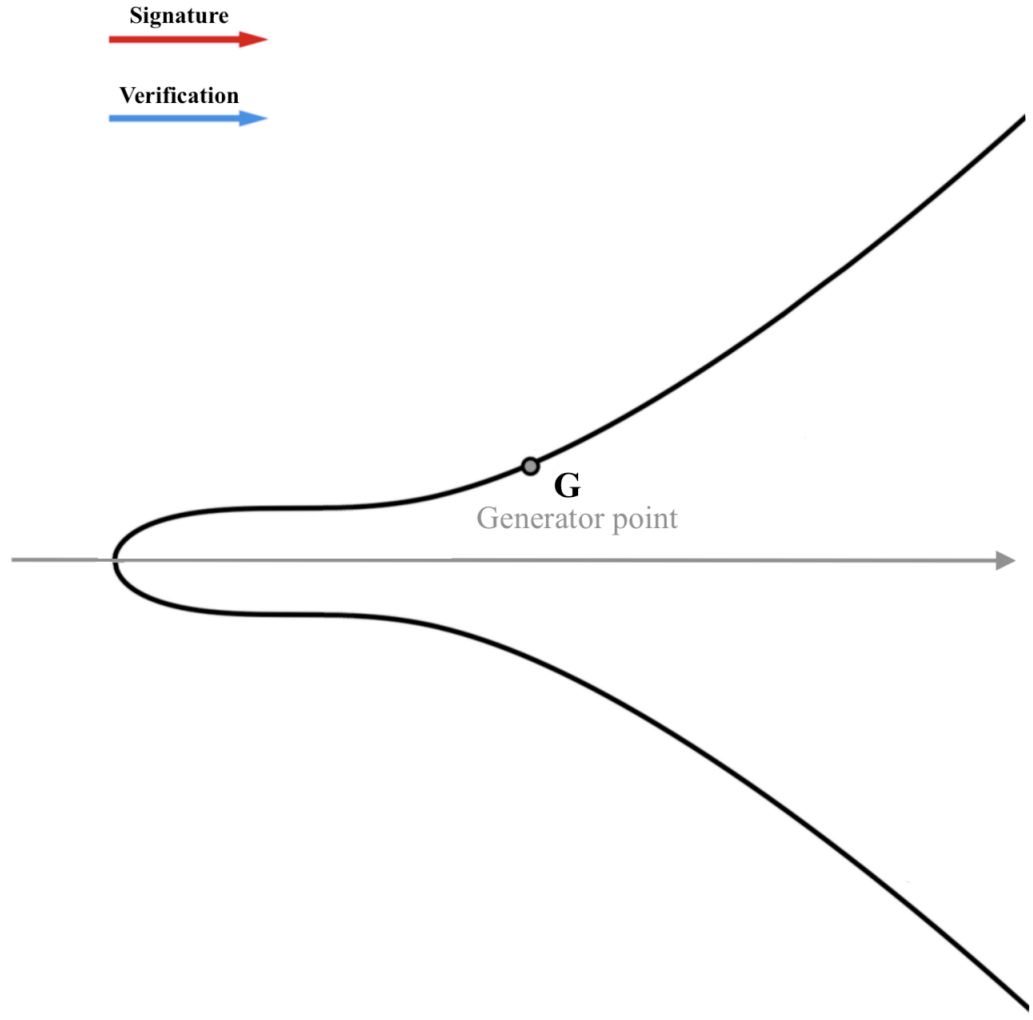


Alice Bob

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

Elliptic curve digital signature algorithm

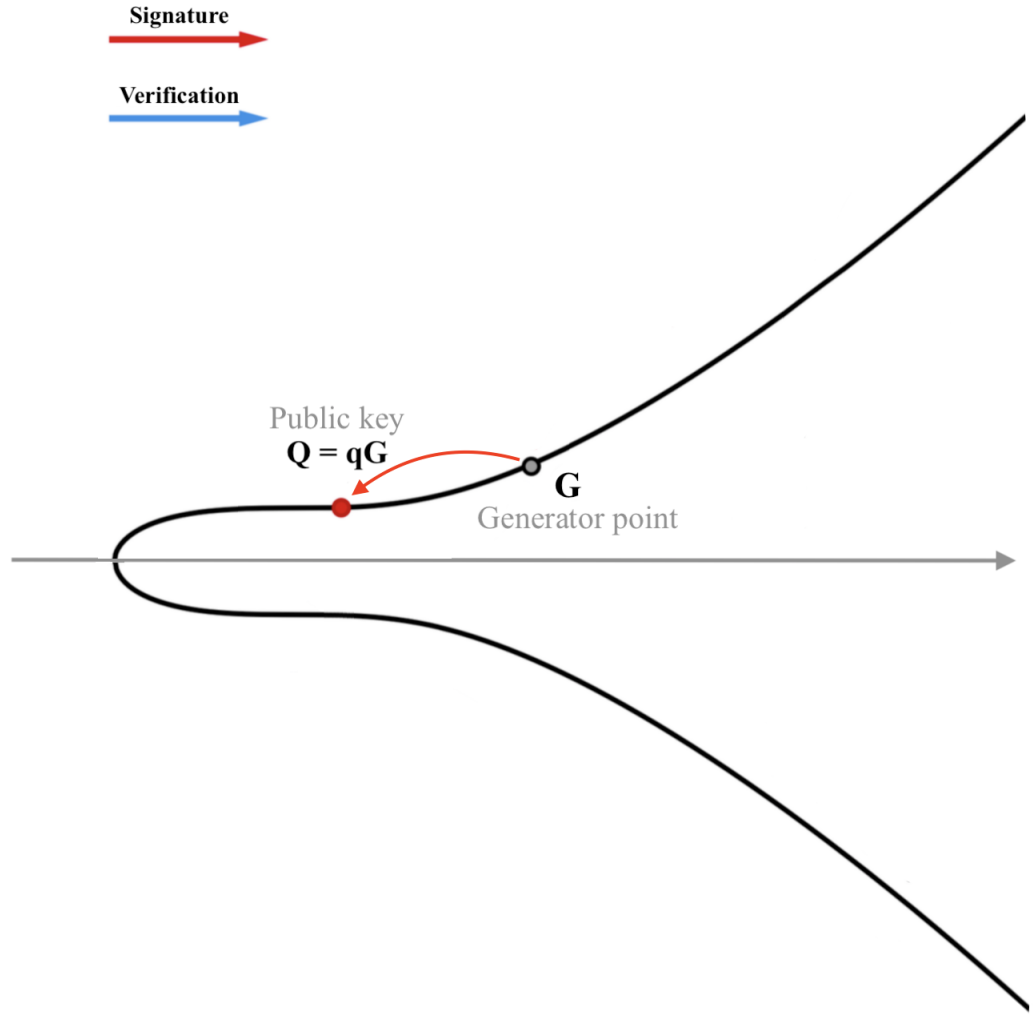
ECDSA_SIG(m, q):



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

Elliptic curve digital signature algorithm

ECDSA_SIG(m, q):

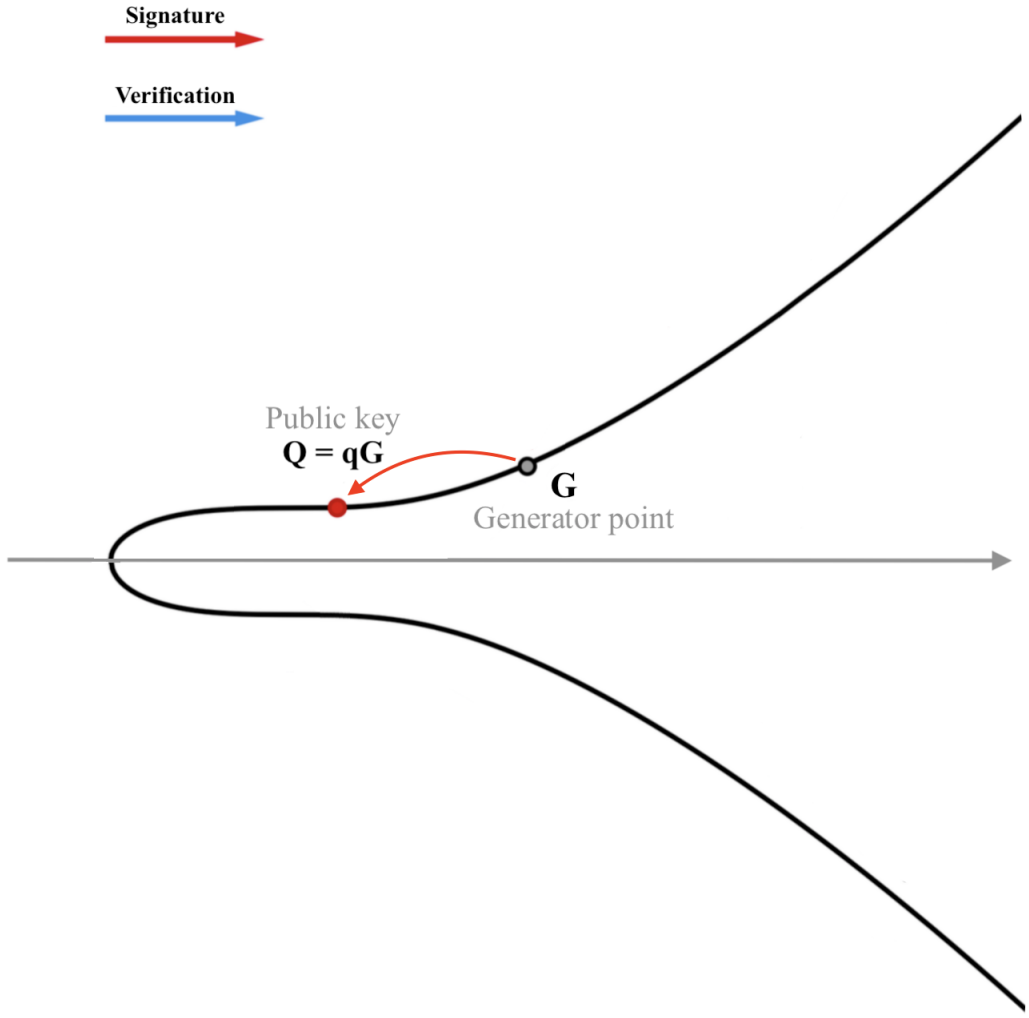


Adapted from:
`https://medium.com/cryptoadvance/how-schnorr-signatures-may-improve-bitcoin-91655bcb4744`

Elliptic curve digital signature algorithm

ECDSA_SIG(m, q):

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

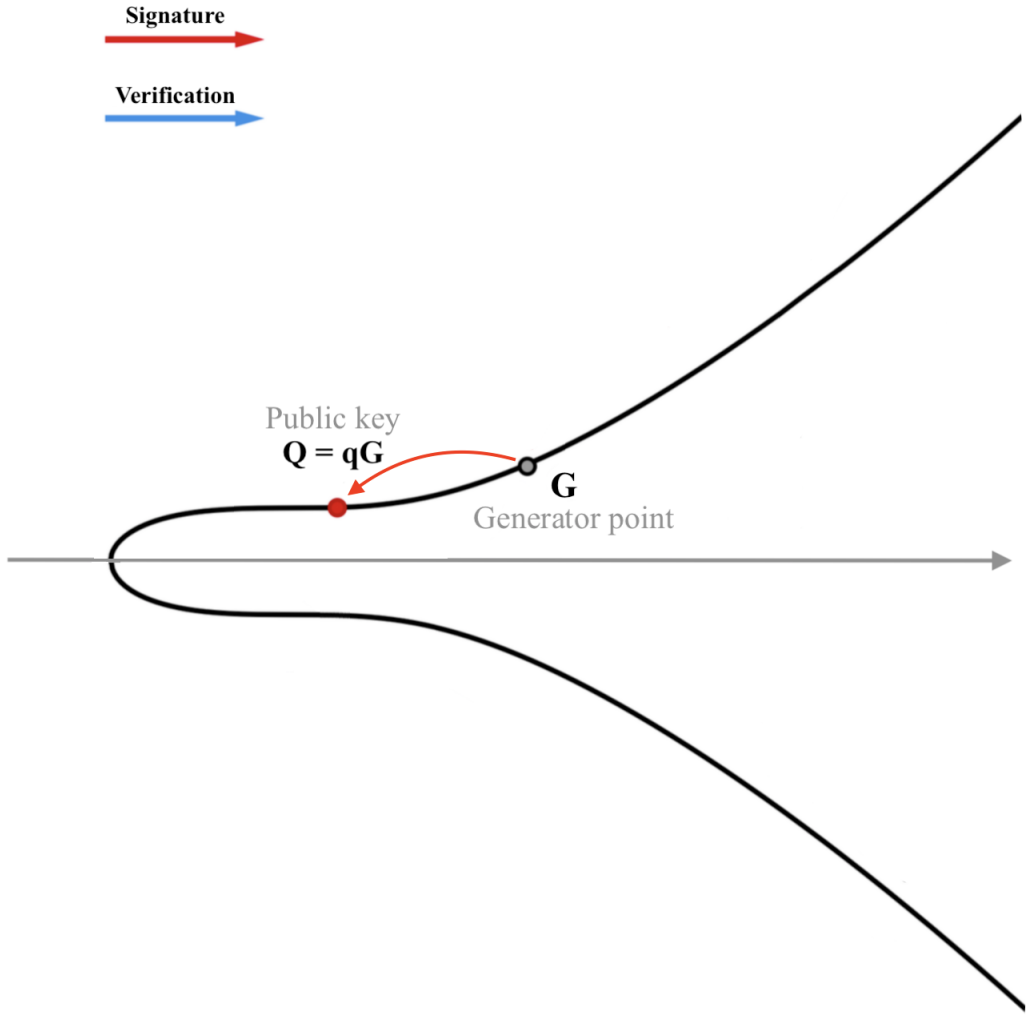


Adapted from:
`https://medium.com/cryptoadvance/how-schnorr-signatures-may-improve-bitcoin-91655bcb4744`

Elliptic curve digital signature algorithm

ECDSA_SIG(m, q):

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

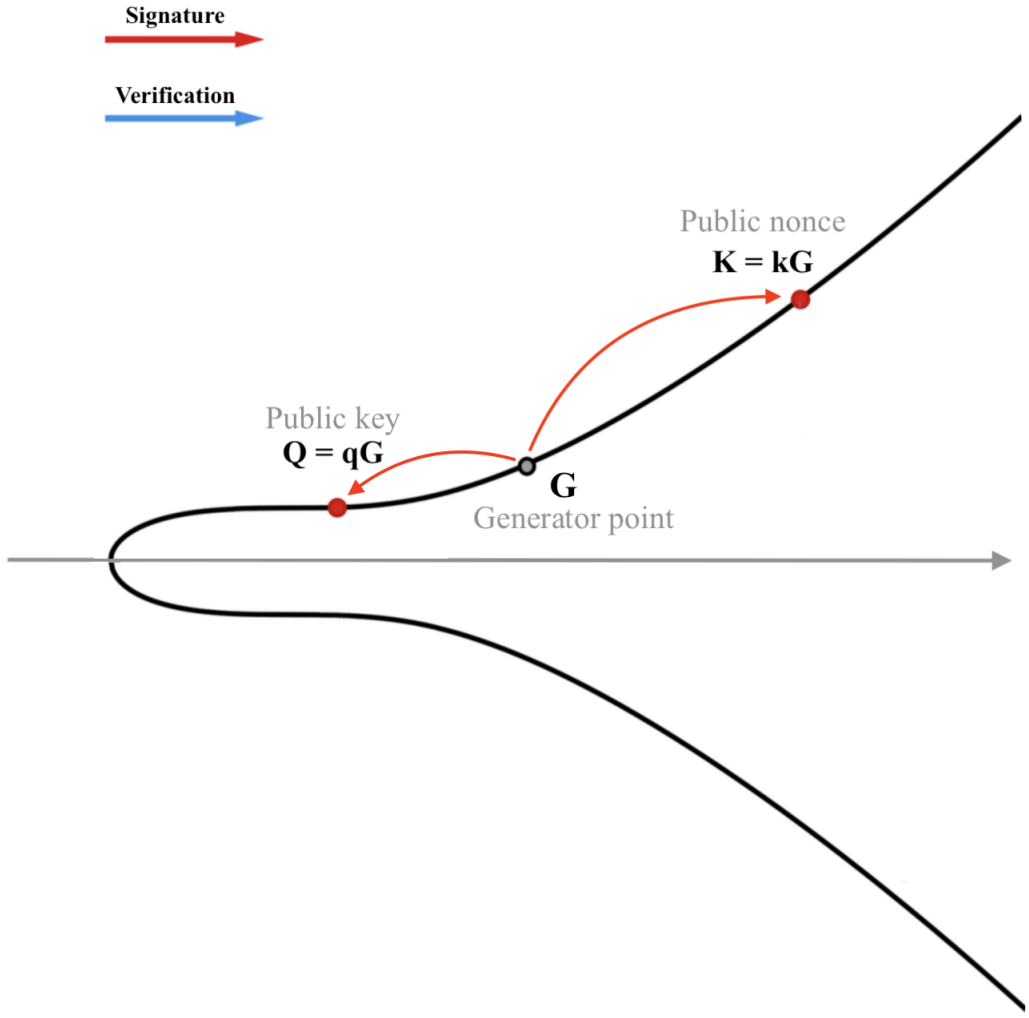


Adapted from:
`https://medium.com/cryptoadvance/how-schnorr-signatures-may-improve-bitcoin-91655bcb4744`

Elliptic curve digital signature algorithm

ECDSA_SIG(m, q):

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

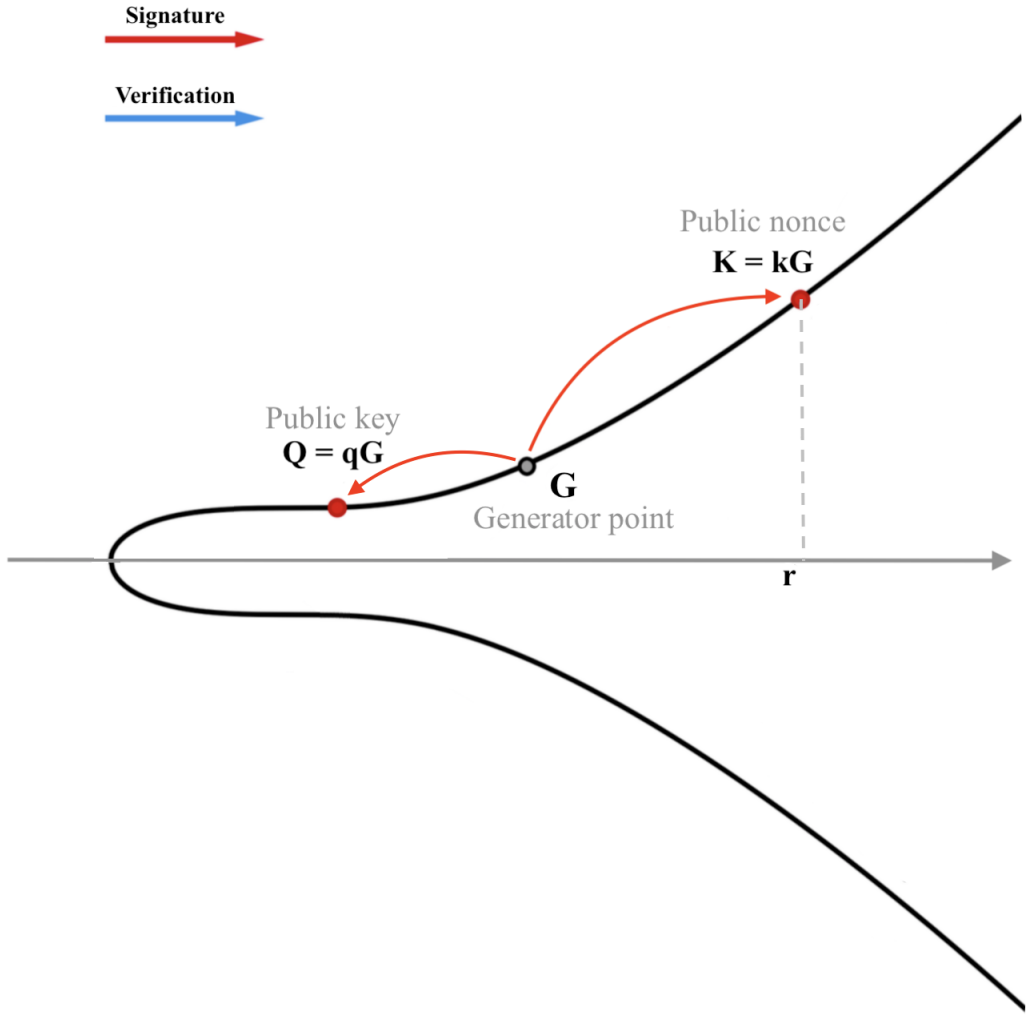


Adapted from:
`https://medium.com/cryptoadvance/how-schnorr-signatures-may-improve-bitcoin-91655bcb4744`

Elliptic curve digital signature algorithm

ECDSA_SIG(m, q):

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

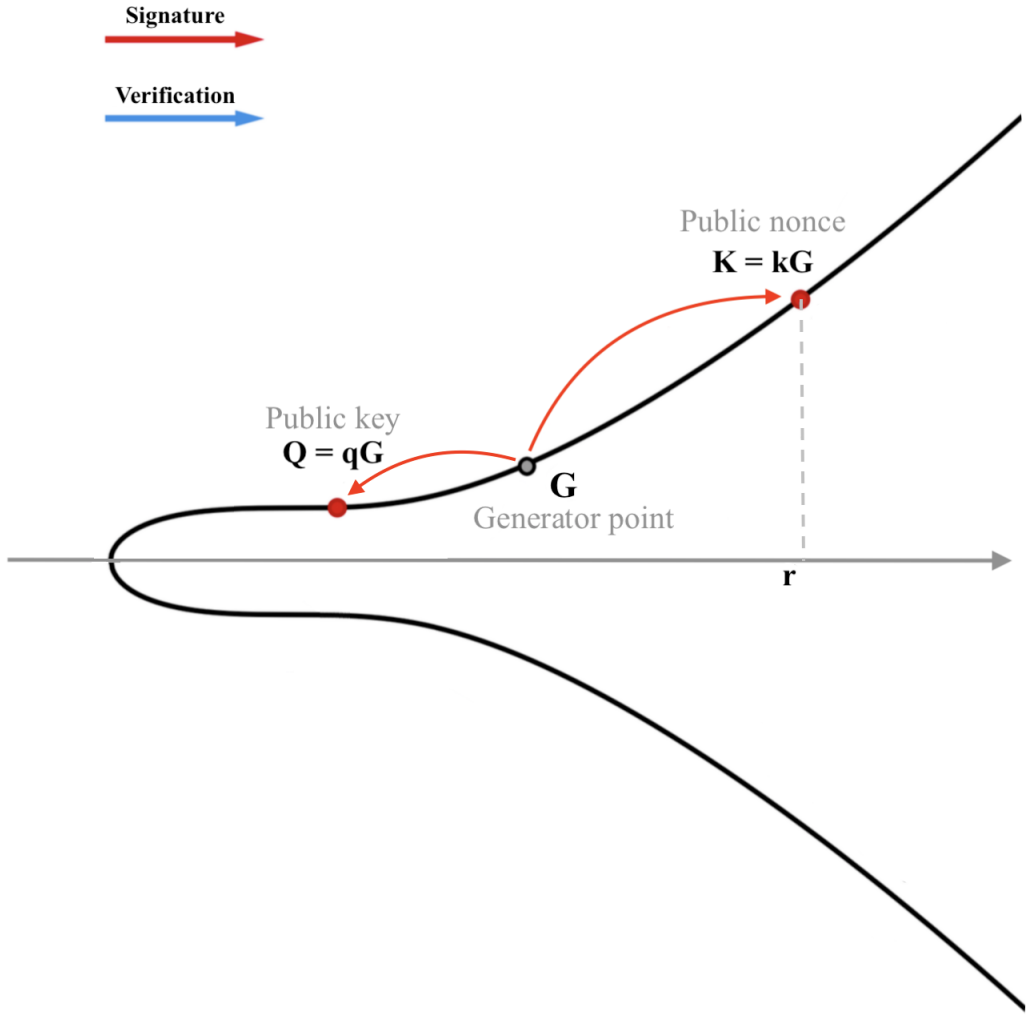


Adapted from:
`https://medium.com/cryptoadvance/how-schnorr-signatures-may-improve-bitcoin-91655bcb4744`

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. $s \leftarrow k^{-1}(z + rq) \pmod{n}$;

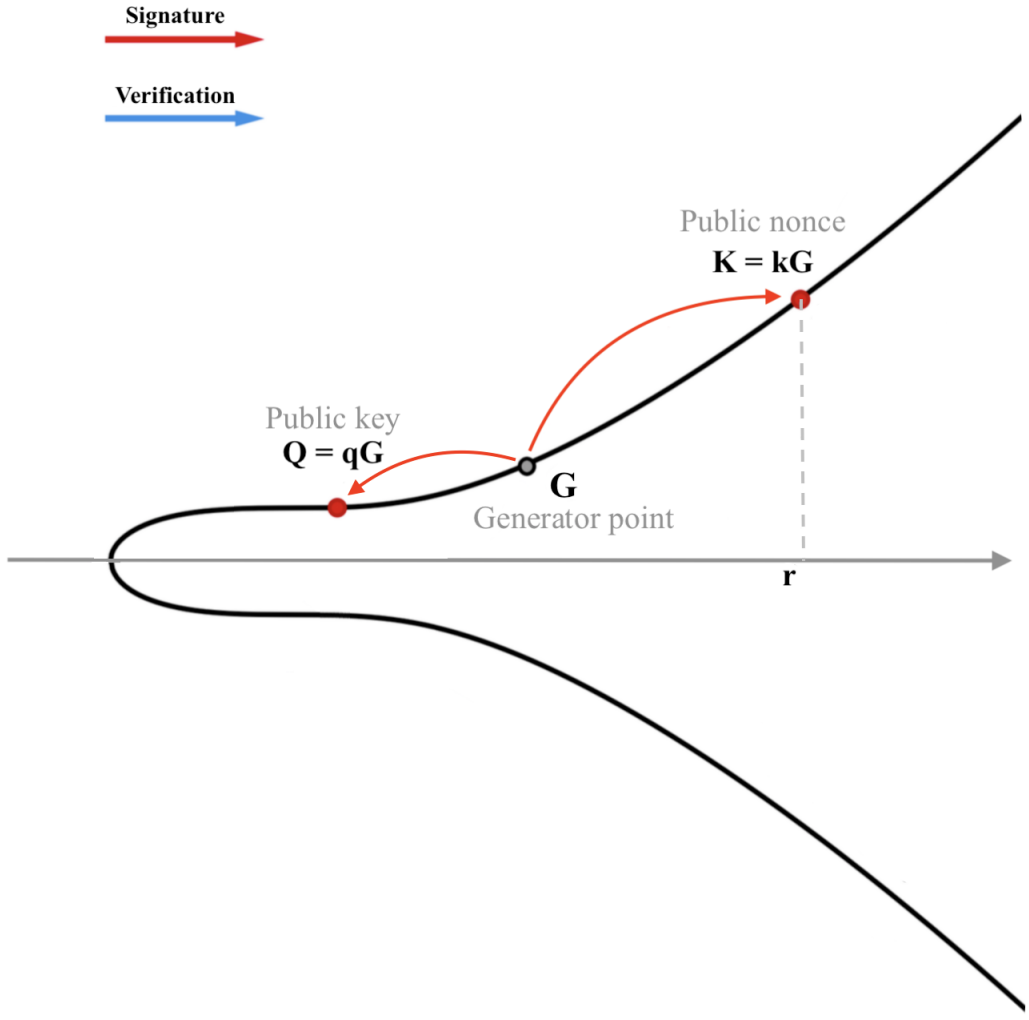


Adapted from:
`https://medium.com/cryptoadvance/how-schnorr-signatures-may-improve-bitcoin-91655bcb4744`

Elliptic curve digital signature algorithm

ECDSA_SIG(m, q):

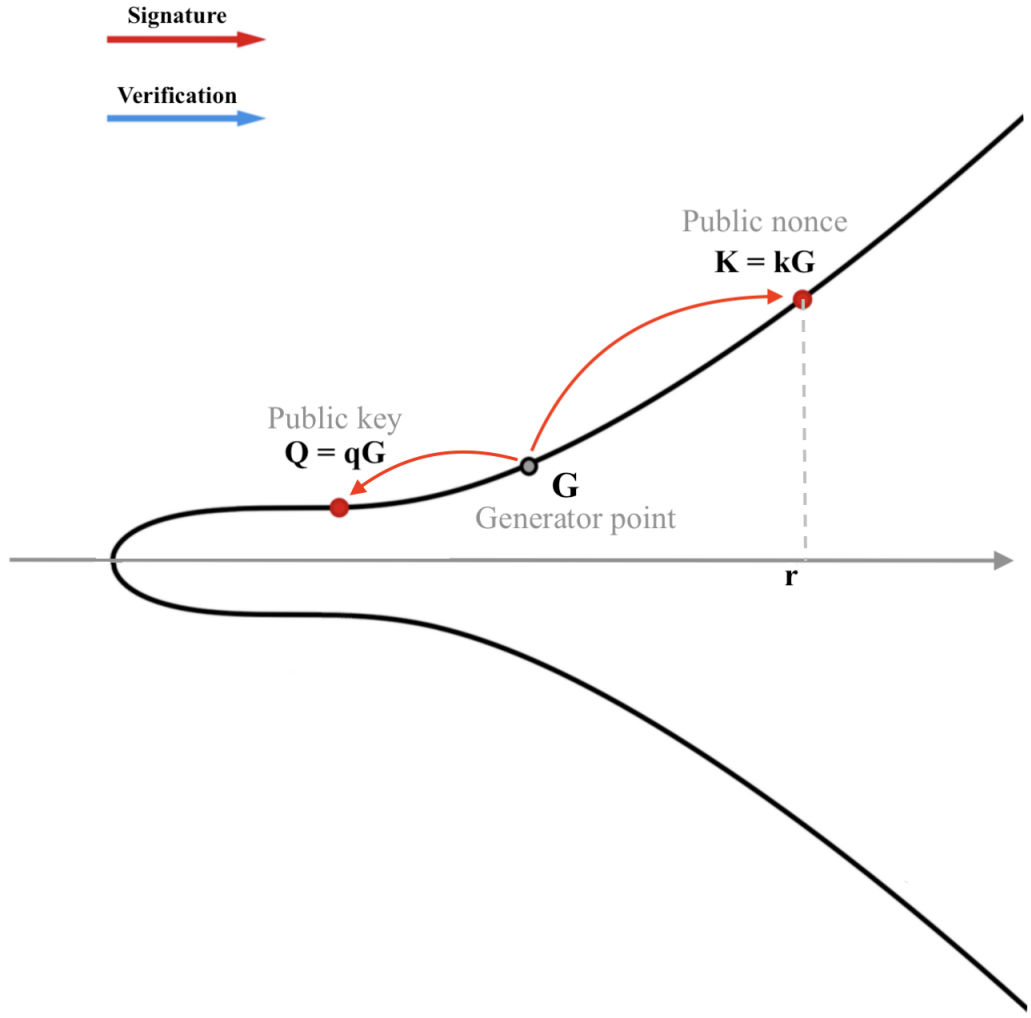
1. $z \leftarrow \text{hash}(m)$;
2. $k \xleftarrow{\$} \{1, ..., n-1\}$;
3. $K \leftarrow kG$;
4. $r \leftarrow x_K \pmod{n}$;
5. $s \leftarrow k^{-1}(z + rq) \pmod{n}$;
6. **return** (r, s).



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

Elliptic curve digital signature algorithm

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

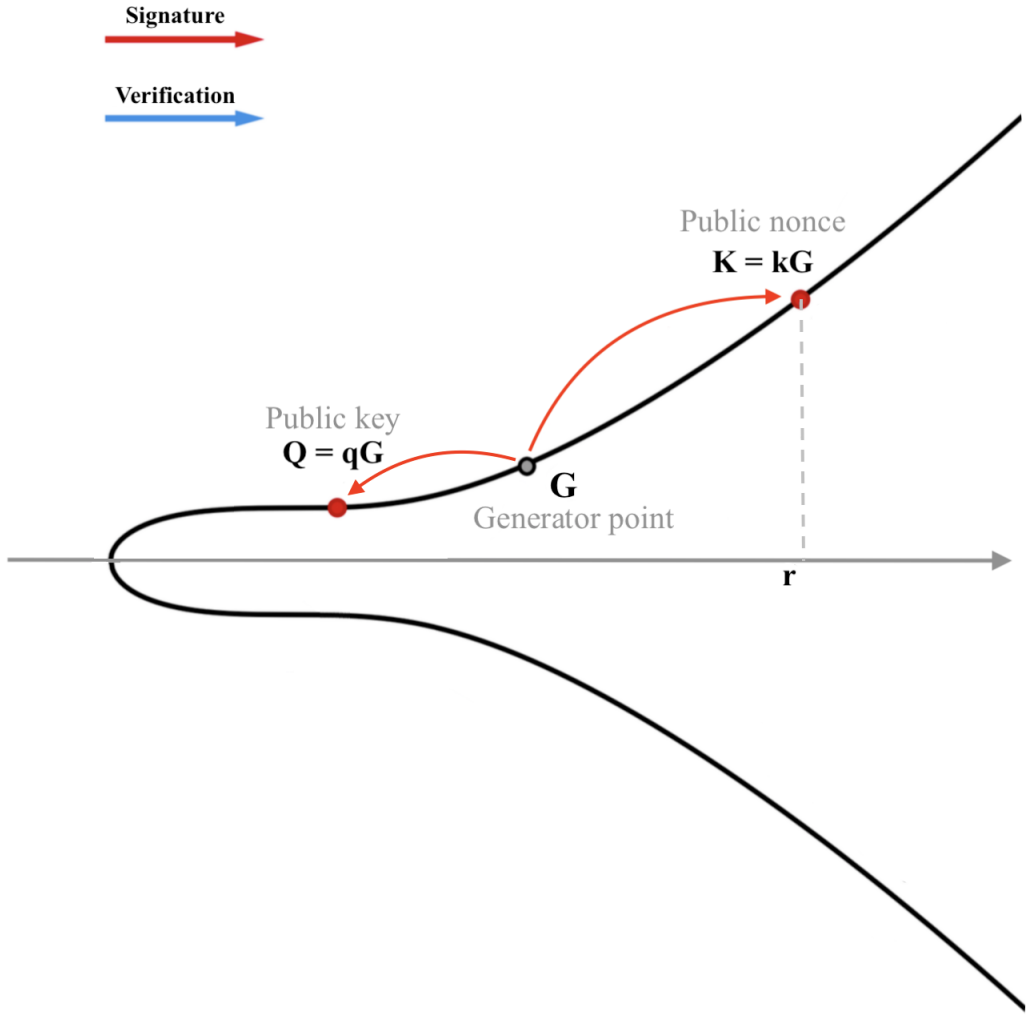


Adapted from:
`https://medium.com/cryptoadvance/how-schnorr-signatures-may-improve-bitcoin-91655bcb4744`

Elliptic curve digital signature algorithm

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

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

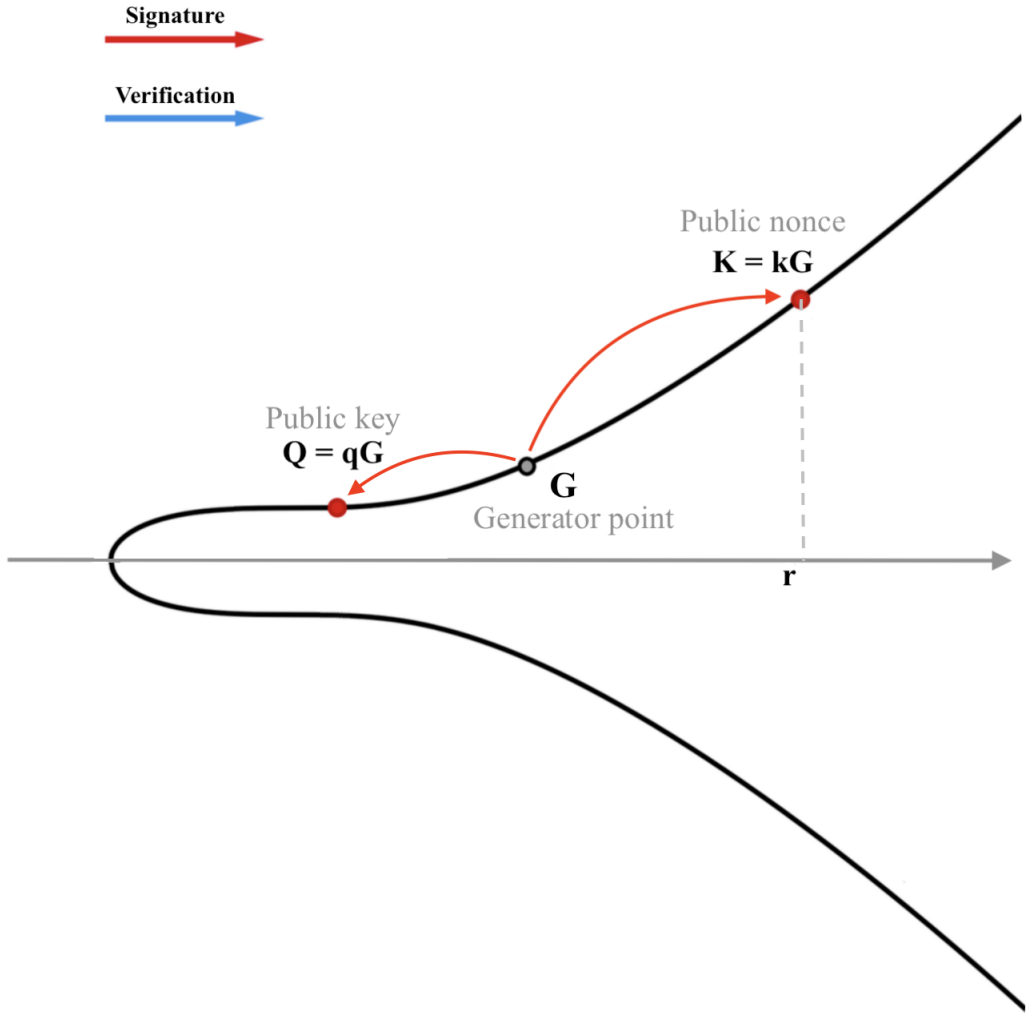


Adapted from:
`https://medium.com/cryptoadvance/how-schnorr-signatures-may-improve-bitcoin-91655bcb4744`

Elliptic curve digital signature algorithm

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

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

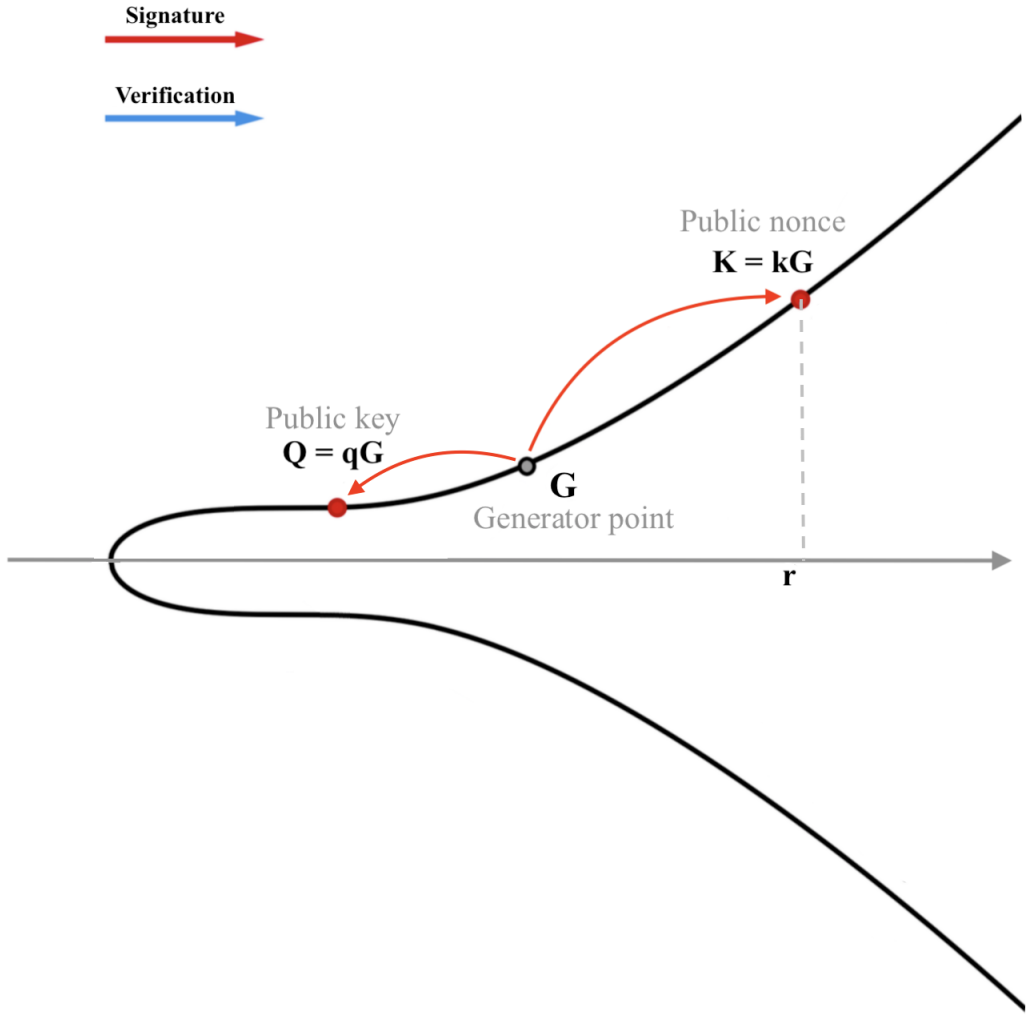


Adapted from:
`https://medium.com/cryptoadvance/how-schnorr-signatures-may-improve-bitcoin-91655bcb4744`

Elliptic curve digital signature algorithm

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

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

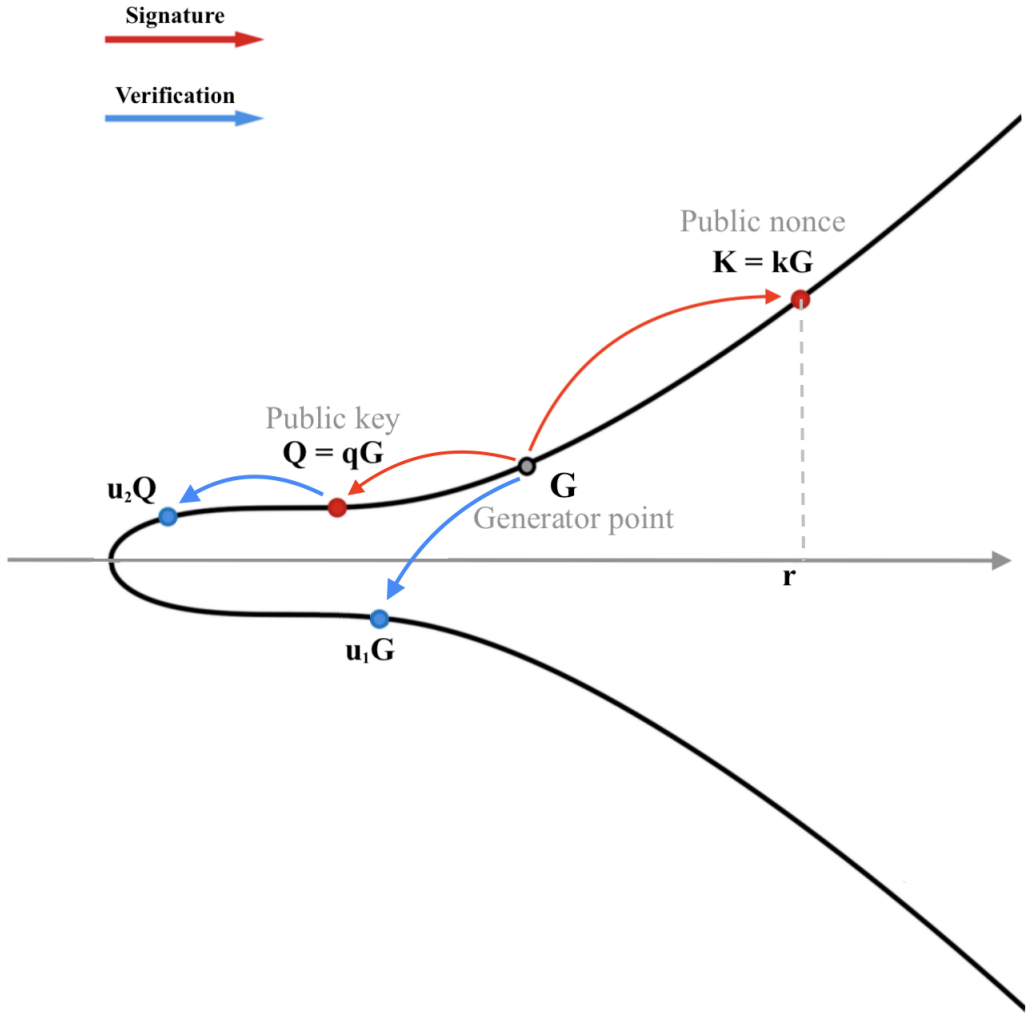


Adapted from:
`https://medium.com/cryptoadvance/how-schnorr-signatures-may-improve-bitcoin-91655bcb4744`

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. $K \leftarrow u_1 G + u_2 Q$;

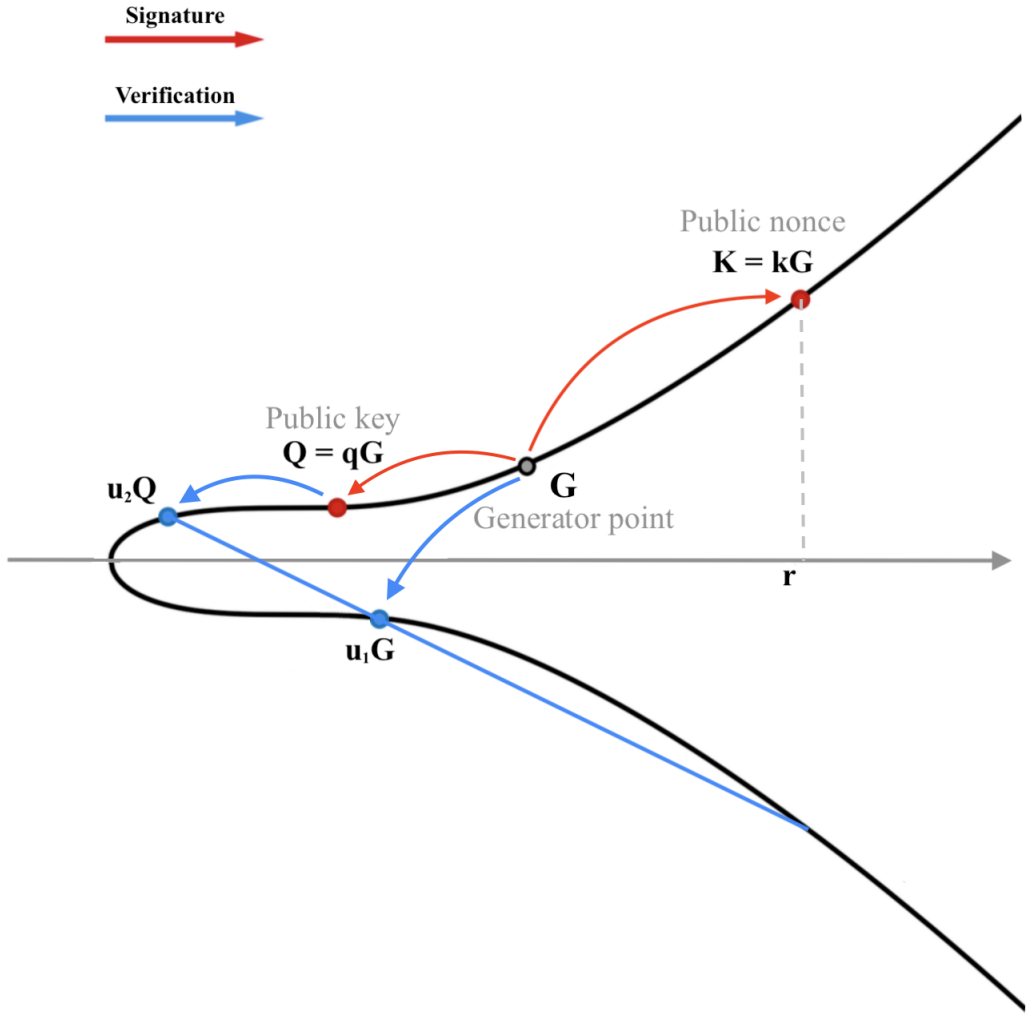


Adapted from:
`https://medium.com/cryptoadvance/how-schnorr-signatures-may-improve-bitcoin-91655bcb4744`

Elliptic curve digital signature algorithm

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

1. **If** $r \notin \{1, ..., n-1\}$ **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. $K \leftarrow u_1 G + u_2 Q$;

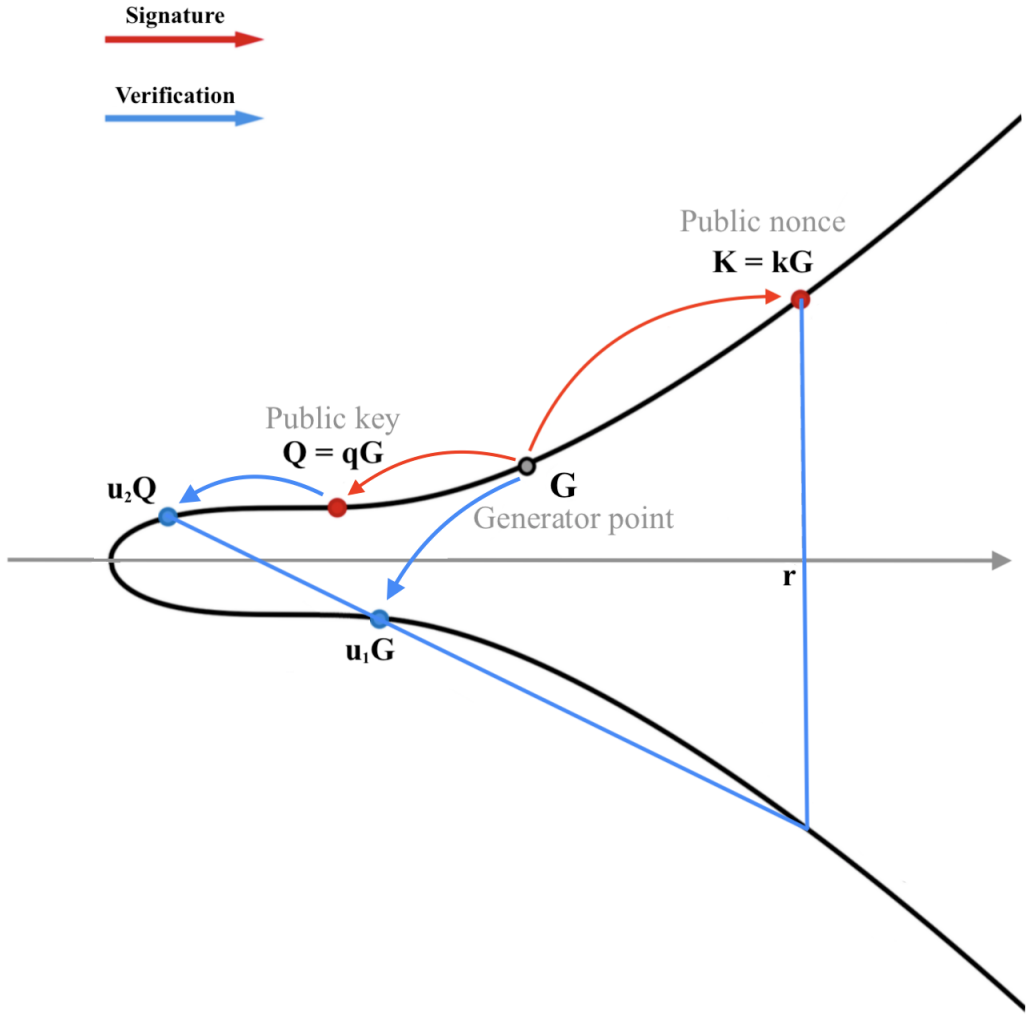


Adapted from:
`https://medium.com/cryptoadvance/how-schnorr-signatures-may-improve-bitcoin-91655bcb4744`

Elliptic curve digital signature algorithm

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

1. **If** $r \notin \{1, ..., n-1\}$ **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. $K \leftarrow u_1 G + u_2 Q$;

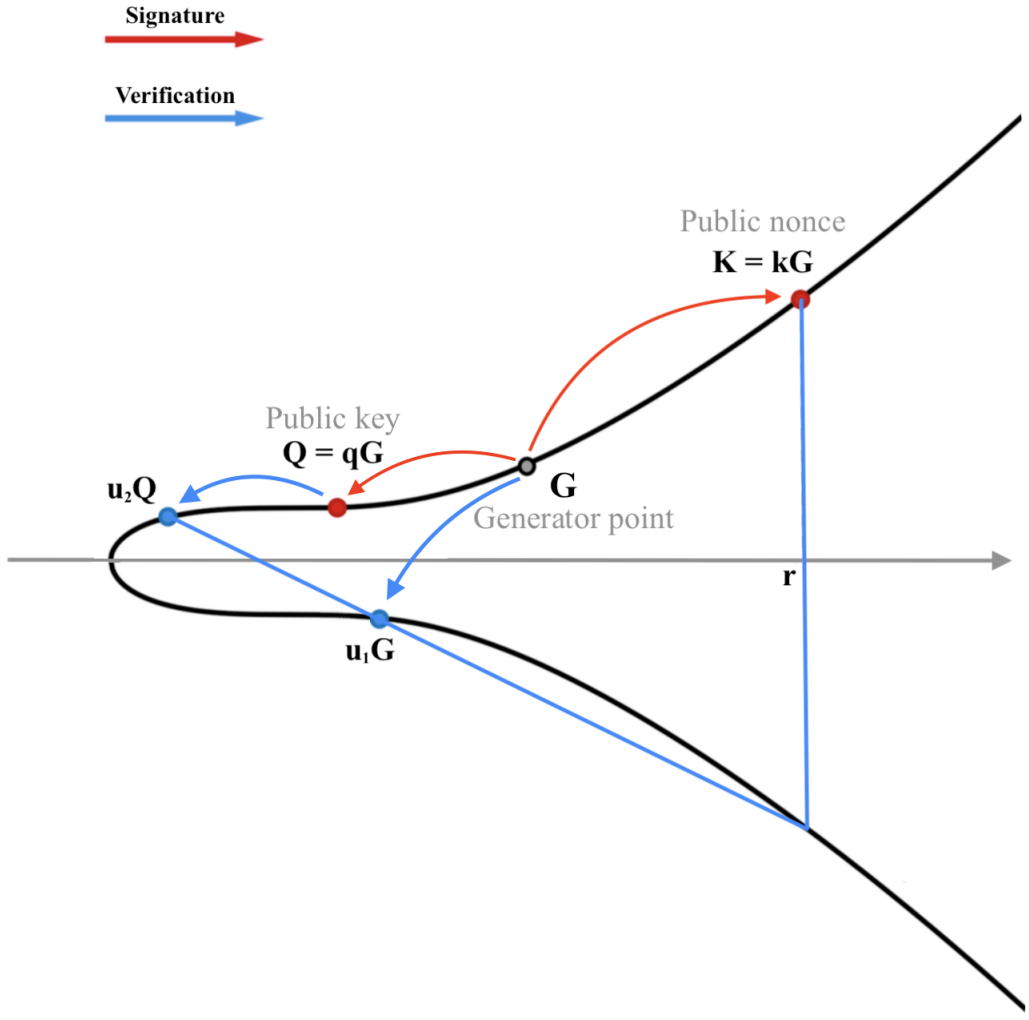


Adapted from:
`https://medium.com/cryptoadvance/how-schnorr-signatures-may-improve-bitcoin-91655bcb4744`

Elliptic curve digital signature algorithm

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

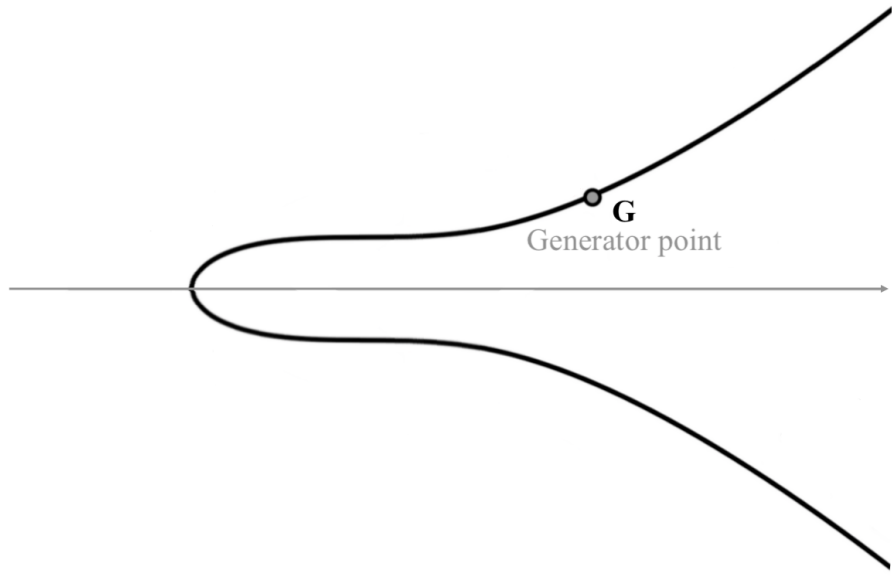
1. **If** $r \notin \{1, ..., n-1\}$ **or** $s \notin \{1, ..., n-1\}$:
   return False;
2. $z \leftarrow \text{hash}(m)$;
3. $u_1 \leftarrow zs^{-1} \pmod{n}$,
   $u_2 \leftarrow rs^{-1} \pmod{n}$;
4. $K \leftarrow u_1 G + u_2 Q$;
5. **return** $r = x_K$ **(mod** n**)**.



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

Elliptic curve Schnorr signature algorithm

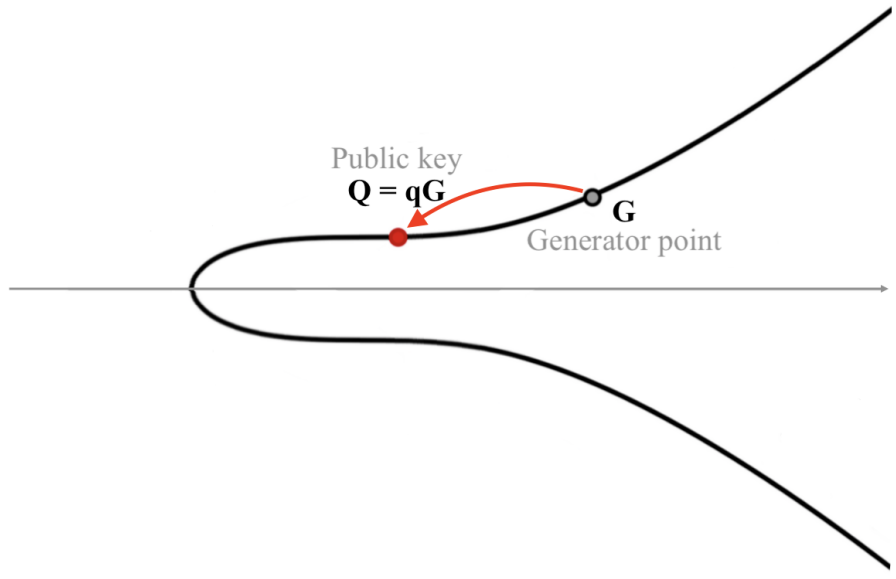
ECSSA_SIG(m, q):



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

Elliptic curve Schnorr signature algorithm

ECSSA_SIG(m, q):

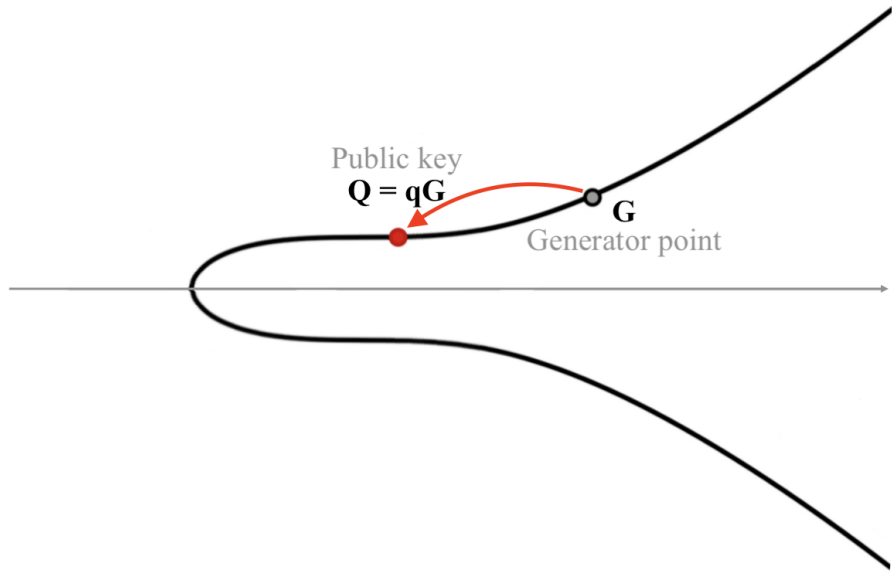


Adapted from:
`https://medium.com/cryptoadvance/how-schnorr-signatures-may-improve-bitcoin-91655bcb4744`

Elliptic curve Schnorr signature algorithm

ECSSA_SIG(m, q):

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

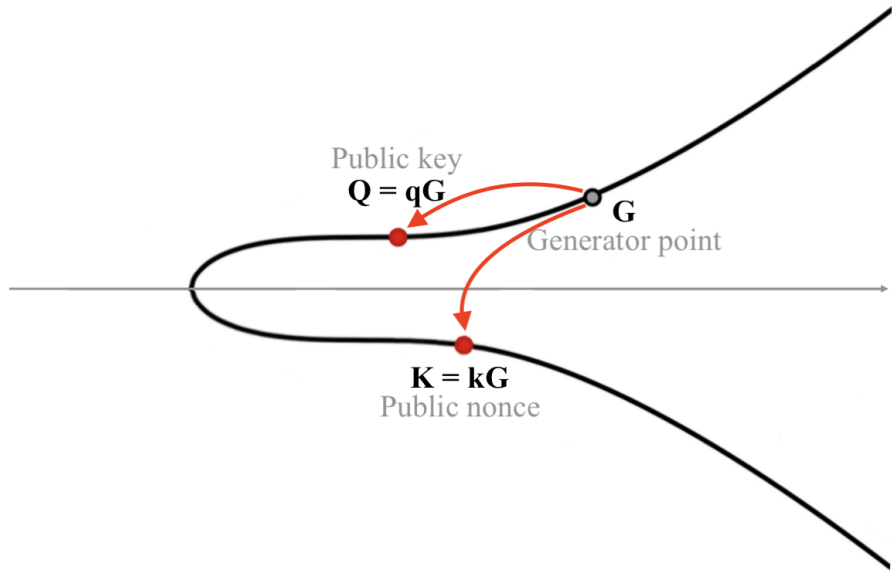


Adapted from:
`https://medium.com/cryptoadvance/how-schnorr-signatures-may-improve-bitcoin-91655bcb4744`

Elliptic curve Schnorr signature algorithm

ECSSA_SIG(m, q):

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

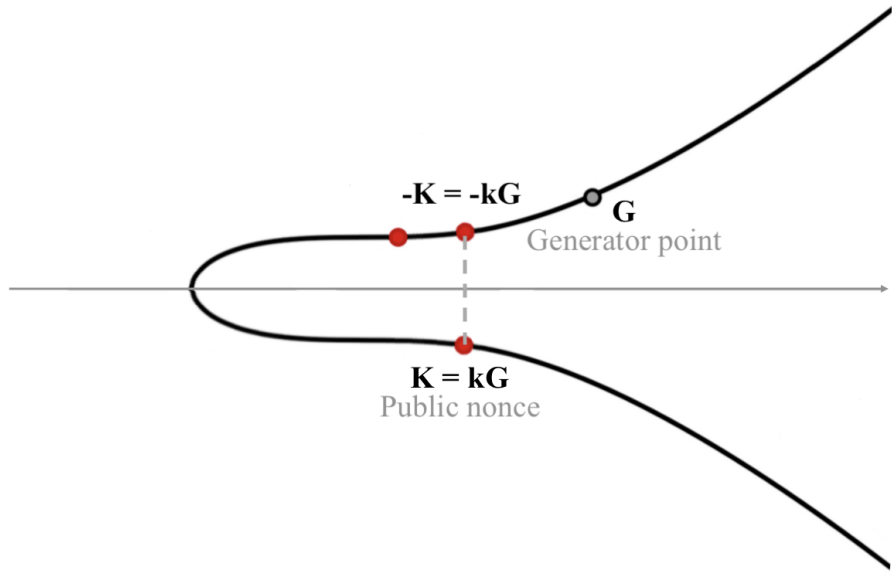


Adapted from:
`https://medium.com/cryptoadvance/how-schnorr-signatures-may-improve-bitcoin-91655bcb4744`

Elliptic curve Schnorr signature algorithm

ECSSA_SIG(m, q):

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

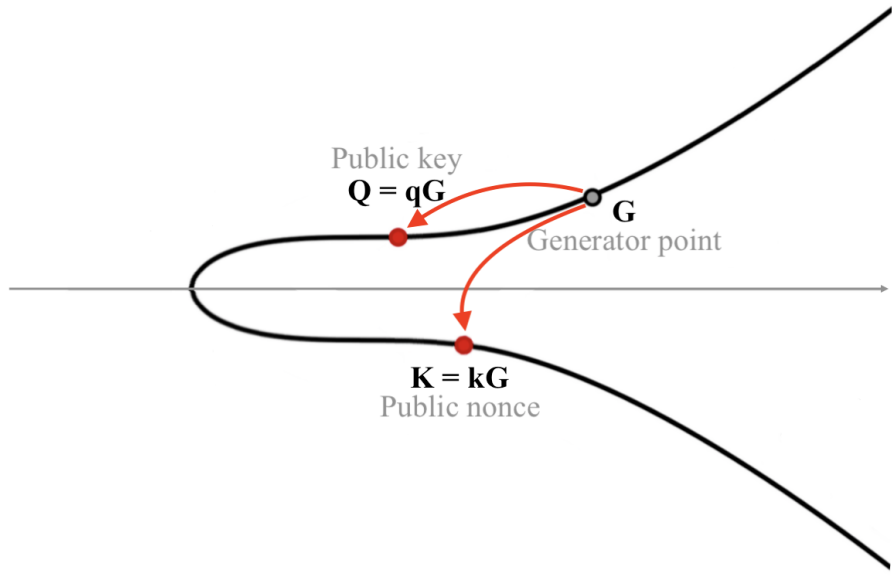


Adapted from:
`https://medium.com/cryptoadvance/how-schnorr-signatures-may-improve-bitcoin-91655bcb4744`

Elliptic curve Schnorr signature algorithm

ECSSA_SIG(m, q):

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

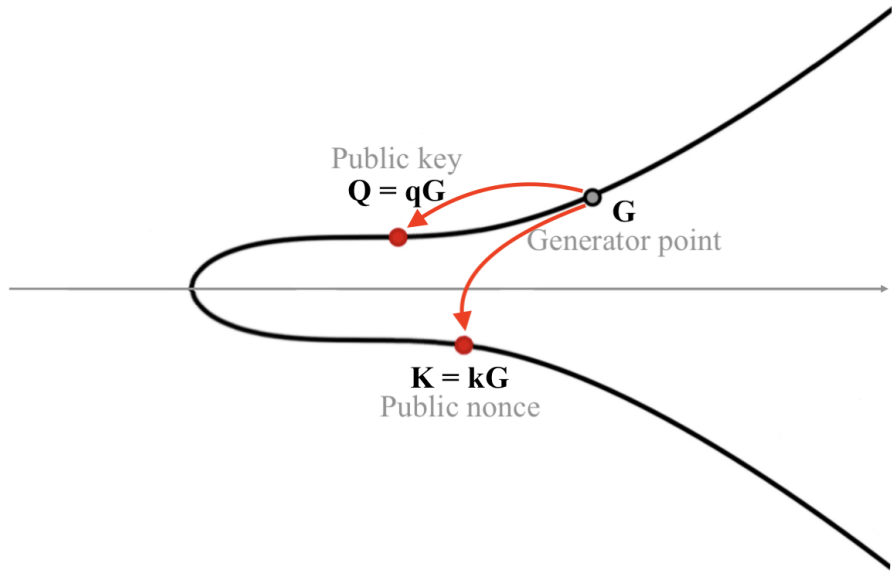


Adapted from:
`https://medium.com/cryptoadvance/how-schnorr-signatures-may-improve-bitcoin-91655bcb4744`

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. $s \leftarrow k + eq \pmod n$;

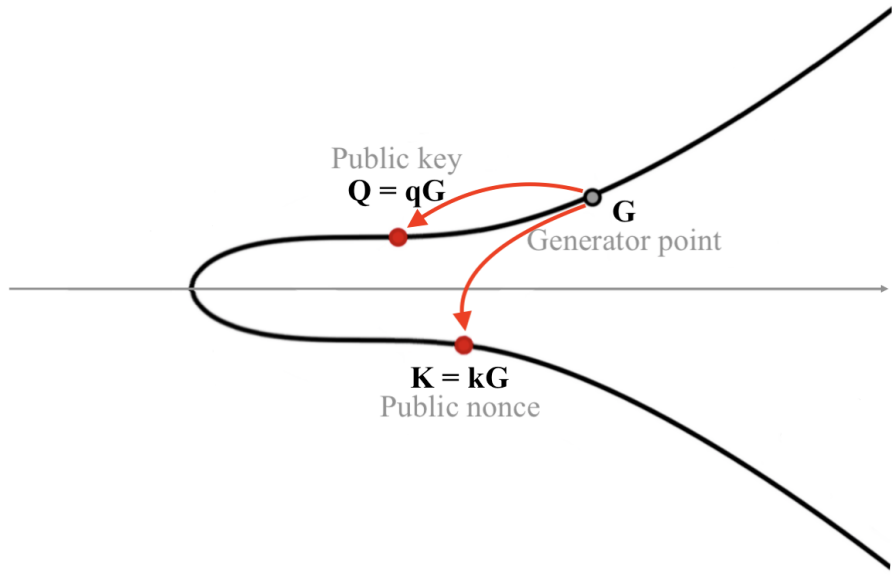


Adapted from:
https://medium.com/cryptoadvance/how-schnorr-signatures-may-improve-bitcoin-91655bcb4744

Elliptic curve Schnorr signature algorithm

ECSSA_SIG(m, q):

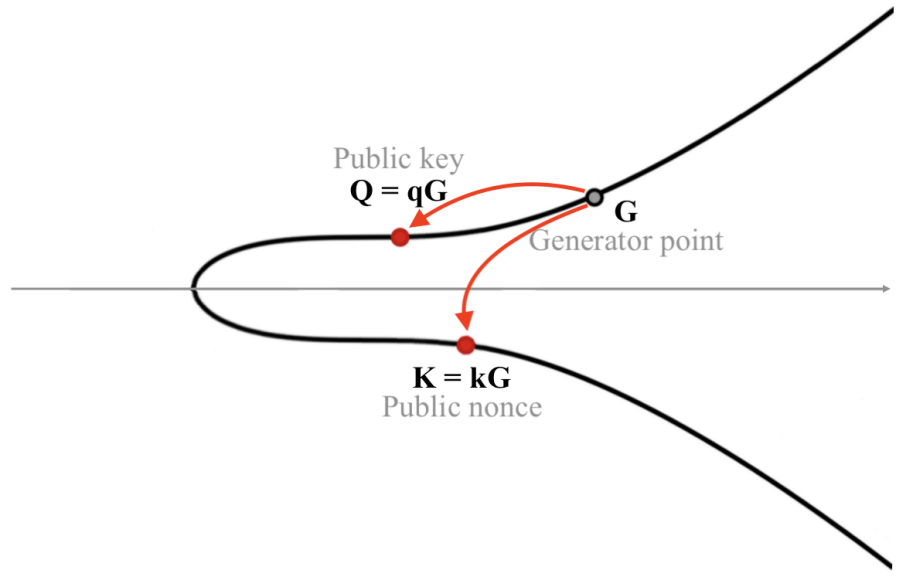
1. $k \xleftarrow{\$} \{1, ..., n-1\}$;
2. $K \leftarrow kG$;
3. **If** jacobi$(y_K) \neq 1$:
   $k \leftarrow n - k$;
4. $e \leftarrow \text{hash}(x_K||qG||m) \pmod n$;
5. $s \leftarrow k + eq \pmod n$;
6. **return** (x_K, s).



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

Elliptic curve Schnorr signature algorithm

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

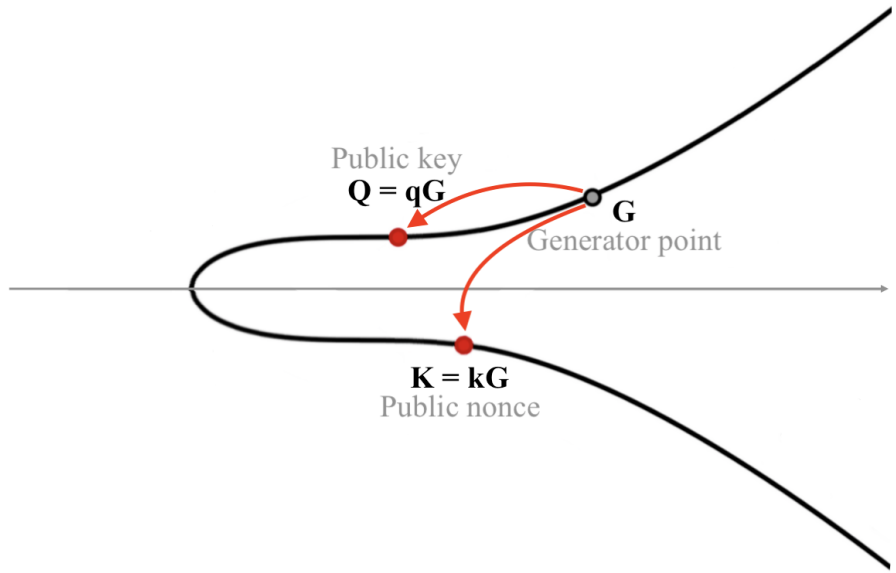


Adapted from:
`https://medium.com/cryptoadvance/how-schnorr-signatures-may-improve-bitcoin-91655bcb4744`

Elliptic curve Schnorr signature algorithm

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

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

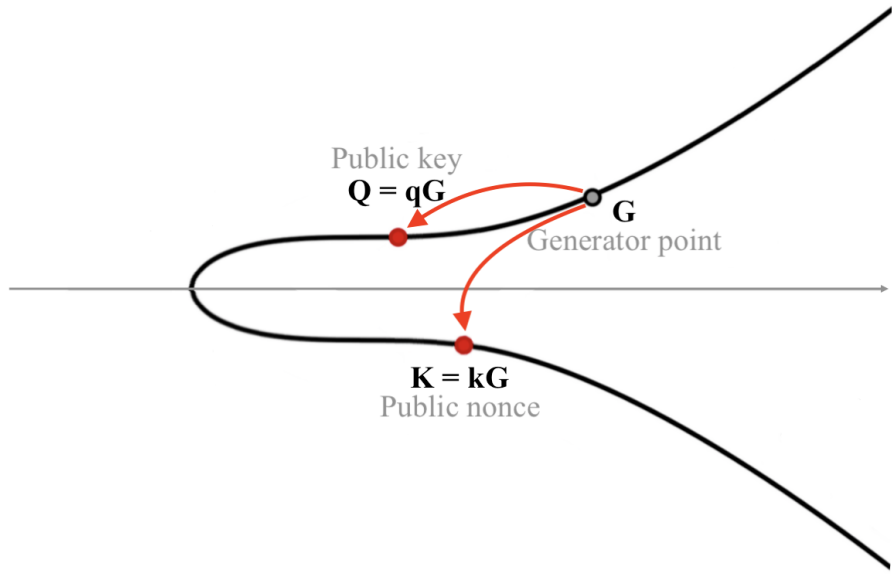


Adapted from:
`https://medium.com/cryptoadvance/how-schnorr-signatures-may-improve-bitcoin-91655bcb4744`

Elliptic curve Schnorr signature algorithm

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

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

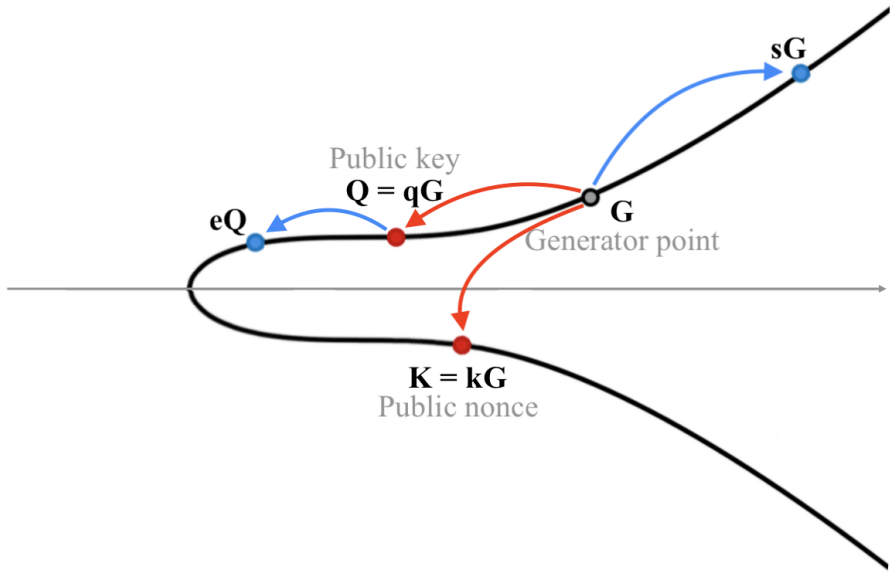


Adapted from:
`https://medium.com/cryptoadvance/how-schnorr-signatures-may-improve-bitcoin-91655bcb4744`

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. $K \leftarrow sG - eQ$;

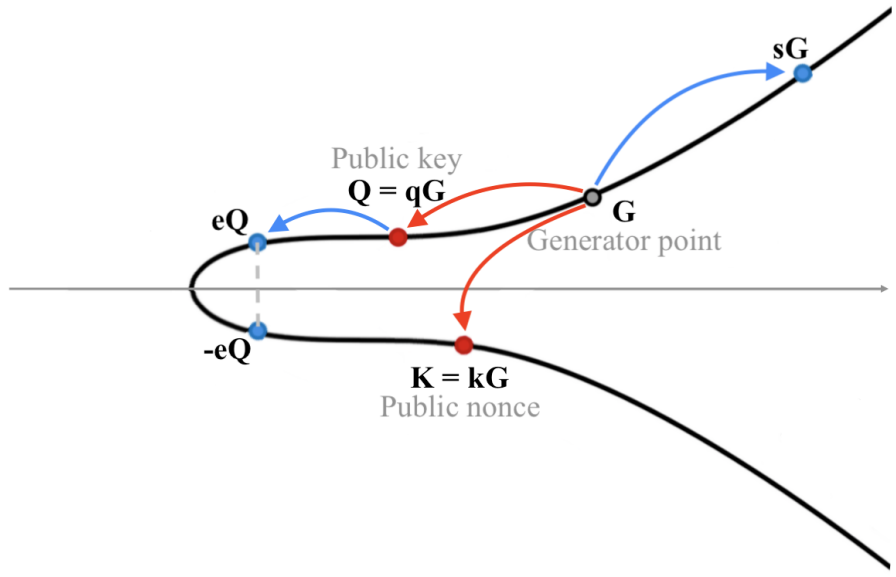


Adapted from:
`https://medium.com/cryptoadvance/how-schnorr-signatures-may-improve-bitcoin-91655bcb4744`

Elliptic curve Schnorr signature algorithm

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

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

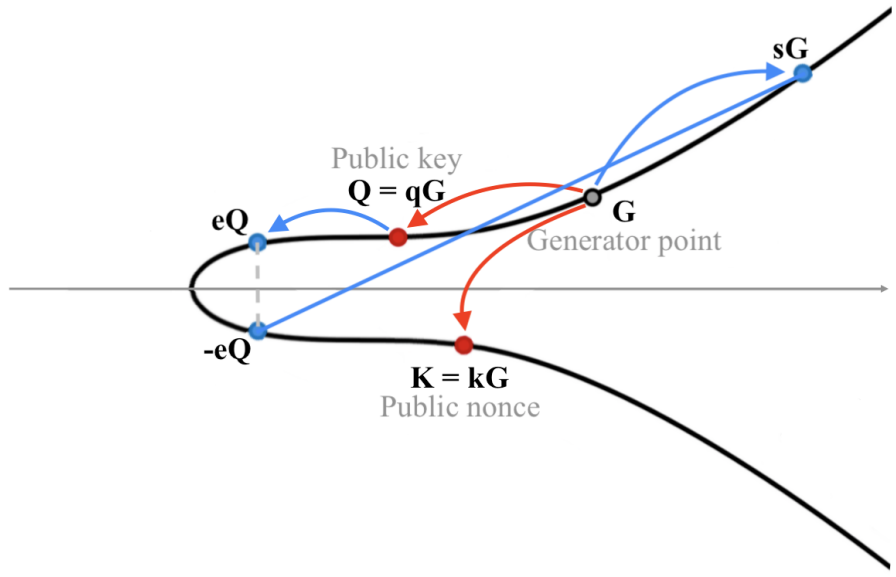


Adapted from:
`https://medium.com/cryptoadvance/how-schnorr-signatures-may-improve-bitcoin-91655bcb4744`

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. $K \leftarrow sG - eQ$;

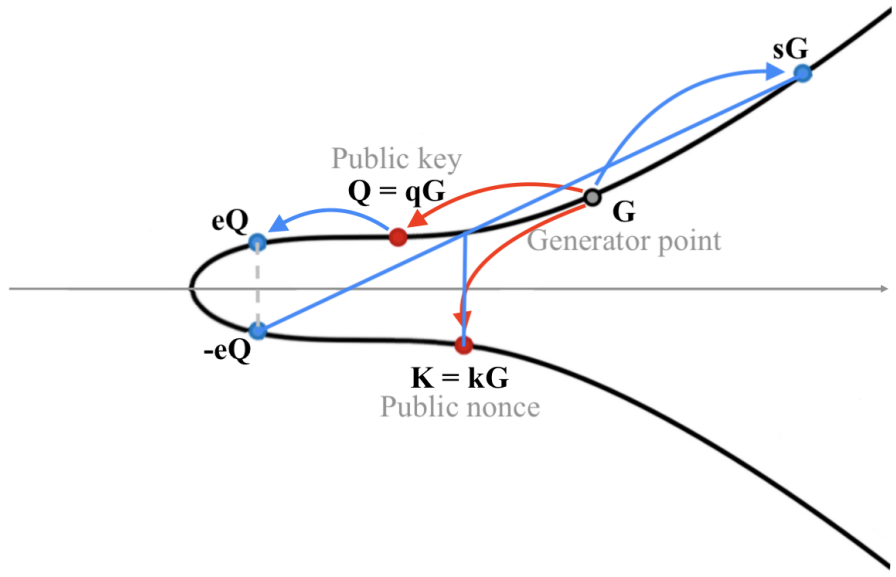


Adapted from:
`https://medium.com/cryptoadvance/how-schnorr-signatures-may-improve-bitcoin-91655bcb4744`

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. $K \leftarrow sG - eQ$;

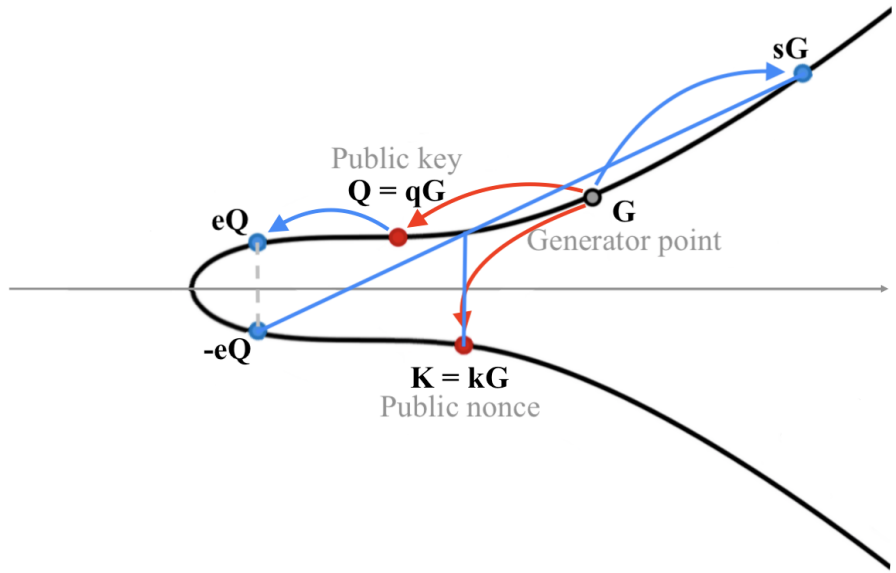


Adapted from:
`https://medium.com/cryptoadvance/how-schnorr-signatures-may-improve-bitcoin-91655bcb4744`

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;

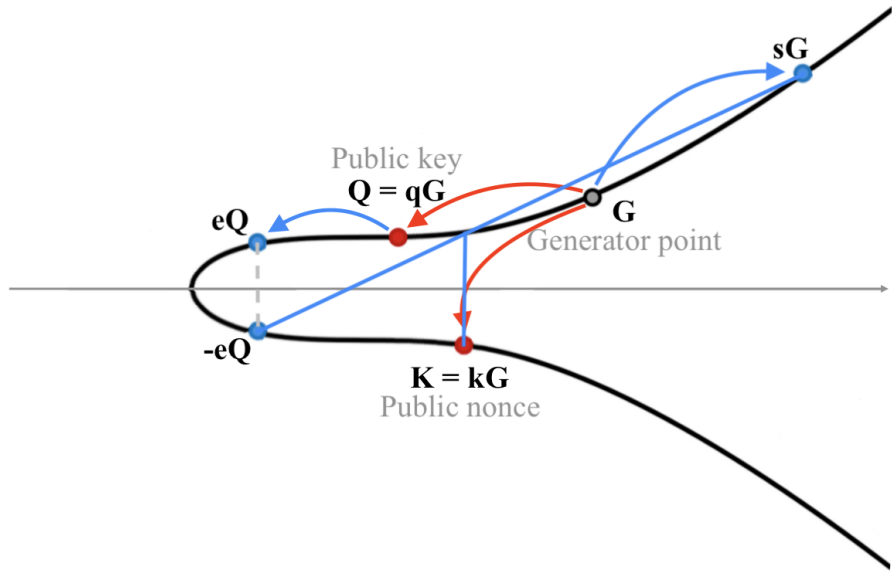


Adapted from:
`https://medium.com/cryptoadvance/how-schnorr-signatures-may-improve-bitcoin-91655bcb4744`

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 $\Longrightarrow$ 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;
- Chosen standardization forbid batch validation;

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;
- 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;
- Chosen standardization forbid batch validation;
- Requires the calculation of modular inverses;

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;
- 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;
- Chosen standardization forbid batch validation;
- 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 $\Longrightarrow$ not malleable;
- New encoding: fixed length, always 64 bytes;
- Batch validation scales logarithmically;
- No computational heavy operations involved;
- Linear: easier higher level constructions.

Outline

Bitcoin's smart contracts

Bitcoin has been conceived as programmable money: the funds are locked by a smart contract that settles the spending conditions.

Signatures are required to enforce property rights in the digital realm, so they are typically necessary to spend bitcoins and are embedded in the spending smma

An easy example: Pay-to-Public-Key (P2PK)

- Locking script: <pubKey> OP_CHECKSIG
- Unlocking script: <sig>

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:

- 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: $s_i = k_i + \text{hash}(x_K||Q||msg)q_i,\ i \in \{A, B\}$;
- The signature $(x_K, s_A + s_B)$ is valid on msg 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: $s_i = k_i + \text{hash}(x_K||Q||msg)q_i,\ i \in \{A, B\}$;
- The signature $(x_K, s_A + s_B)$ is valid on msg for public key Q.

INSECURE: rogue key attack!

MuSig: compact m-of-m signature scheme

To solve the problem without resorting to the Knowledge Of Secret Key (KOSK) setting, the idea is to introduce a "random factor" (ROM) $a_i = \text{hash}(\langle L \rangle || Q_i)$ per public key Q_i:

$$Q = \quad Q_1 + \quad Q_2 ... + \quad Q_m.$$

MuSig: compact m-of-m signature scheme

To solve the problem without resorting to the Knowledge Of Secret Key (KOSK) setting, the idea is to introduce a "random factor" (ROM) $a_i = \text{hash}(\langle L \rangle || Q_i)$ per public key Q_i:

$$Q = a_1 Q_1 + a_2 Q_2 ... + a_m Q_m.$$

MuSig: compact m-of-m signature scheme

To solve the problem without resorting to the Knowledge Of Secret Key (KOSK) setting, the idea is to introduce a "random factor" (ROM) $a_i = \text{hash}(\langle L \rangle || Q_i)$ per public key Q_i:

$$Q = a_1 Q_1 + a_2 Q_2 ... + a_m Q_m.$$

$$s_i = k_i + \text{hash}(x_K || Q || msg) a_i q_i \pmod{n}, \ i = 1, ..., m.$$

The signature $(x_K, s = \sum_{i=1}^{m} s_i \pmod{n})$ can be verified as a simple Schnorr signature against Q:

$$sG = \left(\sum_{i=1}^{m} k_i + \text{hash}(x_K || Q || msg) \sum_{i=1}^{m} a_i q_i \right) G =$$

$$= \sum_{i=1}^{m} K_i + \text{hash}(x_K || Q || msg) \sum_{i=1}^{m} a_i Q_i =$$

$$= K + \text{hash}(x_K || Q || msg) Q.$$

MuSig: compact m-of-m signature scheme

- Compact: same size as the single user case;
- Secure in the plain public key model: allows signature aggregation at transaction level;
- Interactive: affects usability, prevents signature aggregation at block level;
- Key aggregation: signature indistinguishable from the single user case.

The multi-signature policy is completely hidden: this is a huge improvement for both privacy and efficiency.

Threshold signature scheme (t-of-m)

- Pedersen verifiable secret sharing scheme: a dealer chooses secret $r \in \{1, ..., n-1\}$ and embeds it in a random polynomial $f(u) = r + f_1 u + ... + f_{t-1} u^{t-1} \pmod{n}$.
  Each participant $i = 1, ..., m$ of the scheme receives the share $s_i = f(i)$. To reconstruct the secret t participants can rely on Lagrange's interpolation formula;
- Protocol for the generation of a shared random secret: every participant acts as the dealer in the previous protocol.
  - Shared secret: $r = \sum_{i=1}^{m} r_i \pmod{n}$;
  - Share of the secret belonging to i: $s_i = \sum_{j=1}^{m} f_j(i) \pmod{n}$.

  To the shared secret r we can associate the random polynomial $F(u) = \sum_i f_i(u)$ and the public key $R = rG$.

Threshold signature scheme (t-of-m)

The protocol for the generation of a shared random secret is run twice to establish a key pair $\{q, Q\}$ (shares $\alpha_i = F_1(i)$) and a nonce pair $\{k, K\}$ (shares $\beta_i = F_2(i)$).

Each signer computes his partial signature as $\gamma_i = \beta_i + e\alpha_i \pmod{n}$, $e = \mathrm{hash}(x_K||Q||msg)$.

The signature is (x_K, s), with $s = \sum_j \gamma_j \omega_j$, where $\omega_j = \prod_{h \neq j} \frac{h}{h-j} \pmod{n}$. s computed in this way satisfies $s = k + eq \pmod{n}$:

$$F_3(u) = F_2(u) + eF_1(u) \implies s := F_2(0) + eF_1(0) = k + eq \pmod{n}.$$

$$F_3(i) = F_2(i) + eF_1(i) = \beta_i + e\alpha_i \pmod{n}.$$

ECDSA vs. ECSSA (multi-signature)

ECDSA:

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

$\Longrightarrow$ 33 bytes * m + 70 bytes * t.

ECSSA:

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

$\Longrightarrow$ 33 bytes + 64 bytes.

Una media sugli ultimi due anni: 1800 tx. per blocco, 500 bytes per transazione.

Adaptor signatures

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

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

$$(x_T, x_{K+T}, \tilde{s}),\ \tilde{s} = k + \text{hash}(x_{K+T}||Q||m).$$

$$\text{Checking equation: } \tilde{s}G = K + \text{hash}(x_{K+T}||Q||m)Q.$$

Adaptor signatures can be used in conjunction with the MuSig protocol: a signer generates
$s'_i = k_i + \text{hash}(x_K||Q||msg)a_i q_i + t \pmod{n}$ after having used $K_i + T$ as public nonce.

Cross-chain atomic swaps

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

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

- Locking script:

  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 address>
  OP_ENDIF
  OP_EQUALVERIFY OP_CHECKSIG

- Unlocking script:
  - <Bob sig> <Bob pubkey> <preimage> 1
  - <Alice sig> <Alice pubkey> 0

Cross-chain atomic swaps via adaptor signature

- Alice and Bob agree on a pair of transactions which are secured by the pairs of keys (Q_1^A, Q_1^B) and (Q_2^A, Q_2^B), aggregated through the MuSig protocol;
- They engage the MuSig protocol to spend from the two transactions, but Bob generates in parallel adaptor signatures for both transactions: Alice has to verify them, in particular she needs to ensure that he used the same t value;
- This results in two invalid signatures: but Bob, knowing t, can build the a corresponding valid signature;
- When he finally takes Alice's coins, he publish this valid signature on chain: Alice, that has to monitor the blockchain, learns it. But since she has the corresponding adaptor signature she can extract t and take Bob's coins.

Cross-chain atomic swaps via adaptor signature

Efficiency and privacy gains:

- We were able to condensate the verbose script semantics of the HTLC in a signature;
- The policy is indistinguishable from the single user setting (adaptor signatures are deniable: for every signature on the blockchain one can come up with some t and construct an adaptor signature);
- It is impossible to link the two transactions.

Outline

Conclusion

We have seen that Schnorr would result in huge privacy (fungibility) and efficiency (scalability) improvements for Bitcoin:

- Batch validation: a group of signatures could be validated much faster;
- Smaller key size, aggregation at transaction level, reduction of scripts to signatures: smaller blockchain and hidden policies;
- Improved security, being Schnorr signature provably secure;

Conclusion

We have seen that Schnorr would result in huge privacy (fungibility) and efficiency (scalability) improvements for Bitcoin:

- Batch validation: a group of signatures could be validated much faster;
- Smaller key size, aggregation at transaction level, reduction of scripts to signatures: smaller blockchain and hidden policies;
- Improved security, being Schnorr signature provably secure;

But there is even more:

- Adaptor signatures could be applied also to layer 2 protocols (e.g. Lightning Network) with huge enhancements in privacy;
- Taproot: built on top of the concepts of Merkelized Abstract Syntax Tree (MAST) and Pay-To-Contract its aim is to make any arbitrary script, no matter how complex it is, to look the same as a single signer transaction in the cooperative case.

Thank you for your attention!

References I

[1] Gregory Maxwell, Andrew Poelstra, Yannick Seurin, and Pieter Wuille. Simple schnorr multi-signatures with applications to bitcoin. https://eprint.iacr.org/2018/068.pdf, 2018.

[2] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. https://bitcoin.org/bitcoin.pdf, 2008.

[3] Torben Pryds Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. https://link.springer.com/content/pdf/10.1007/3-540-46766-1_9.pdf, 1992.

[4] D. R. Stinson and R. Strobl. Provably secure distributed schnorr signatures and a (t, n) threshold scheme for implicit certificates. cacr.uwaterloo.ca/techreports/2001/corr2001-13.ps, 2001.

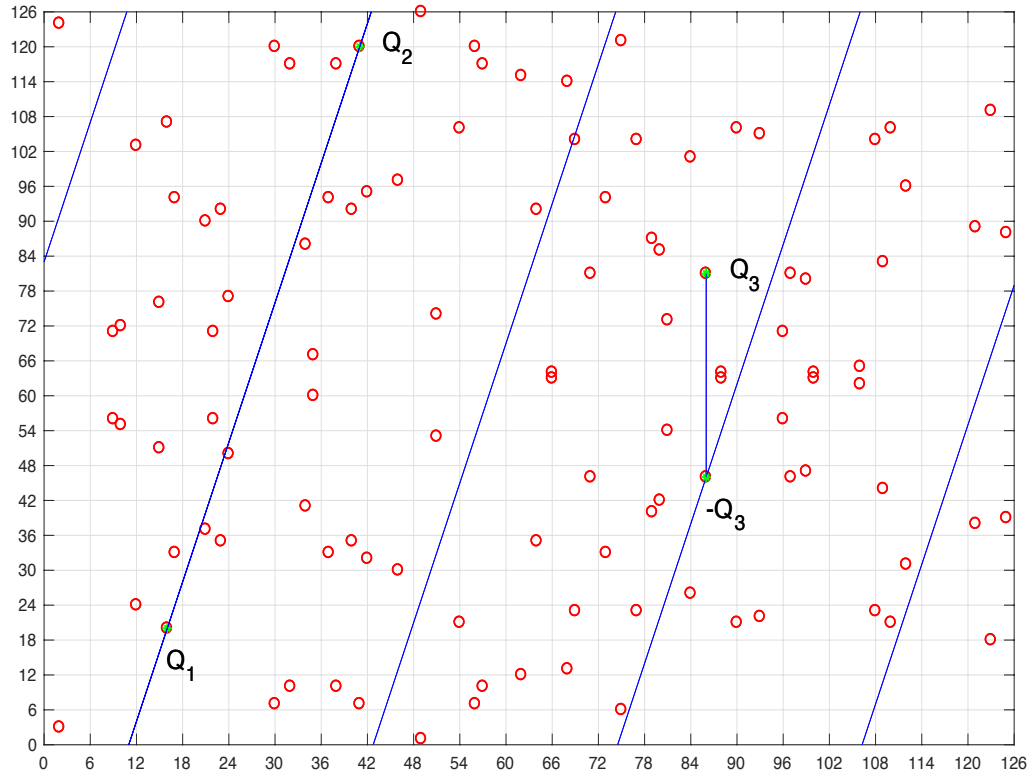
References II

[5] Lawrence C. Washington. Elliptic curves: Number theory and cryptography. Chapman and Hall/CRC; 2nd edition, 2008.

[6] Pieter Wuille. Schnorr's bip. https://github.com/sipa/bips/blob/bip-schnorr/bip-schnorr.mediawiki.

Point addition - Geometric interpretation

The intuition to add points belonging to an EC defined over a finite field is the same presented for the curve defined over the real numbers: draw the line passing through the points (that repeats along the plane in this case) until it intersects a third point: then reflect this point with respect to the line $y = \frac{p}{2}$, i.e. apply the transformation $(x_3, y_3) \rightarrow (x_3, p - y_3)$.



The curve $y^2 = x^3 - x + 3$ over $\mathbb{F}_{127}$.

Point addition - Algebraic formulas

There are some cases to be considered:

- $Q_2 = -Q_1$: by definition we have $Q_1 + Q_2 = \infty$, where ∞ is the identity element of the addition operation;
- $Q_2 = \infty$: $Q_1 + Q_2 = Q_1$;
- $Q_2 = Q_1$: $x_3 = m^2 - 2x_1$ and $y_3 = m(x_1 - x_3) - y_1$, where $m = \frac{3x_1^2 + a}{2y_1}$;
- $Q_2 \neq \pm Q_1$: $x_3 = m^2 - x_1 - x_2$ and $y_3 = m(x_1 - x_3) - y_1$, where $m = \frac{y_2 - y_1}{x_2 - x_1}$.

Double and add algorithm

Scalar multiplication is the core of ECC due to its computational asymmetry:

- The direct operation $q \to Q$ can be made efficiently (i.e. there exist some polynomial time algorithms);
- The inverse operation $Q \to q$ in general cannot be made efficiently (i.e. do not exist sub-exponential algorithms).

Double and add algorithm: $q = 41$

We decompose q according to its binary representation:

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

5 point doubling and 2 additions vs. 40 additions.

Batch validation

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

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

Insecure: introduction of random factors.

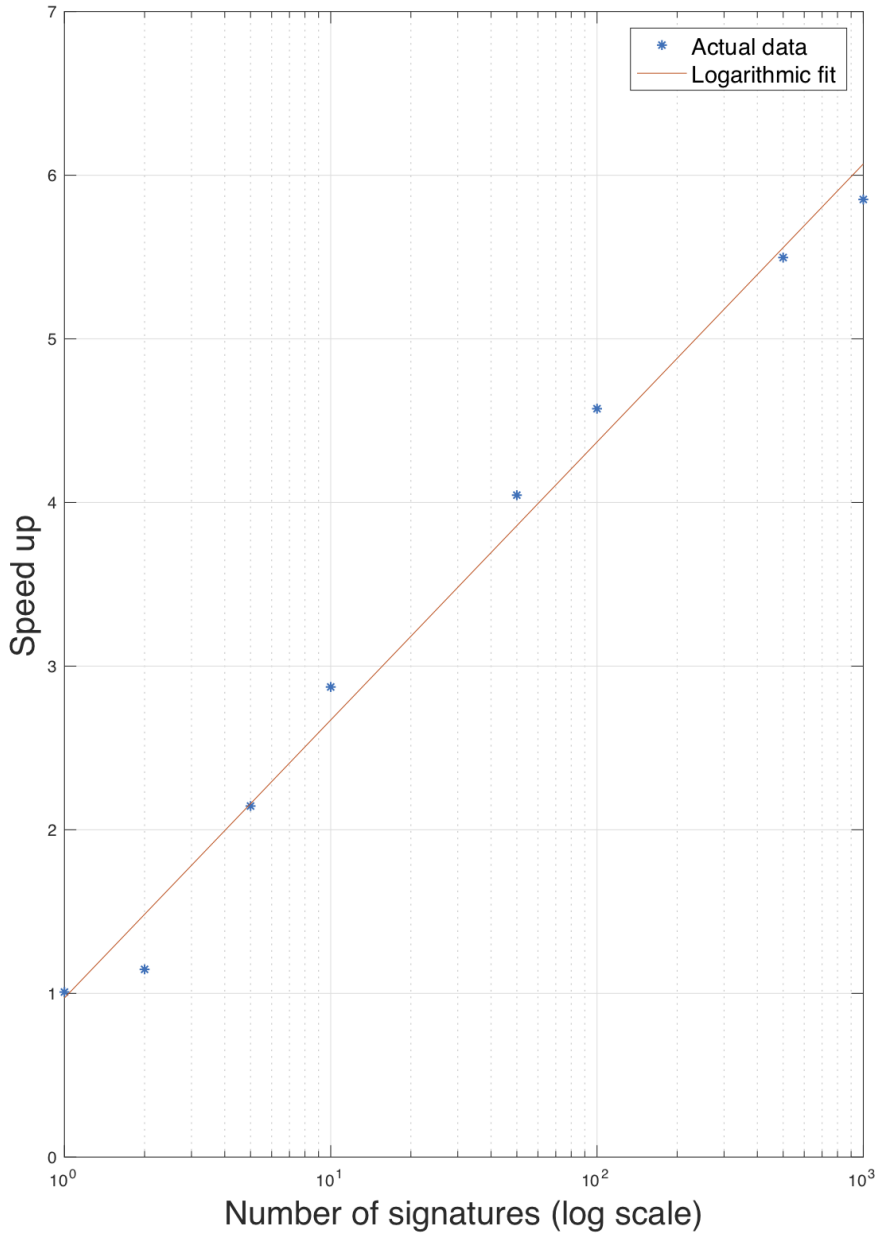
$$a_0 K_0 + a_1 K_1 =$$

$$= (a_0 s_0 + a_1 s_1)G - a_0 \mathrm{hash}(x_{K_0} \,||\, Q_0 \,||\, m_0)Q_0 - a_1 \mathrm{hash}(x_{K_1} \,||\, Q_1 \,||\, m_1)Q_1.$$

Batch validation - Bos-Coster's algorithm

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

- 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).



MuSig: compact m-of-m signature scheme

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 $a_i \leftarrow \mathsf{hash}(\langle L \rangle || Q_i)$;

a_1, a_2, a_3

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

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

1: Alice

t_1

2: Bob

t_1

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

1: Alice

t_2

2: Bob

t_3

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

K_1

1: Alice

K_1

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

1: Alice

K_2

2: Bob

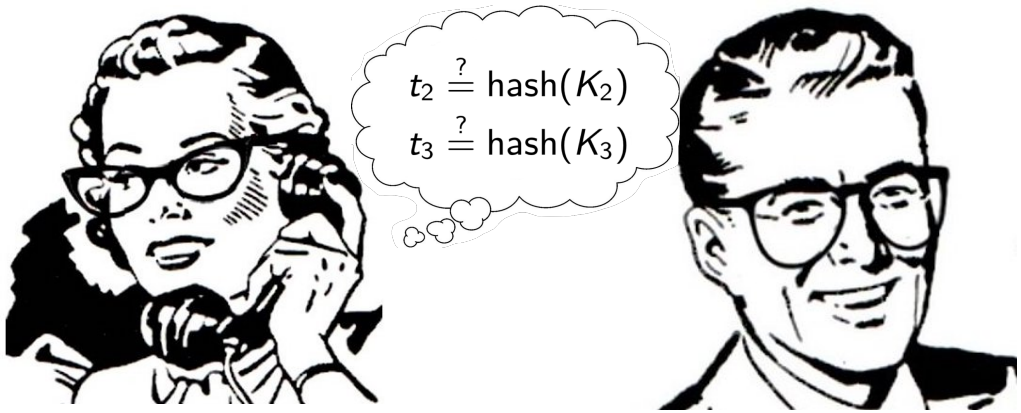
K_3

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

K

1: Alice

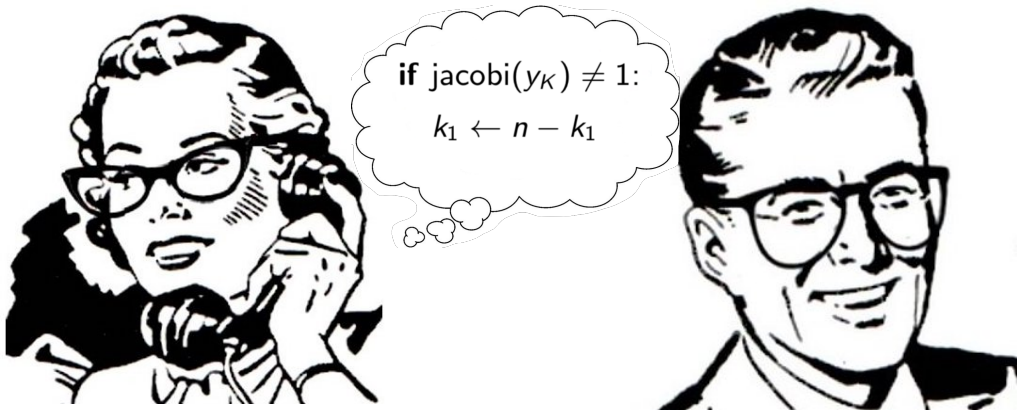
2: Bob

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. $k_1 \xleftarrow{\$} \{1, ..., n-1\}$;
4. $K_1 \leftarrow k_1 G, \ t_1 \leftarrow \text{hash}(K_1)$;
5. **send** t_1, K_1;
6. $K \leftarrow \sum_{i=1}^{m} K_i$;



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;
6. $K \leftarrow \sum_{i=1}^{m} K_i$;
7. $c \leftarrow \text{hash}(x_K || Q || m)$;

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;
6. $K \leftarrow \sum_{i=1}^{m} K_i$;
7. $c \leftarrow \text{hash}(x_K || Q || m)$;
8. $s_1 \leftarrow k_1 + c a_1 q_1 \pmod{n}$;

s_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;
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;

1: Alice

s_1

2: Bob

s_1

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. $K \leftarrow \sum_{i=1}^{m} K_i$;
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;

s_2

1: Alice

s_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. $K \leftarrow \sum_{i=1}^{m} K_i$;
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. $k_1 \xleftarrow{\$} \{1, ..., n-1\}$;
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).

(x_K, s)

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

Dealer: Alice

r, r'

The dealer:

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

1: Bob

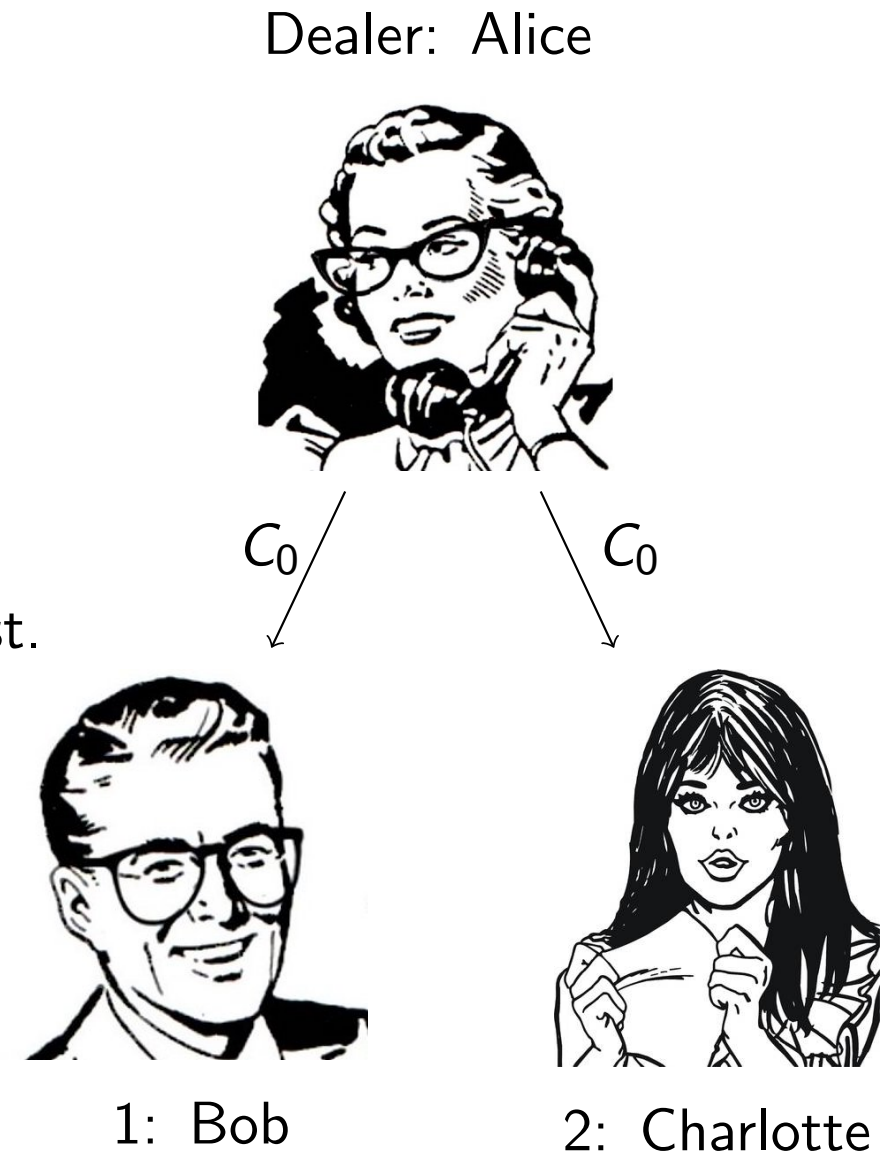
2: Charlotte

Threshold signature scheme (t-of-m)

Verifiable secret sharing scheme

The dealer:

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



Threshold signature scheme (t-of-m)

Verifiable secret sharing scheme

Dealer: Alice

f, f'

The dealer:

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

1: Bob

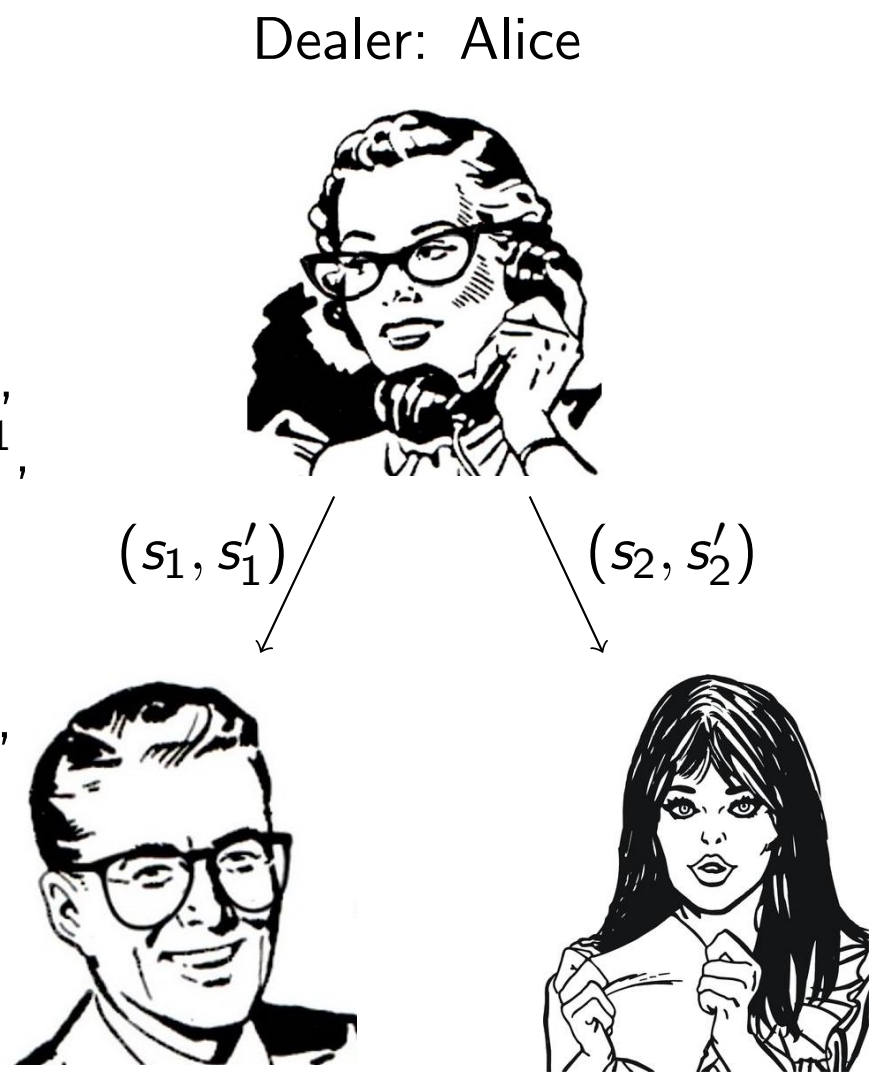
2: Charlotte

Threshold signature scheme (t-of-m)

Verifiable secret sharing scheme

The dealer:

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

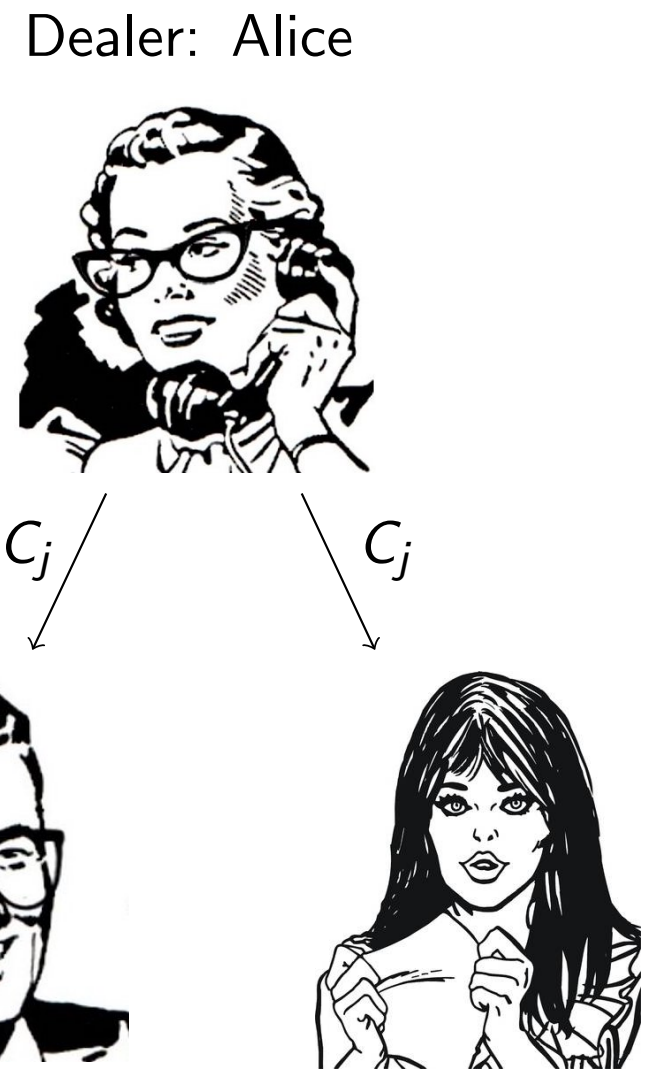


Threshold signature scheme (t-of-m)

Verifiable secret sharing scheme

The dealer:

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



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

Dealer: Alice

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
  $\omega_i(u) = \prod_{j \neq i} \frac{u-j}{i-j} \pmod n$.
  $r = f(0) = \sum_i s_i \omega_i$, with
  $\omega_i = \omega_i(0) = \prod_{j \neq i} \frac{j}{j-i} \pmod n$.

s

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, r_1'
f_1, f_1'

r_2, r_2'
f_2, f_2'

2: Bob

r_3, r_3'
f_3, f_3'

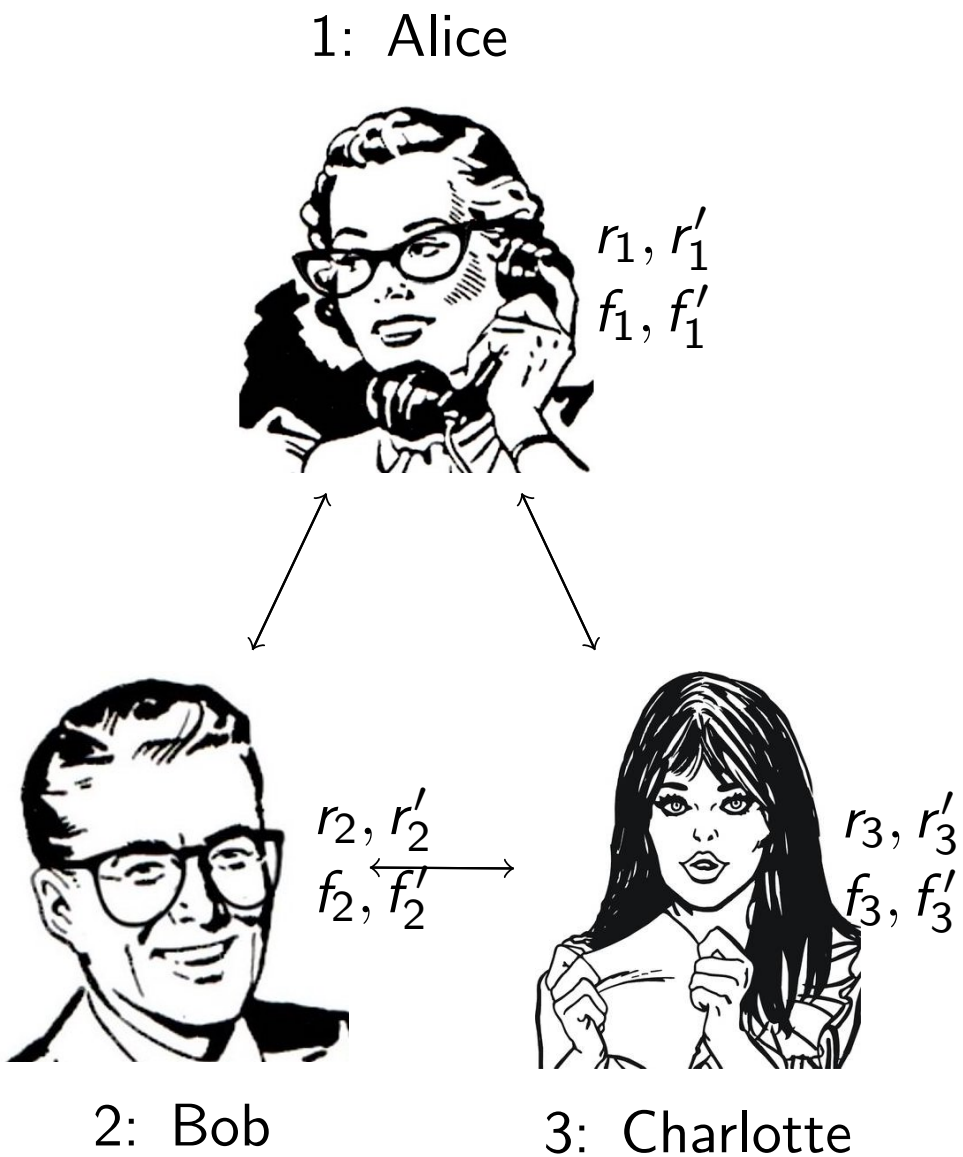
3: Charlotte

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

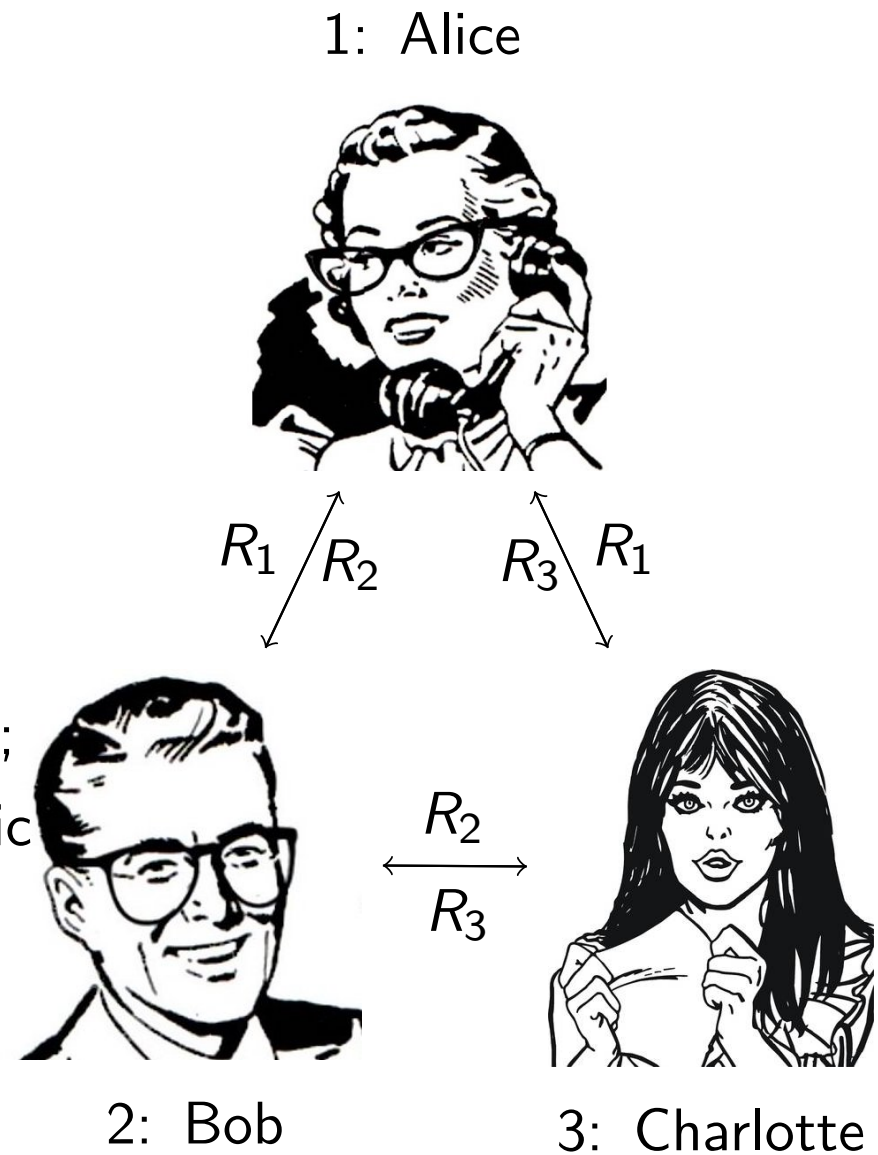
3: Charlotte

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

1: Alice

α_1

α_3

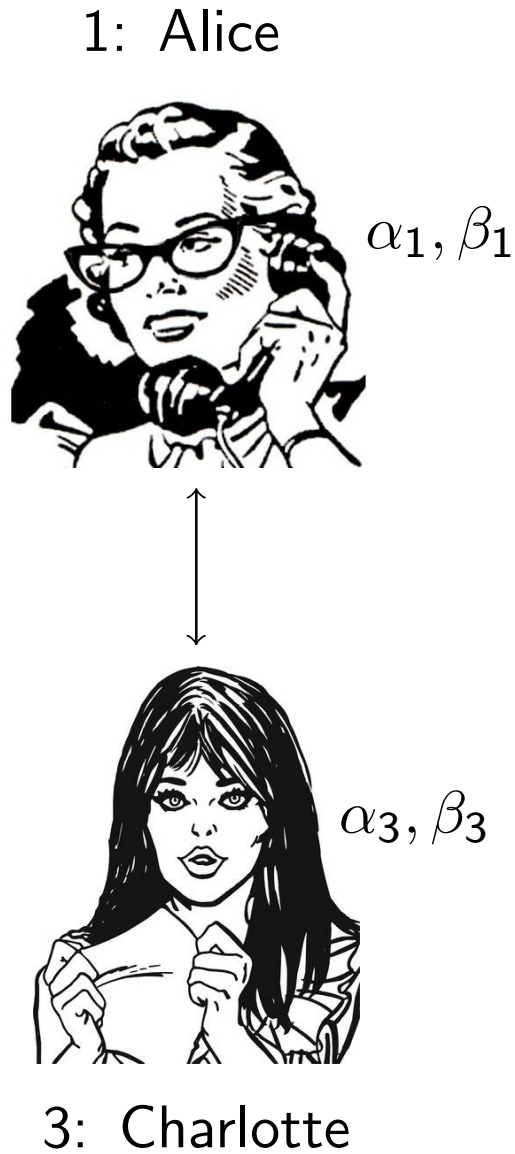
3: Charlotte

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: $(\beta_1, ..., \beta_m) \xleftrightarrow{(\text{t, m})} (k|K)$.



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

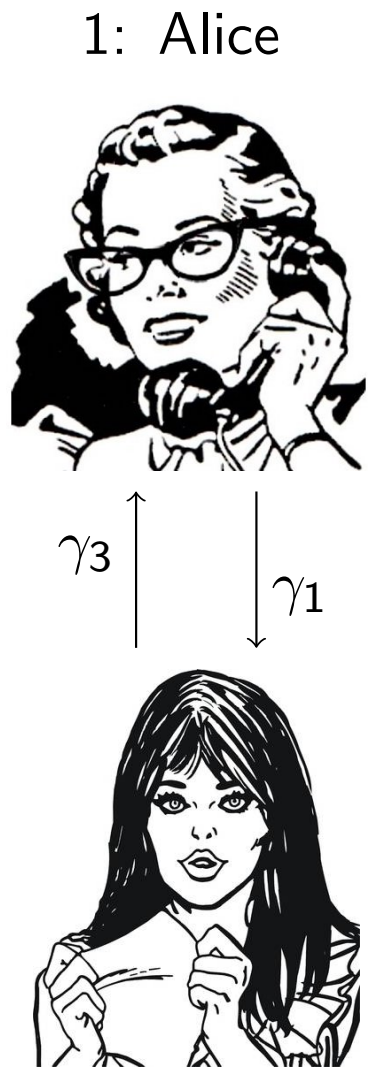
3: Charlotte

Threshold signature scheme (t-of-m)

Threshold scheme

Then each signer i:

- checks whether 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 $e = \text{hash}(x_K||Q||msg)$;



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 $\sigma = k + eq \pmod{n}$;

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).

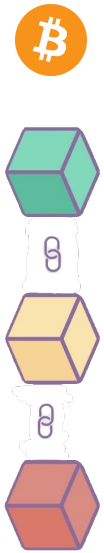
1: Alice

(x_K, σ)

(x_K, σ)

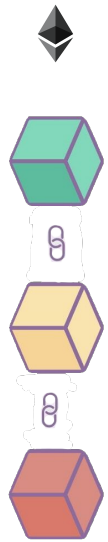
3: Charlotte

Cross-chain atomic swaps via adaptor signatures

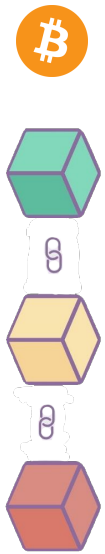


Alice

Bob



Cross-chain atomic swaps via adaptor signatures

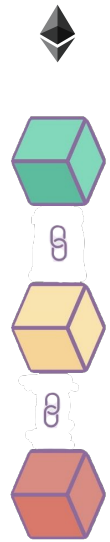


Q_A, Q'_A

Alice

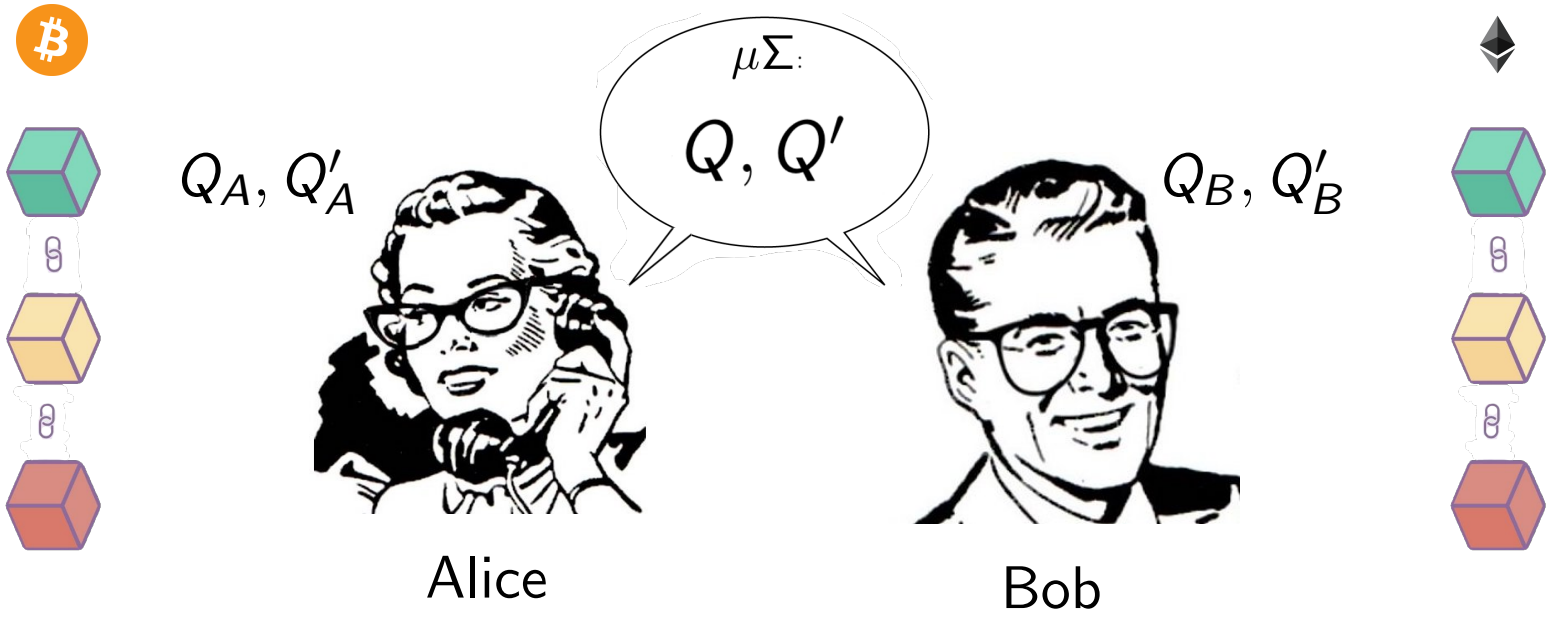
Q_B, Q'_B

Bob



The participants have a key pair for each chain:
$\{q_A, Q_A\}$, $\{q'_A, Q'_A\}$, $\{q_B, Q_B\}$, $\{q'_B, Q'_B\}$.

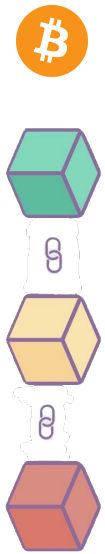
Cross-chain atomic swaps via adaptor signatures



According to the MuSig protocol they construct the aggregated public keys:

$$Q = \text{hash}(\langle Q_A, Q_B \rangle || Q_A) Q_A + \text{hash}(\langle Q_A, Q_B \rangle || Q_B) Q_B,$$
$$Q' = \text{hash}(\langle Q'_A, Q'_B \rangle || Q'_A) Q'_A + \text{hash}(\langle Q'_A, Q'_B \rangle || Q'_B) Q'_B.$$

Cross-chain atomic swaps via adaptor signatures



Q_A, Q'_A

Q, Q'

Alice

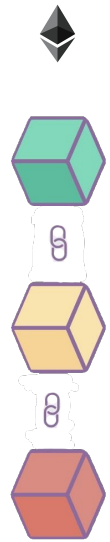
$\boxed{\mathsf{TX}_1}$

Q_B, Q'_B

Q, Q'

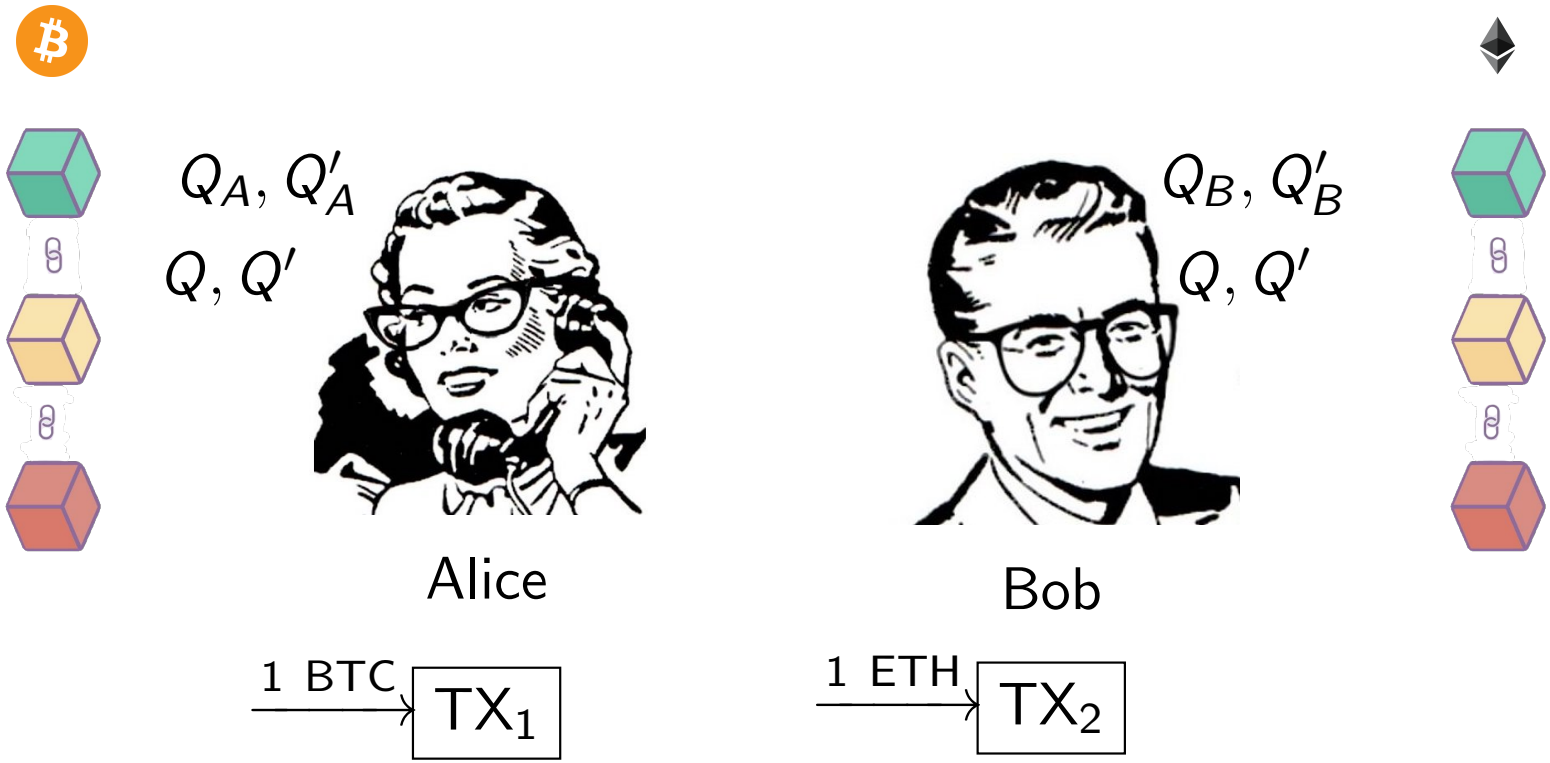
Bob

$\boxed{\mathsf{TX}_2}$



They start constructing transactions on the respective chain:

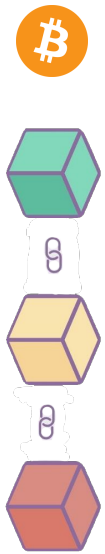
Cross-chain atomic swaps via adaptor signatures



They start constructing transactions on the respective chain:

- They agree on the exchange rate;

Cross-chain atomic swaps via adaptor signatures



Q_A, Q'_A

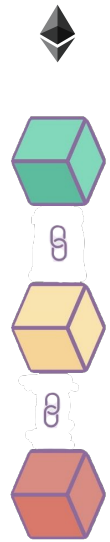
Q, Q'

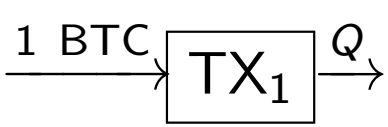
Alice

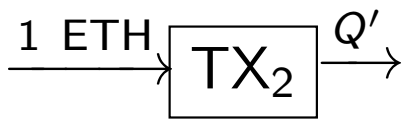
Q_B, Q'_B

Q, Q'

Bob



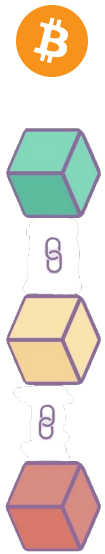




They start constructing transactions on the respective chain:

- They agree on the exchange rate;
- The funds are locked with the aggregated public keys.

Cross-chain atomic swaps via adaptor signatures



Q_A, Q'_A

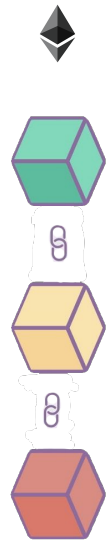
Q, Q'

Alice

Q_B, Q'_B

Q, Q'

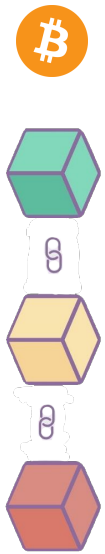
Bob



$1\ \text{BTC} \longrightarrow \boxed{\text{TX}_1} \xrightarrow{Q} \boxed{\text{REF}_1}$

Before signing, they construct a refund transaction (here Alice's example). Both transactions have to be time locked and the time lock over Bob's transaction has to be greater.

Cross-chain atomic swaps via adaptor signatures



Q_A, Q'_A

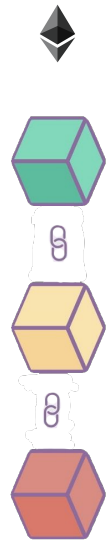
Q, Q'

Alice

Q_B, Q'_B

Q, Q'

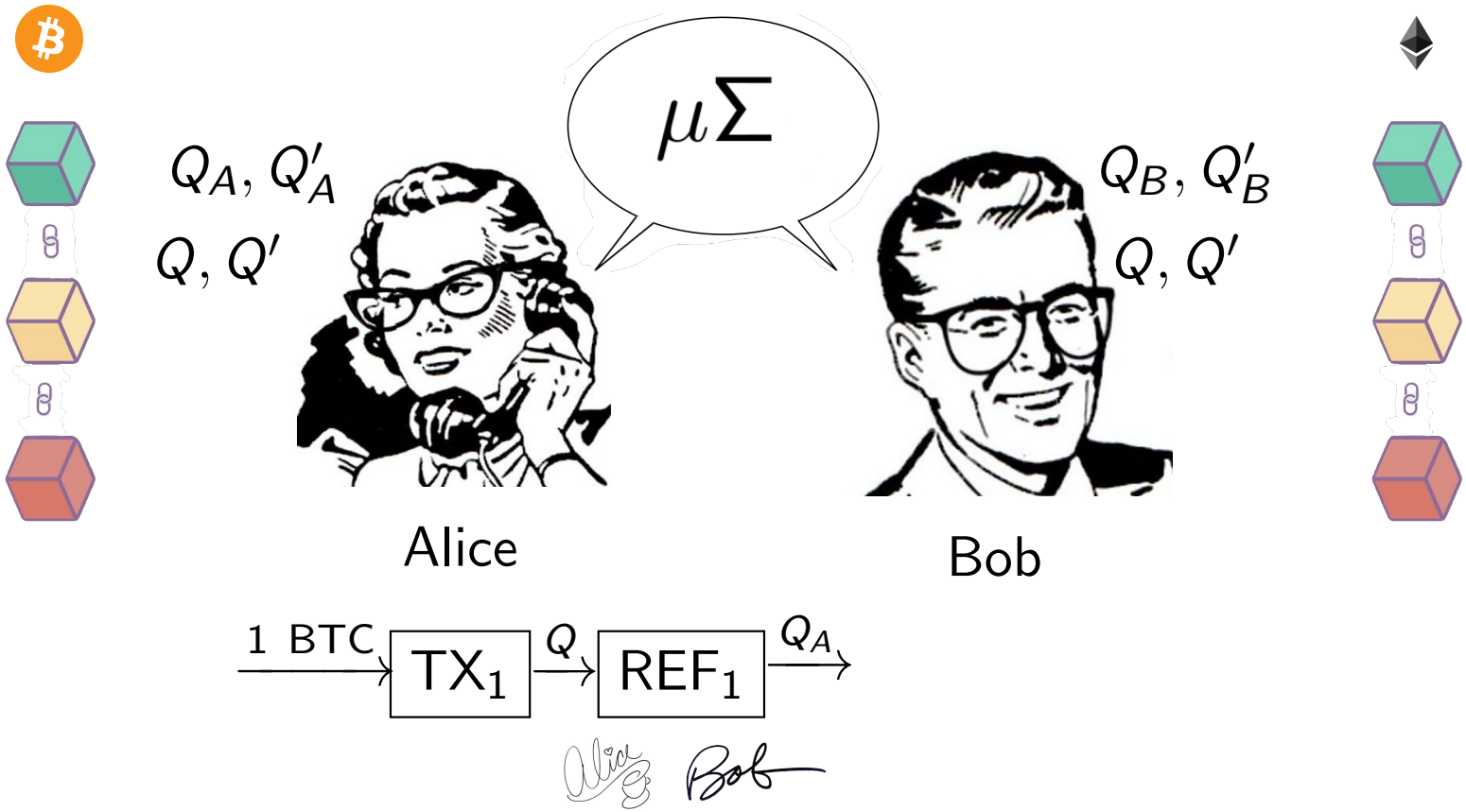
Bob



$$1\ \text{BTC} \longrightarrow \boxed{\text{TX}_1} \xrightarrow{Q} \boxed{\text{REF}_1} \xrightarrow{Q_A}$$

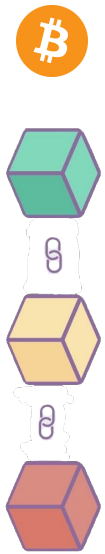
Before signing, they construct a refund transaction (here Alice's example). Both transactions have to be time locked and the time lock over Bob's transaction has to be greater.

Cross-chain atomic swaps via adaptor signatures



Before signing, they construct a refund transaction (here Alice's example). Both transactions have to be time locked and the time lock over Bob's transaction has to be greater. To sign it, the participants have to interact via the MuSig protocol.

Cross-chain atomic swaps via adaptor signatures



Q_A, Q'_A

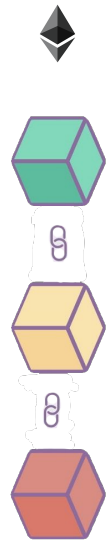
Q, Q'

Alice

Q_B, Q'_B

Q, Q'

Bob

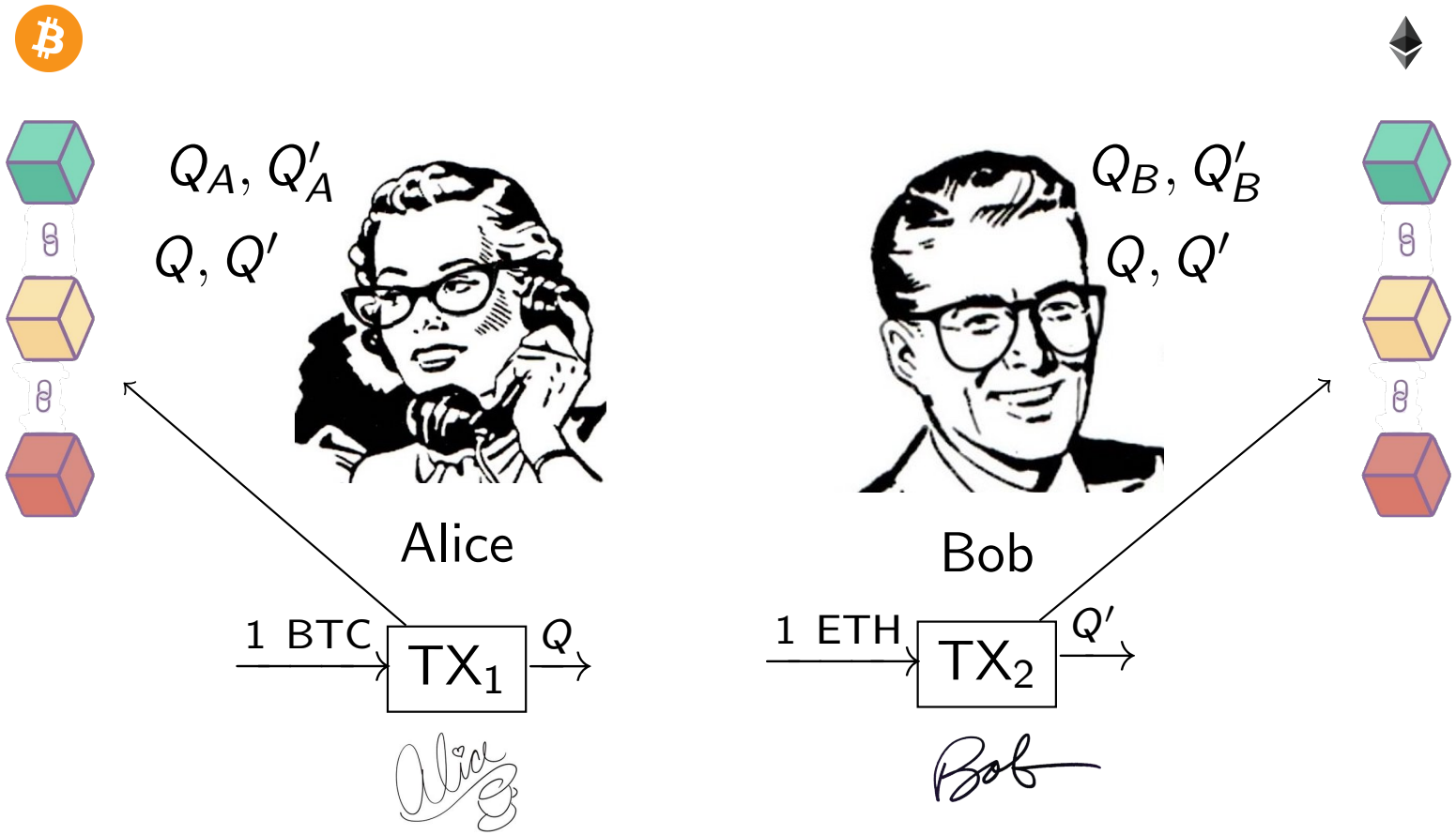


1 BTC → TX_1 → Q

1 ETH → TX_2 → Q'

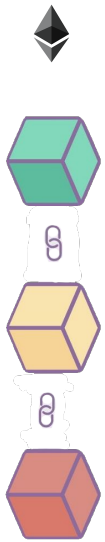
Having the refund transactions, Alice and Bob can confidently sign TX_1 and TX_2.

Cross-chain atomic swaps via adaptor signatures



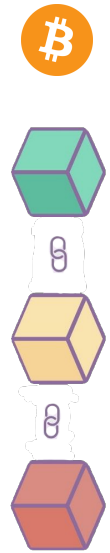
Having the refund transactions, Alice and Bob can confidently sign TX_1 and TX_2. Then they broadcast them and wait for confirmation.

Cross-chain atomic swaps via adaptor signatures



Alice

Bob

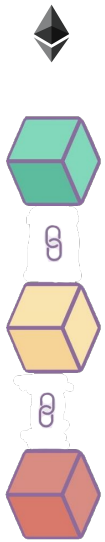


$\boxed{\mathrm{TX}_2} \xrightarrow{Q'}$

$\boxed{\mathrm{TX}_1} \xrightarrow{Q}$

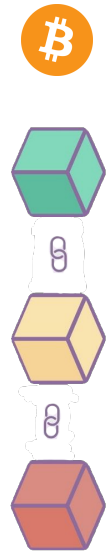
At this point they construct transactions to grab the other coin:

Cross-chain atomic swaps via adaptor signatures



Alice

Bob

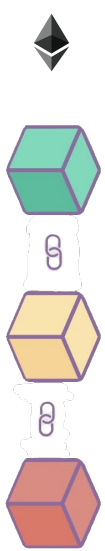


$\boxed{\mathrm{TX}_2} \xrightarrow{Q'} \boxed{\mathrm{TX}_4}$

$\boxed{\mathrm{TX}_1} \xrightarrow{Q} \boxed{\mathrm{TX}_3}$

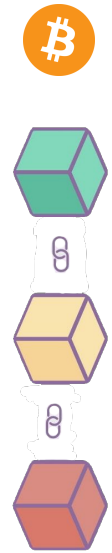
At this point they construct transactions to grab the other coin:

Cross-chain atomic swaps via adaptor signatures



Alice

Bob

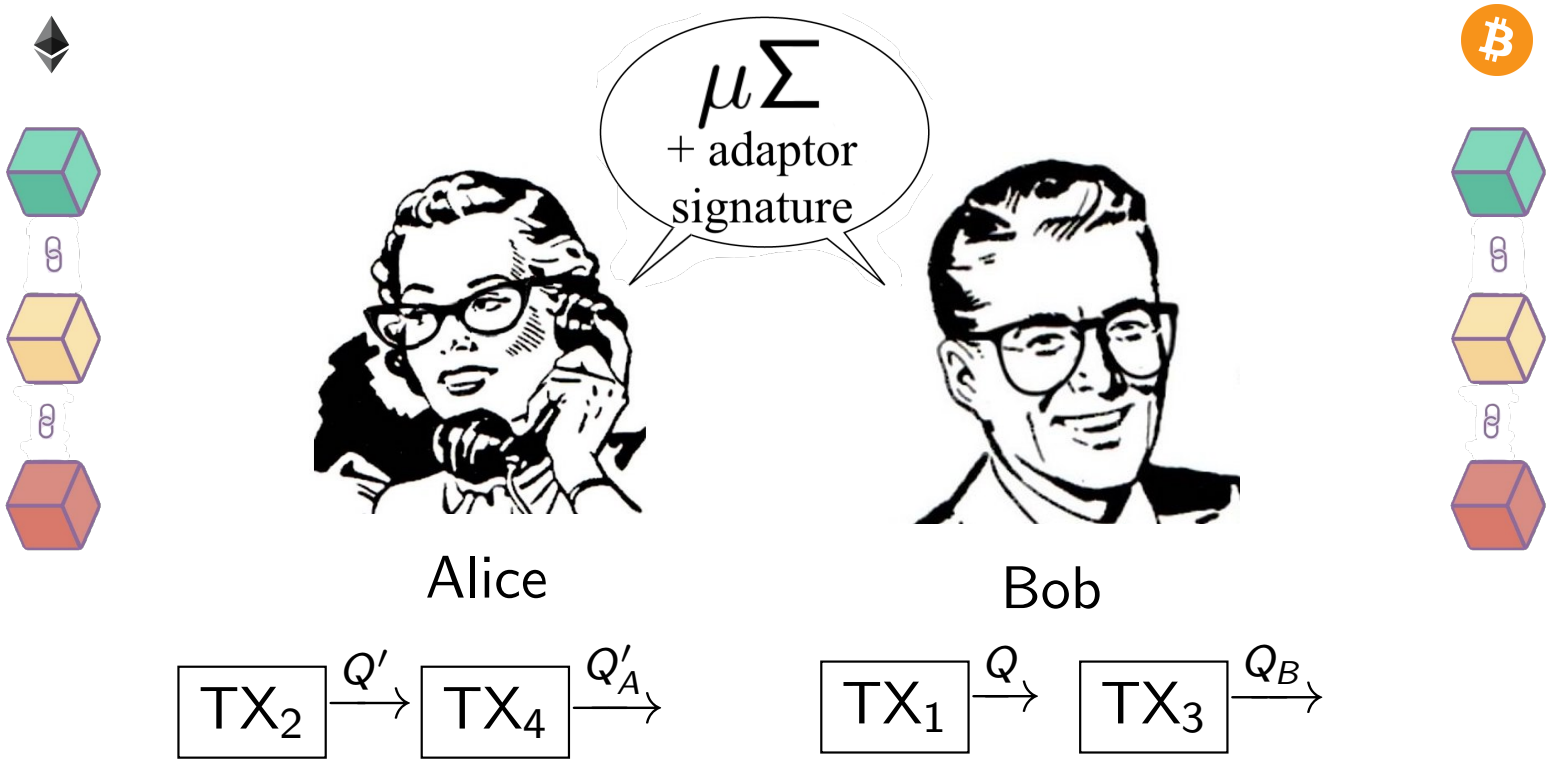


$$\boxed{\mathrm{TX}_2} \xrightarrow{Q'} \boxed{\mathrm{TX}_4} \xrightarrow{Q'_A}$$

$$\boxed{\mathrm{TX}_1} \xrightarrow{Q} \boxed{\mathrm{TX}_3} \xrightarrow{Q_B}$$

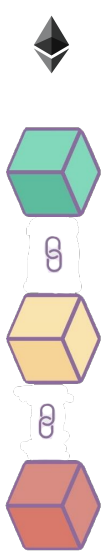
At this point they construct transactions to grab the other coin:

Cross-chain atomic swaps via adaptor signatures



At this point they construct transactions to grab the other coin: they need to resort to the MuSig protocol to generate valid signatures; to ensure atomicity, Bob produces two adaptor signatures using the same secret value t.

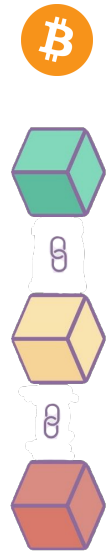
Cross-chain atomic swaps via adaptor signatures

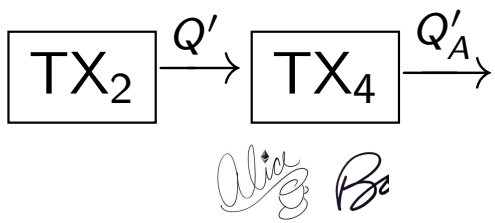


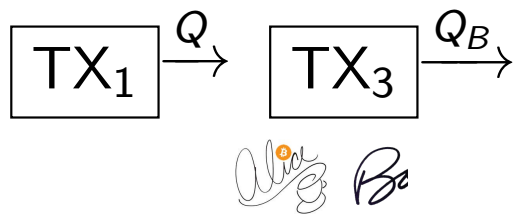
Alice

t

Bob

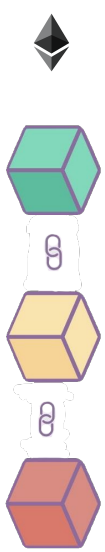






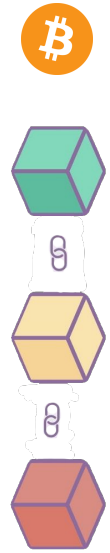
Alice has to check that the two signatures are valid adaptor signatures and that it has been used the same secret value.

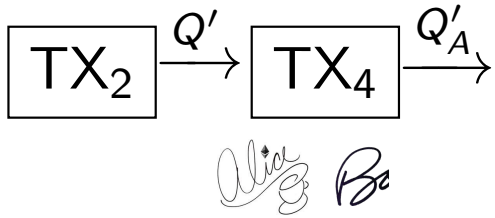
Cross-chain atomic swaps via adaptor signatures



Alice

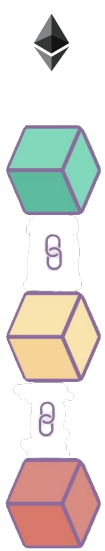
Bob





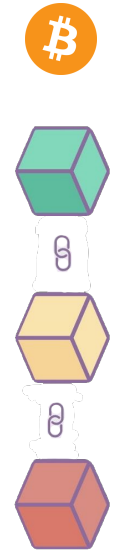
This results in two partial (invalid) signatures: $(x_{K_A+K_B+T}, \tilde{s})$ and $(x_{K'_A+K'_B+T}, \tilde{s}')$, with
$\tilde{s} = k_A + k_B + \text{hash}(x_{K_A+K_B+T}||Q||\text{TX}_3)(q_A + q_B)$ and
$\tilde{s}' = k'_A + k'_B + \text{hash}(x_{K'_A+K'_B+T}||Q'||\text{TX}_4)(q'_A + q'_B)$.

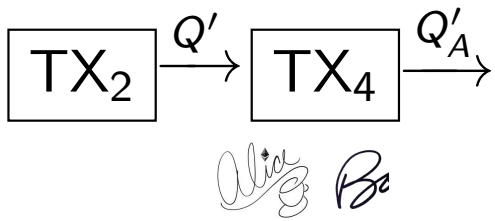
Cross-chain atomic swaps via adaptor signatures



Alice

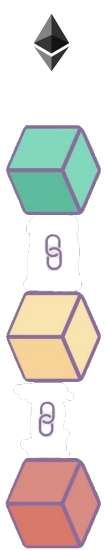
Bob





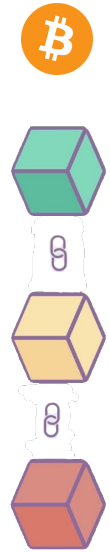
Whenever Bob wants to take the coins he can add t to the adaptor signature, making it valid: $(x_{K_A+K_B+T}, \tilde{s} + t)$.

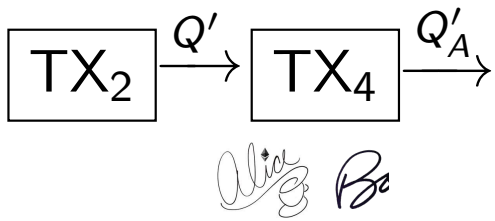
Cross-chain atomic swaps via adaptor signatures



Alice

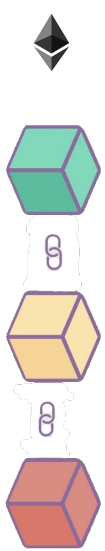
Bob





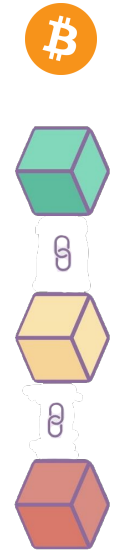
Whenever Bob wants to take the coins he can add t to the adaptor signature, making it valid: $(x_{K_A+K_B+T}, \tilde{s} + t)$.

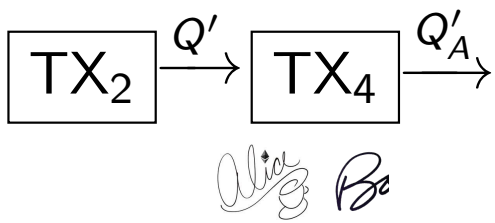
Cross-chain atomic swaps via adaptor signatures

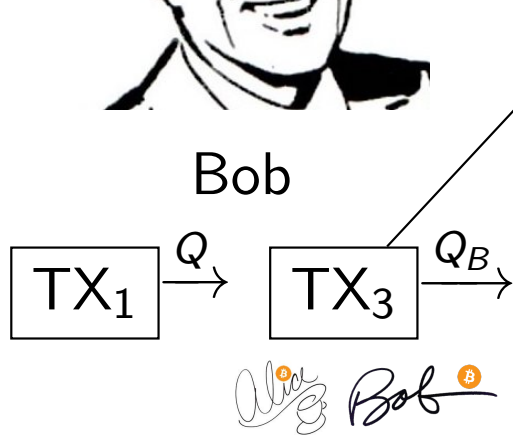


Alice

Bob

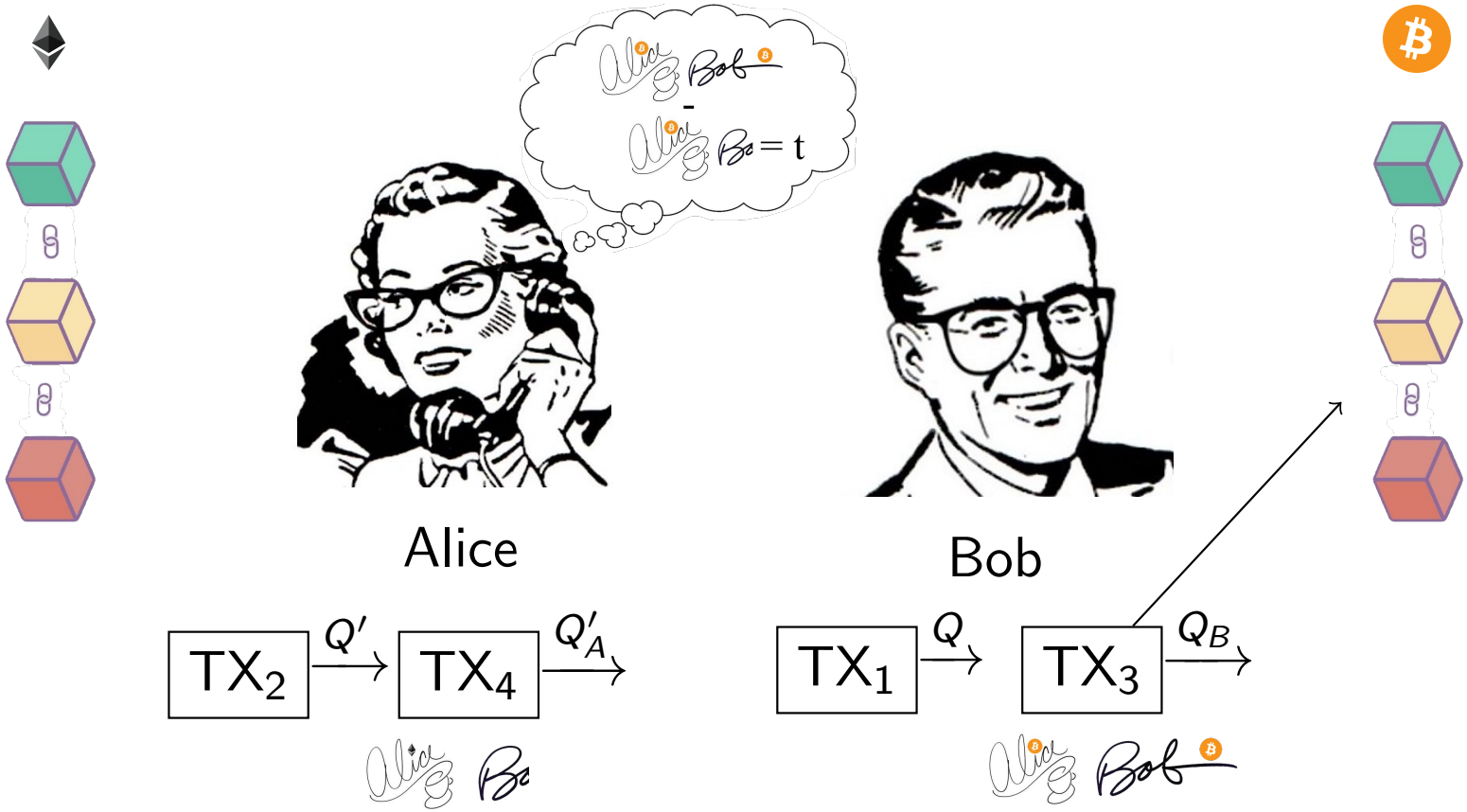






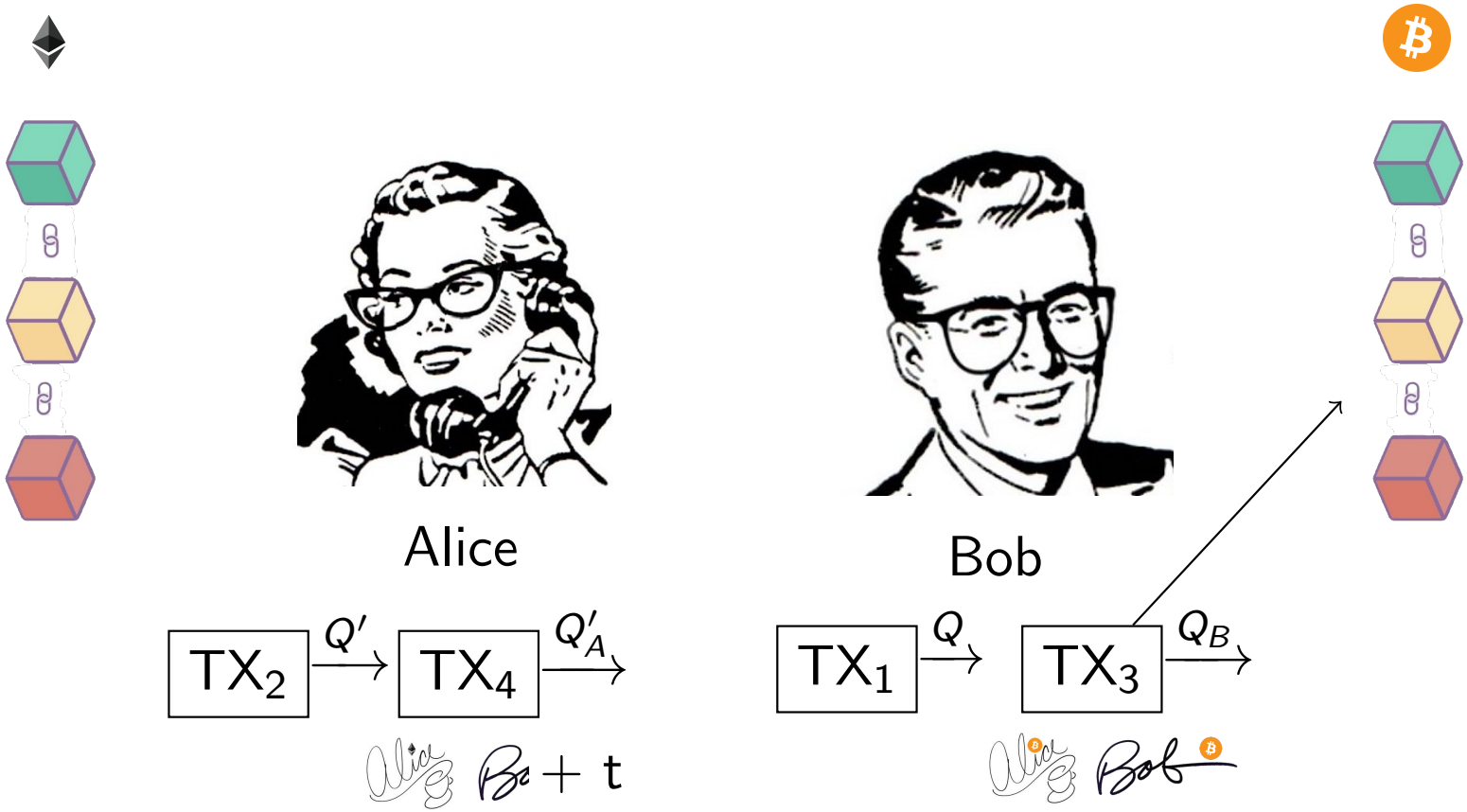
Now he can broadcast to the blockchain.

Cross-chain atomic swaps via adaptor signatures



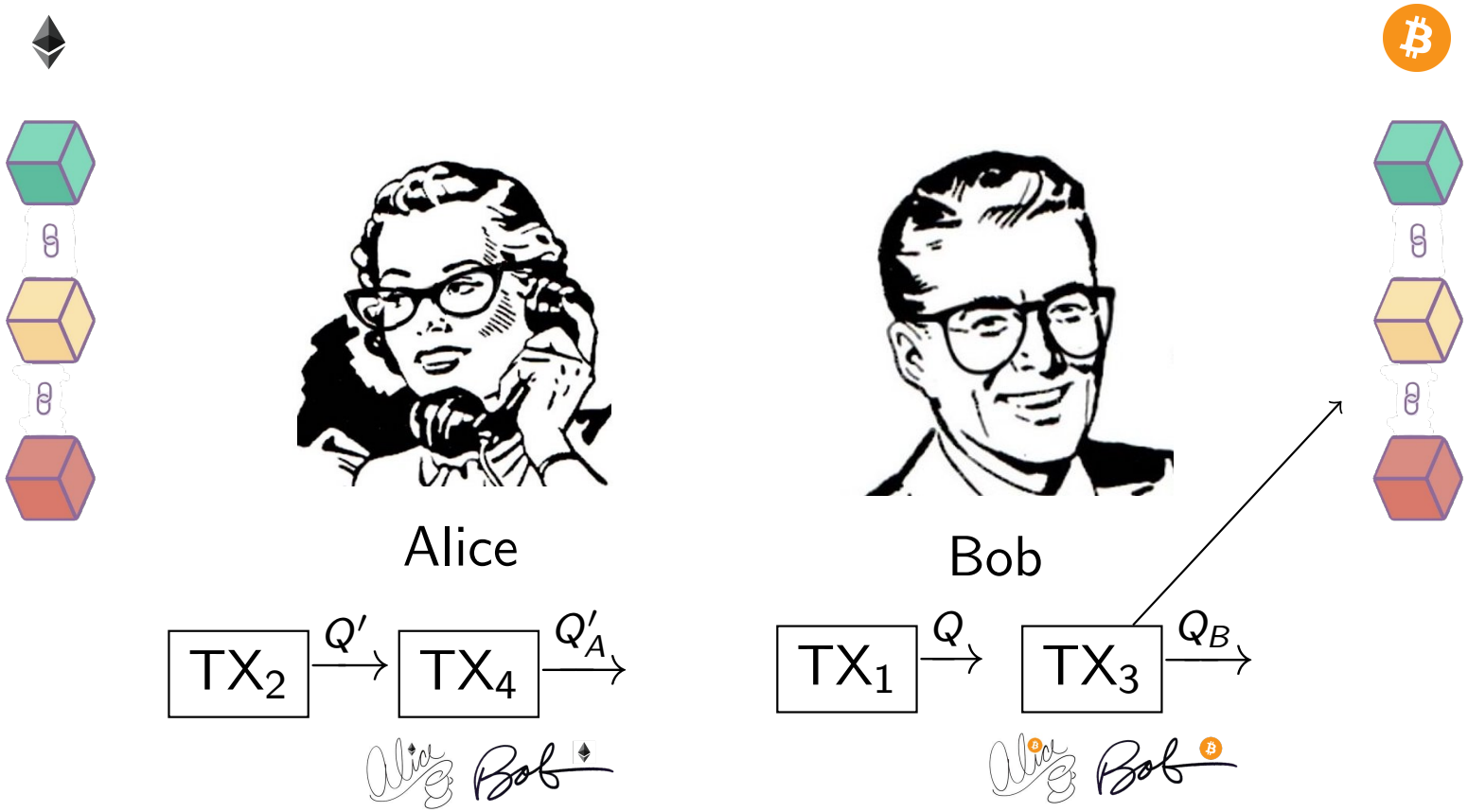
Monitoring the blockchain Alice sees the valid signature: subtracting from it the adaptor signature she learns the secret t.

Cross-chain atomic swaps via adaptor signatures



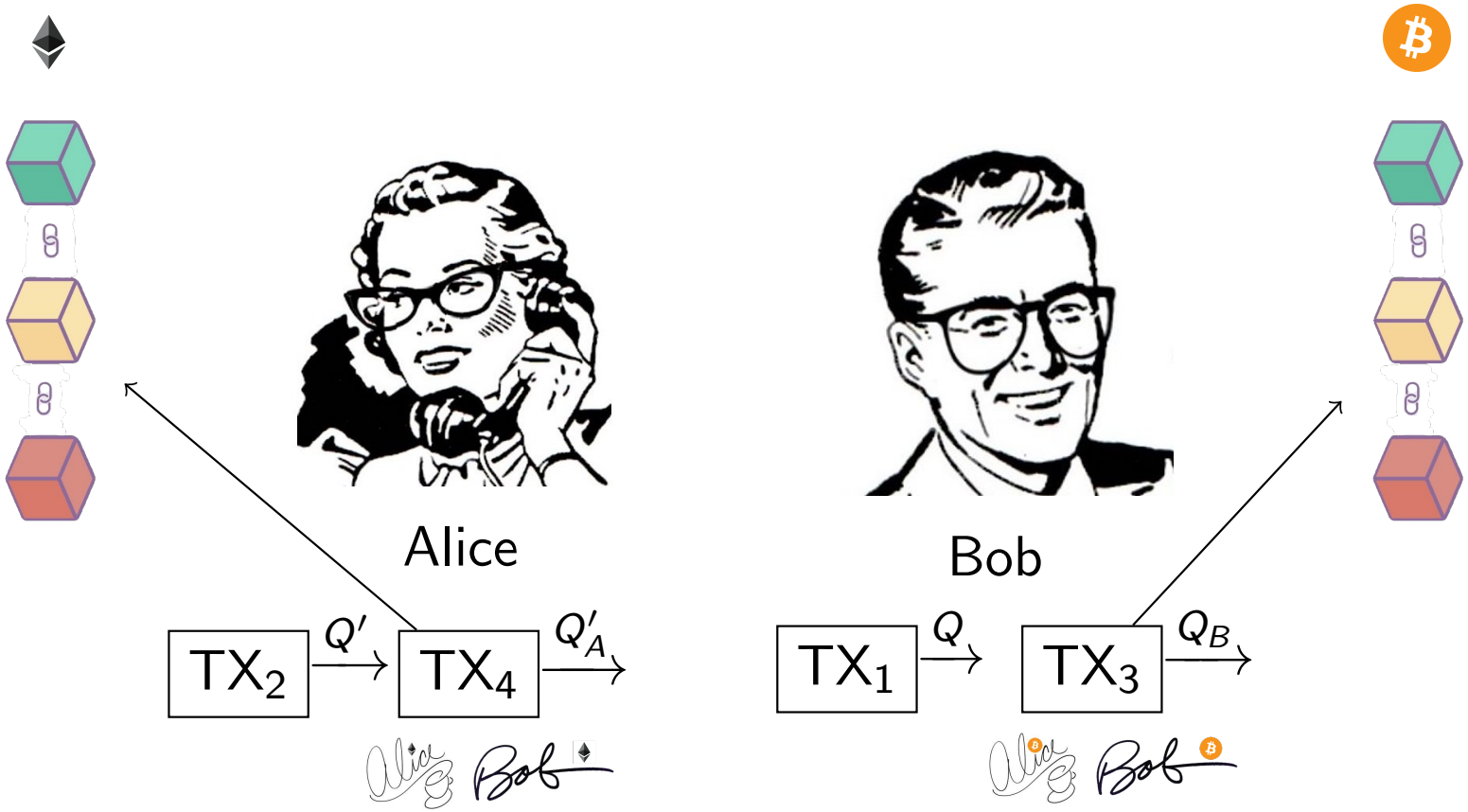
Adding it to the other adaptor signature results again in a valid signature for TX_4.

Cross-chain atomic swaps via adaptor signatures



Adding it to the other adaptor signature results again in a valid signature for TX_4.

Cross-chain atomic swaps via adaptor signatures



She can now broadcast it to the blockchain: this concludes the protocol.