



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

20th 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. 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

Mathematical background and cryptographic primitives

- Hash functions

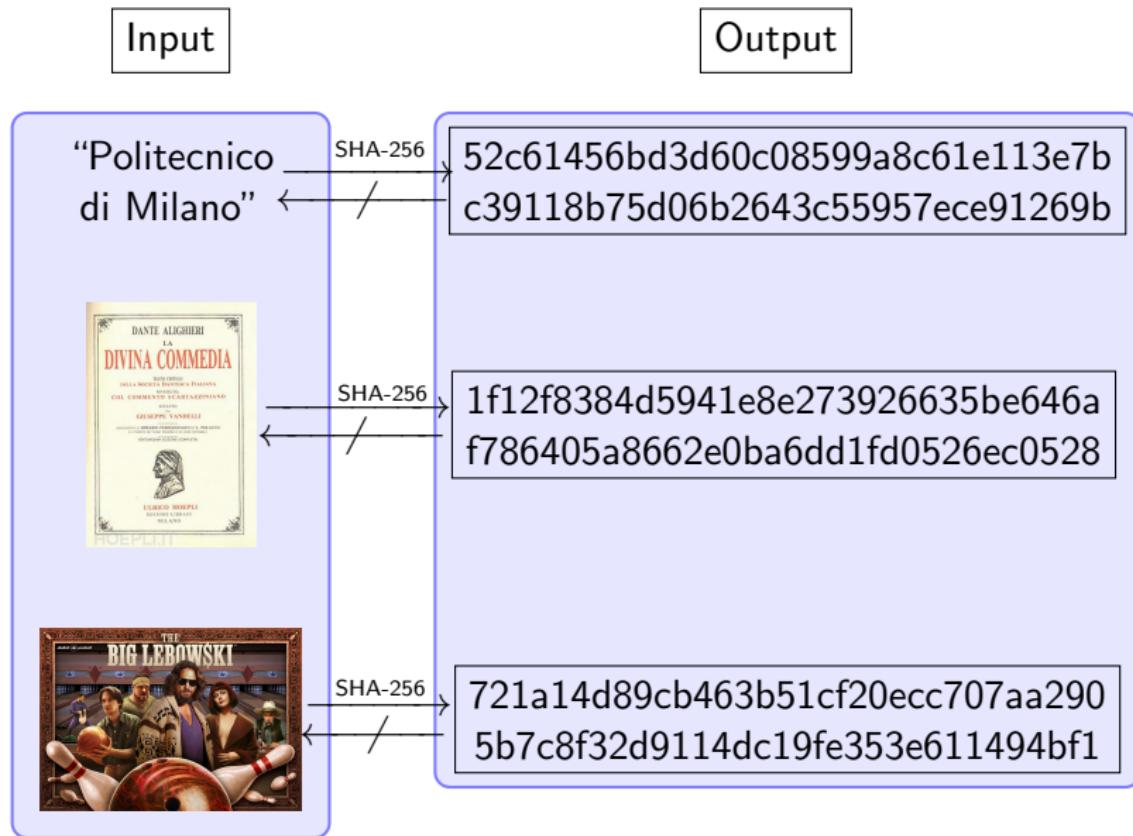
- Elliptic curve cryptography

Digital signature schemes

ECSSA applications

Conclusion

Hash functions (\simeq Random functions)



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:

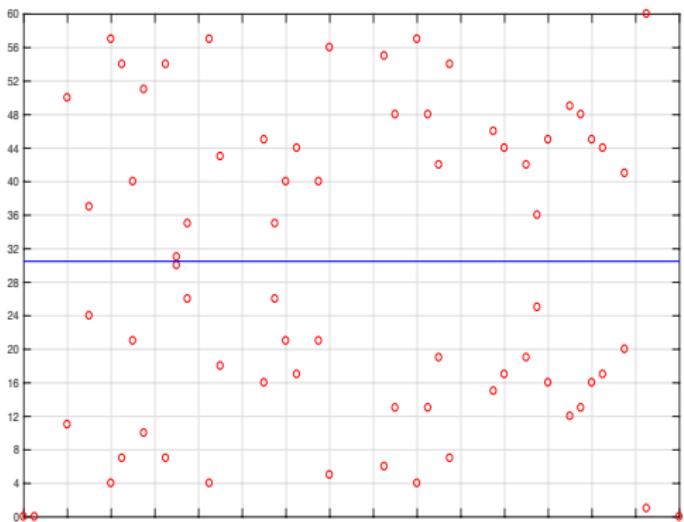
$$Q_3 := Q_1 + Q_2,$$

$$\forall Q_1, Q_2 \in E(\mathbb{F}_p);$$

- ▶ Scalar multiplication:

$$nG := G + \dots + G,$$

$$\forall G \in E(\mathbb{F}_p), \forall n \in \mathbb{N}.$$



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

Discrete logarithm problem

Fixed $G \in E(\mathbb{F}_p)$, we can define $Q = qG \quad \forall q \in \mathbb{N}$:

- ▶ 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.

Outline

Mathematical background and cryptographic primitives

Digital signature schemes

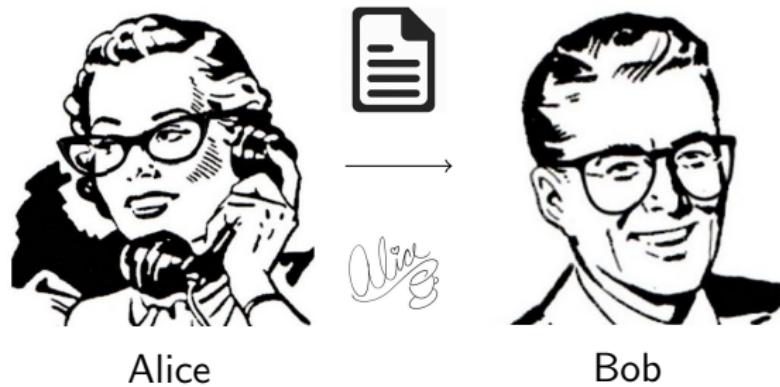
ECDSA

ECSSA

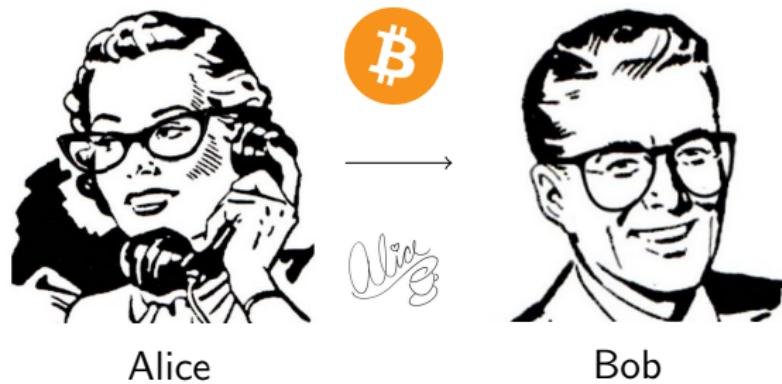
ECSSA applications

Conclusion

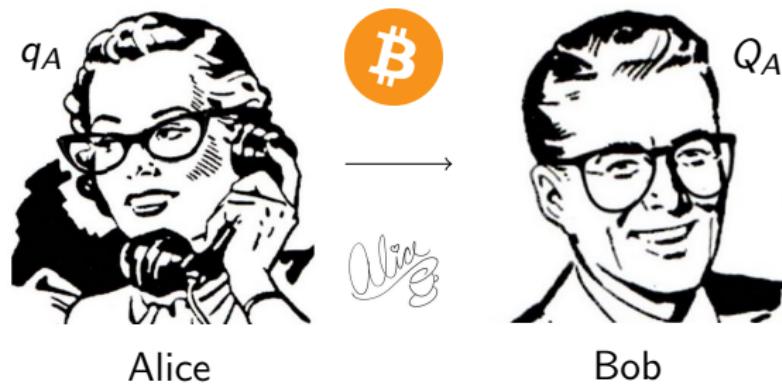
Digital signature



Digital signature

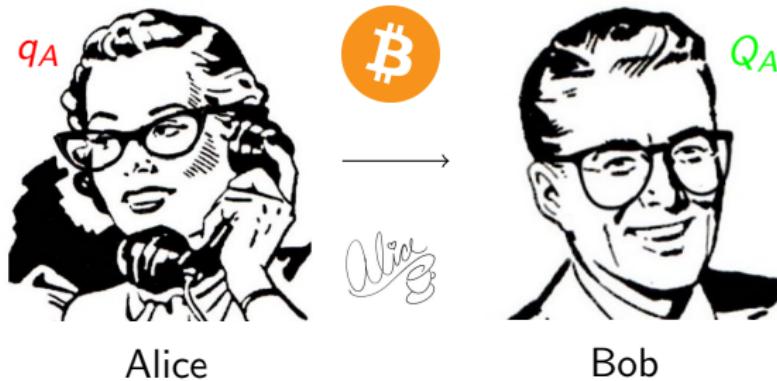


Digital signature



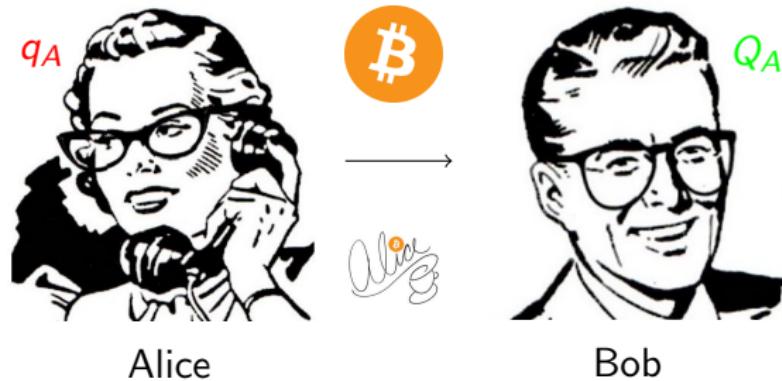
- ▶ Authentication: the recipient is confident that the message comes from the alleged sender;

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

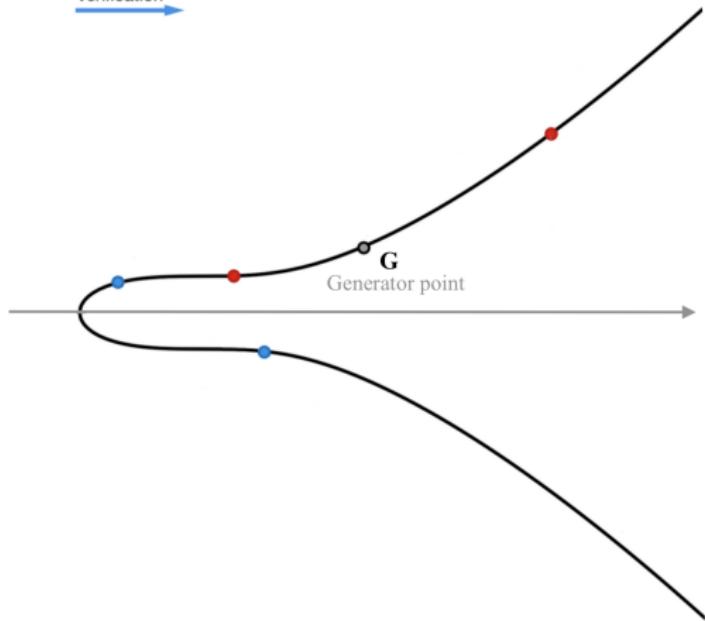


- ▶ Authentication: the recipient is confident that the message comes from the alleged sender;
- ▶ Non repudiation: the sender cannot deny having sent the message;
- ▶ Integrity: ensures that the message has not been altered during transmission.

Elliptic curve digital signature algorithm

Signing
→
Verification
→

ECDSA_SIG(m, q):

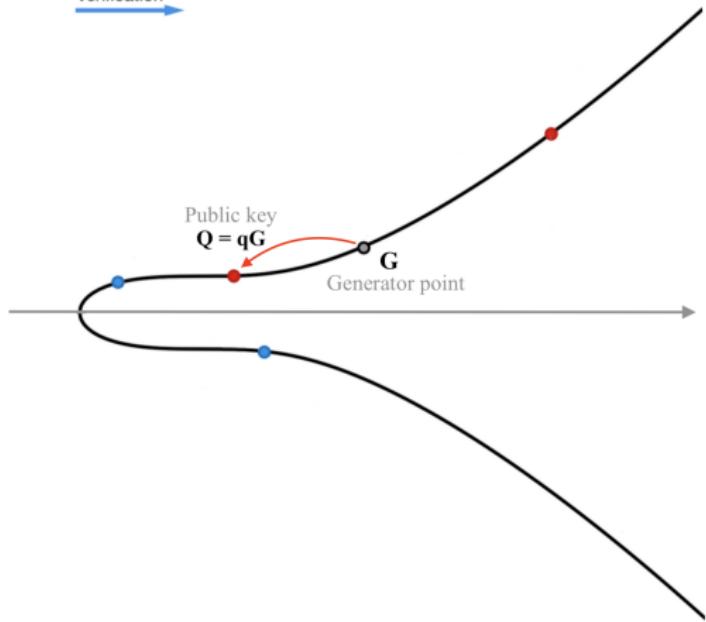


Adapted from:
<https://medium.com/cryptoadvance/how-schnorr-signatures-may-improve-bitcoin-91655bcb4744>

Elliptic curve digital signature algorithm

Signing
→
Verification
→

ECDSA_SIG(m, q):



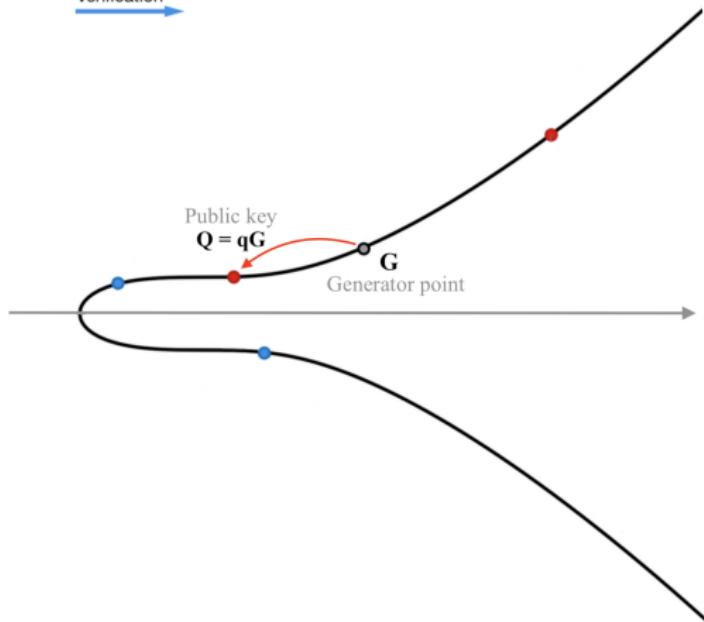
Adapted from:
<https://medium.com/cryptoadvance/how-schnorr-signatures-may-improve-bitcoin-91655bcb4744>

Elliptic curve digital signature algorithm

Signing
→
Verification
→

ECDSA_SIG(m, q):

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



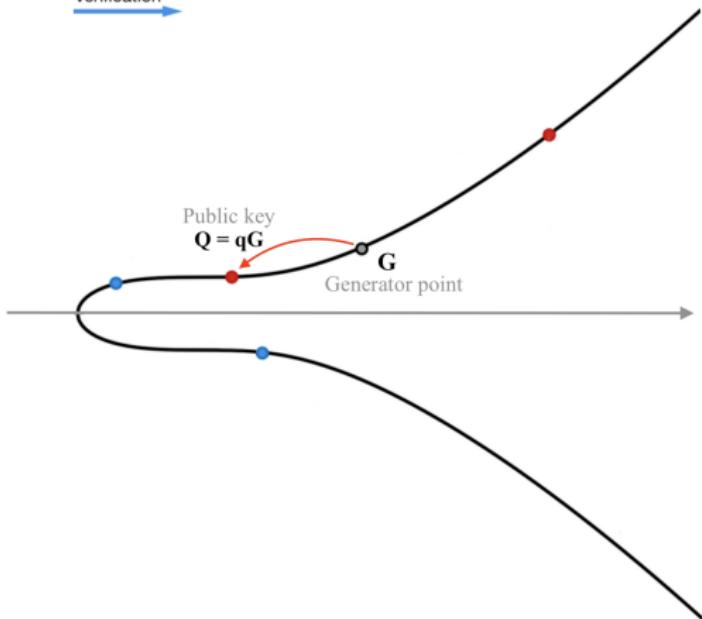
Adapted from:
<https://medium.com/cryptoadvance/how-schnorr-signatures-may-improve-bitcoin-91655bcb4744>

Elliptic curve digital signature algorithm

Signing
→
Verification
→

ECDSA_SIG(m, q):

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



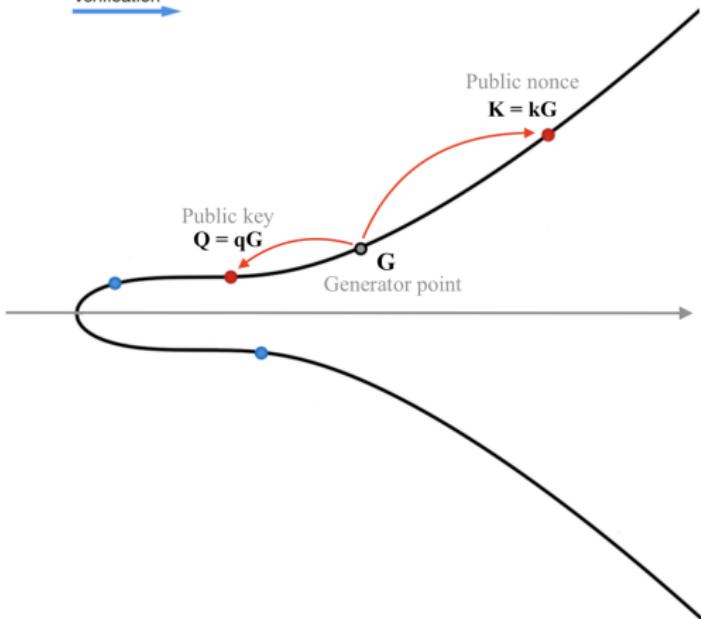
Adapted from:
<https://medium.com/cryptoadvance/how-schnorr-signatures-may-improve-bitcoin-91655bcb4744>

Elliptic curve digital signature algorithm

Signing
→
Verification
→

ECDSA_SIG(m, q):

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



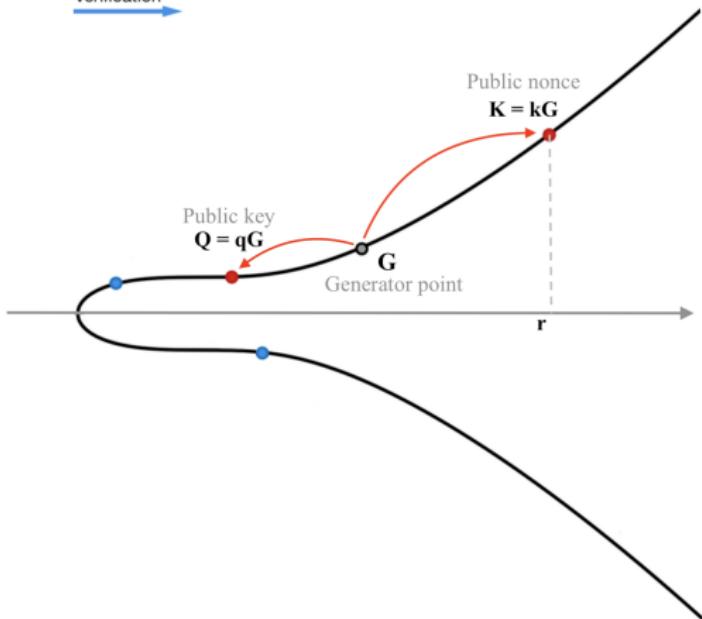
Adapted from:
<https://medium.com/cryptoadvance/how-schnorr-signatures-may-improve-bitcoin-91655bcb4744>

Elliptic curve digital signature algorithm

Signing
→
Verification
→

ECDSA_SIG(m, q):

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



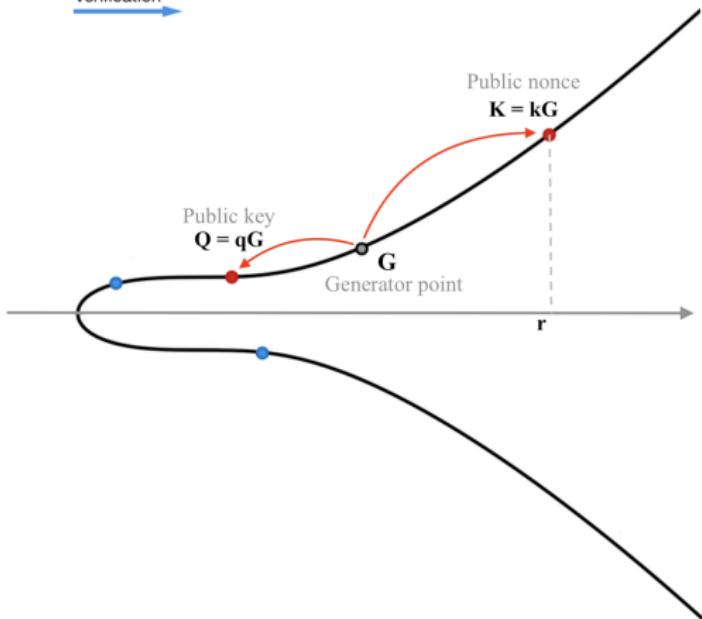
Adapted from:
<https://medium.com/cryptoadvance/how-schnorr-signatures-may-improve-bitcoin-91655bcb4744>

Elliptic curve digital signature algorithm



ECDSA_SIG(m, q):

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



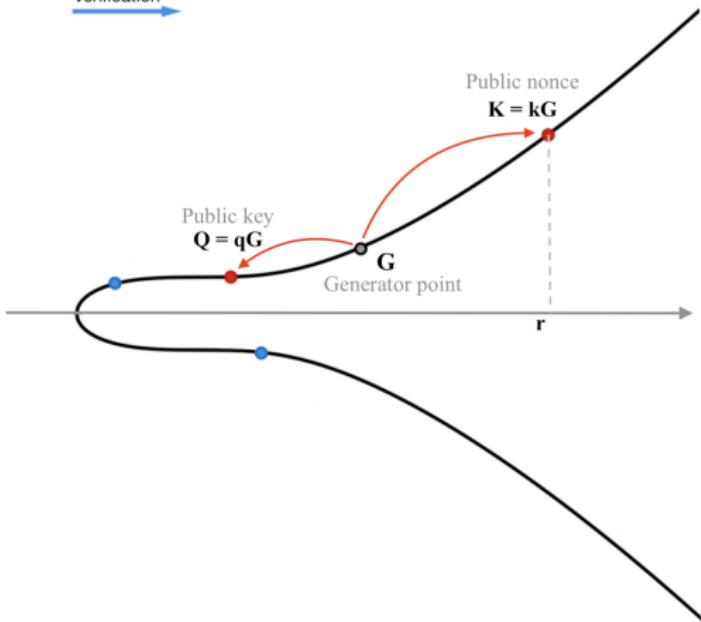
Adapted from:
<https://medium.com/cryptoadvance/how-schnorr-signatures-may-improve-bitcoin-91655bcb4744>

Elliptic curve digital signature algorithm

Signing
→
Verification
→

ECDSA_SIG(m, q):

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

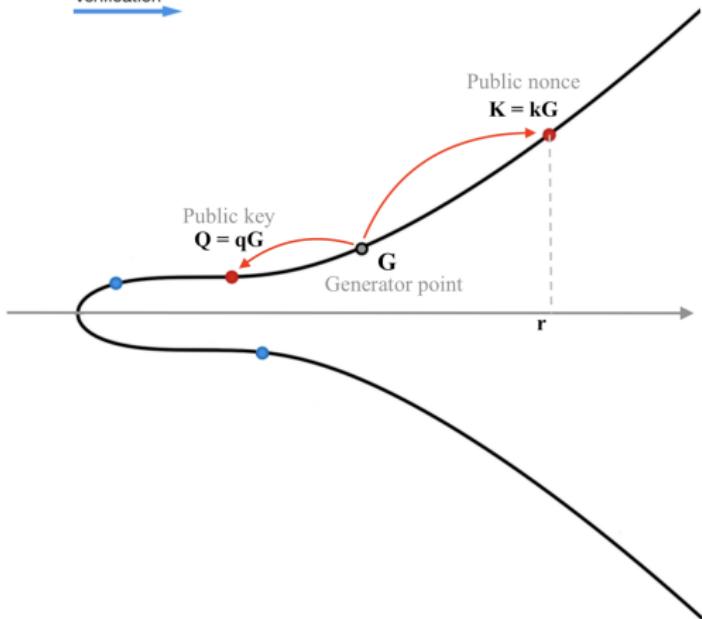


Adapted from:
<https://medium.com/cryptoadvance/how-schnorr-signatures-may-improve-bitcoin-91655bcb4744>

Elliptic curve digital signature algorithm

Signing
→
Verification
→

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



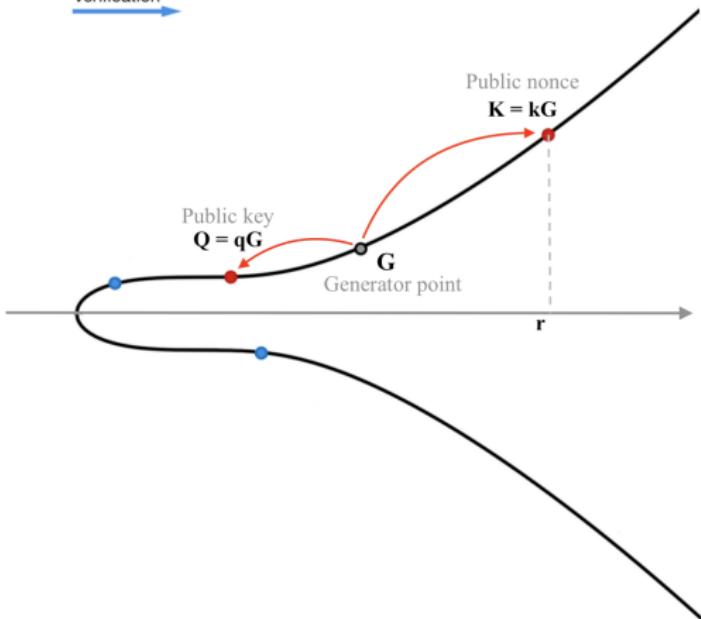
Adapted from:
<https://medium.com/cryptoadvance/how-schnorr-signatures-may-improve-bitcoin-91655bcb4744>

Elliptic curve digital signature algorithm

Signing
Verification

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

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



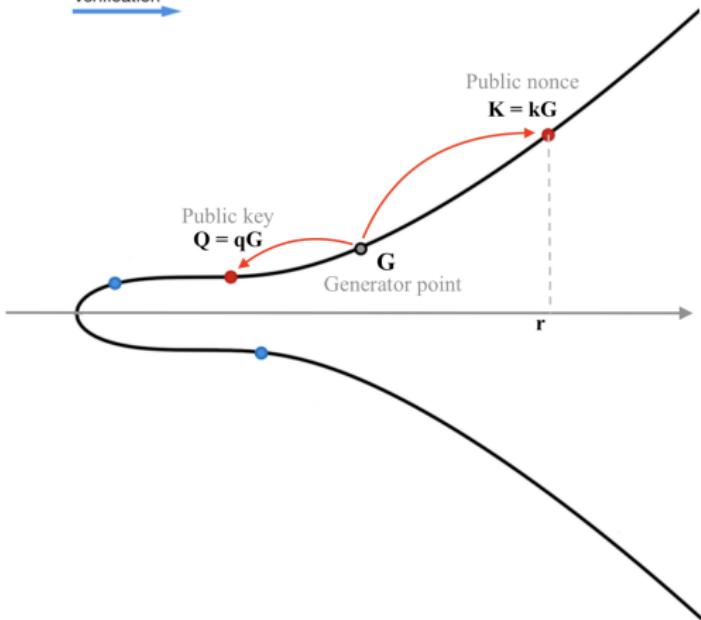
Adapted from:
<https://medium.com/cryptoadvance/how-schnorr-signatures-may-improve-bitcoin-91655bcb4744>

Elliptic curve digital signature algorithm

Signing
Verification

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

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

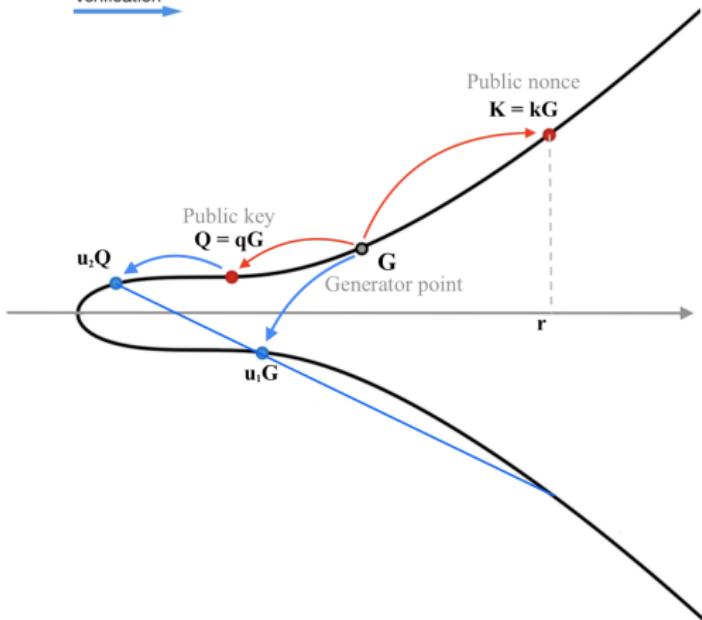
Adapted from:
<https://medium.com/cryptoadvance/how-schnorr-signatures-may-improve-bitcoin-91655bcb4744>

Elliptic curve digital signature algorithm



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

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



Adapted from:

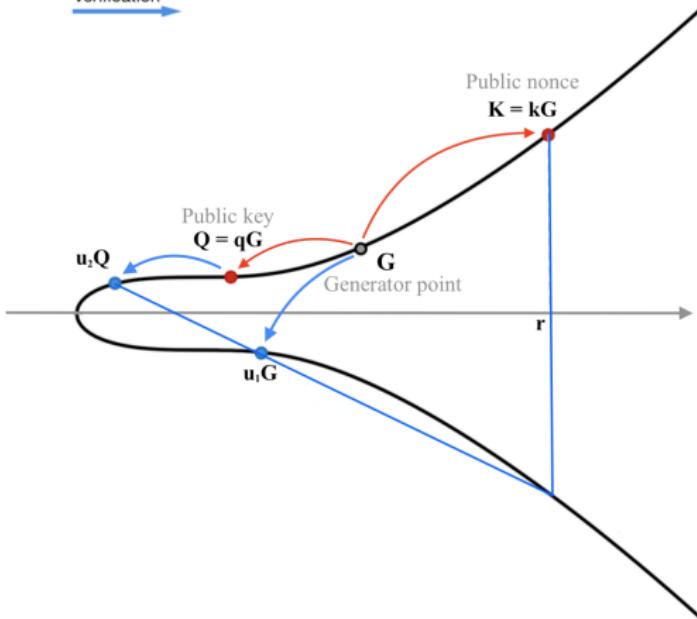
<https://medium.com/cryptoadvance/how-schnorr-signatures-may-improve-bitcoin-91655bcb4744>

Elliptic curve digital signature algorithm



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

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



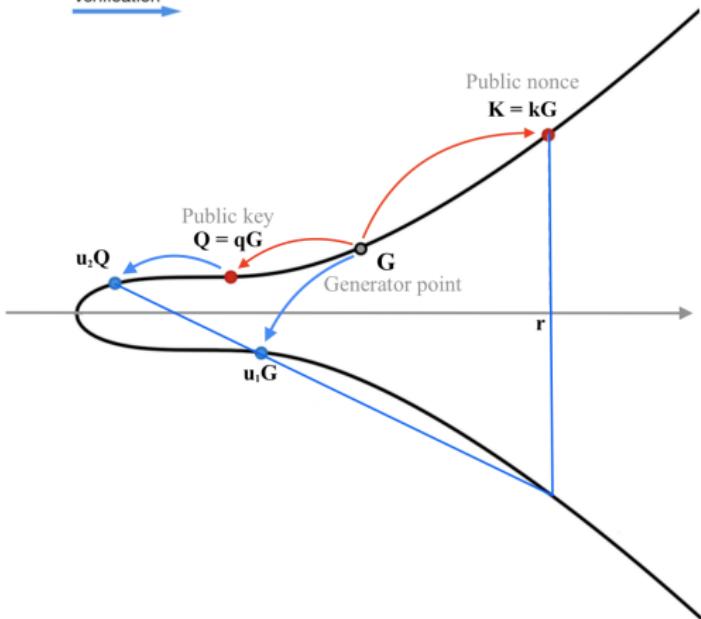
Adapted from:
<https://medium.com/cryptoadvance/how-schnorr-signatures-may-improve-bitcoin-91655bcb4744>

Elliptic curve digital signature algorithm



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

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



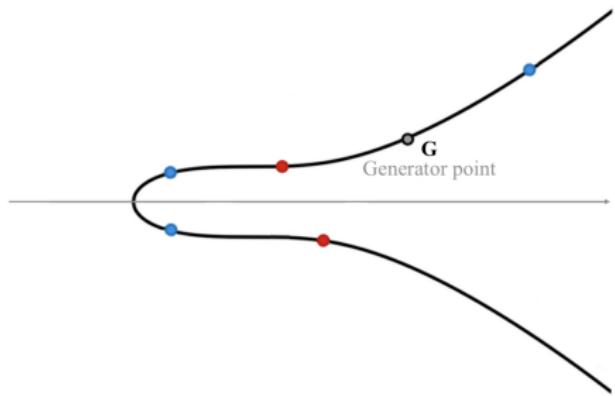
Adapted from:

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

Elliptic curve Schnorr signature algorithm



ECSSA_SIG(m, q):

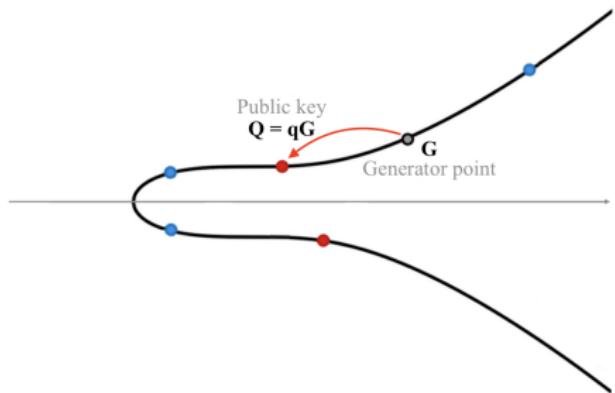


Adapted from:
<https://medium.com/cryptoadvance/how-schnorr-signatures-may-improve-bitcoin-91655bcb4744>

Elliptic curve Schnorr signature algorithm



ECSSA_SIG(m, q):



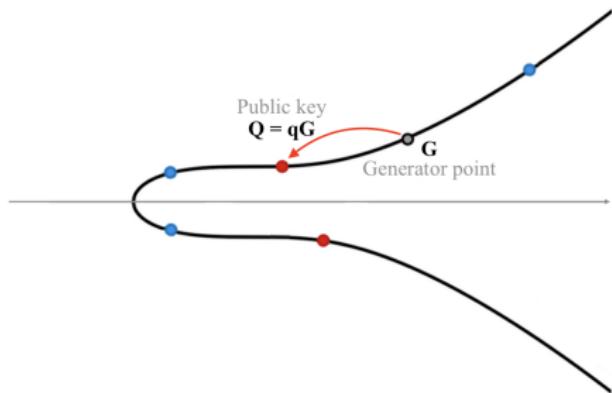
Adapted from:
<https://medium.com/cryptoadvance/how-schnorr-signatures-may-improve-bitcoin-91655bcb4744>

Elliptic curve Schnorr signature algorithm



ECSSA_SIG(m, q):

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



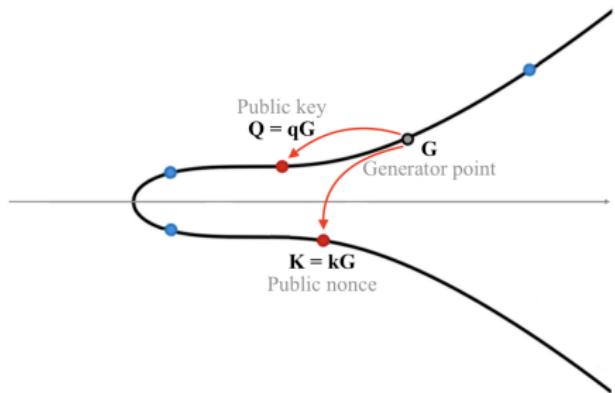
Adapted from:
<https://medium.com/cryptoadvance/how-schnorr-signatures-may-improve-bitcoin-91655bcb4744>

Elliptic curve Schnorr signature algorithm



ECSSA_SIG(m, q):

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



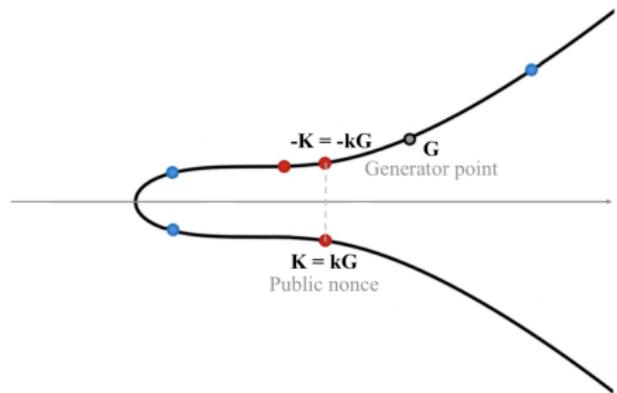
Adapted from:
<https://medium.com/cryptoadvance/how-schnorr-signatures-may-improve-bitcoin-91655bcb4744>

Elliptic curve Schnorr signature algorithm



ECSSA_SIG(m, q):

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



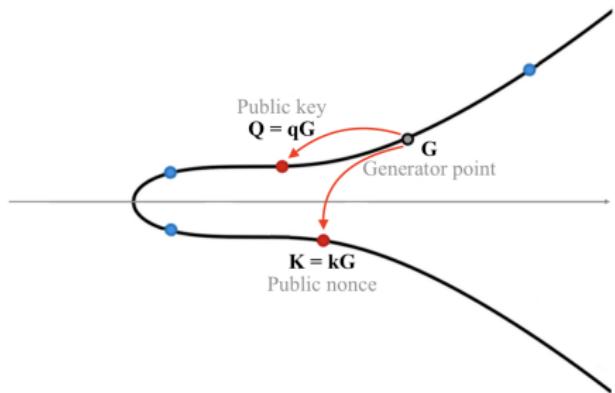
Adapted from:
<https://medium.com/cryptoadvance/how-schnorr-signatures-may-improve-bitcoin-91655bcb4744>

Elliptic curve Schnorr signature algorithm



ECSSA_SIG(m, q):

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



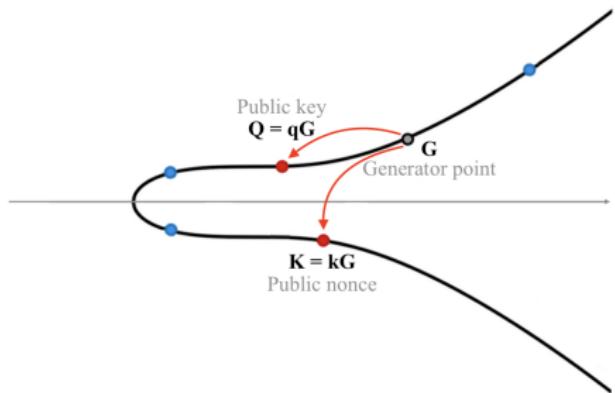
Adapted from:
<https://medium.com/cryptoadvance/how-schnorr-signatures-may-improve-bitcoin-91655bcb4744>

Elliptic curve Schnorr signature algorithm



ECSSA_SIG(m, q):

1. $k \xleftarrow{\$} \{1, \dots, n - 1\}$;
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. $s \leftarrow k + eq \pmod{n}$;



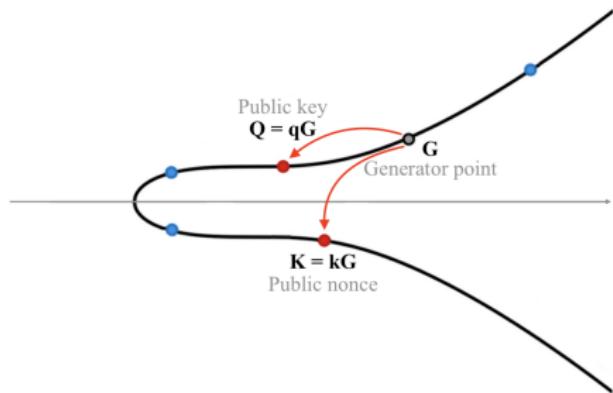
Adapted from:
<https://medium.com/cryptoadvance/how-schnorr-signatures-may-improve-bitcoin-91655bcb4744>

Elliptic curve Schnorr signature algorithm



ECSSA_SIG(m, q):

1. $k \xleftarrow{\$} \{1, \dots, n - 1\}$;
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. $s \leftarrow k + eq \pmod{n}$;
6. **return** (x_K, s) .

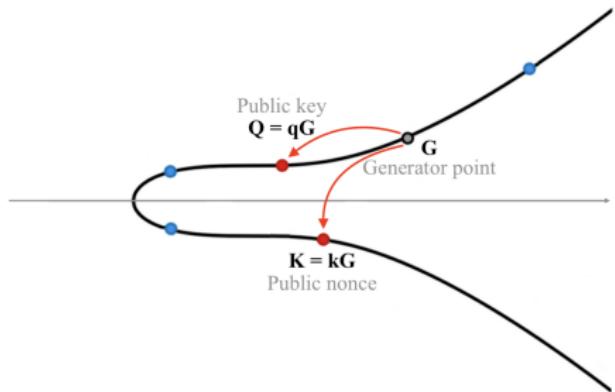


Adapted from:
<https://medium.com/cryptoadvance/how-schnorr-signatures-may-improve-bitcoin-91655bcb4744>

Elliptic curve Schnorr signature algorithm



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



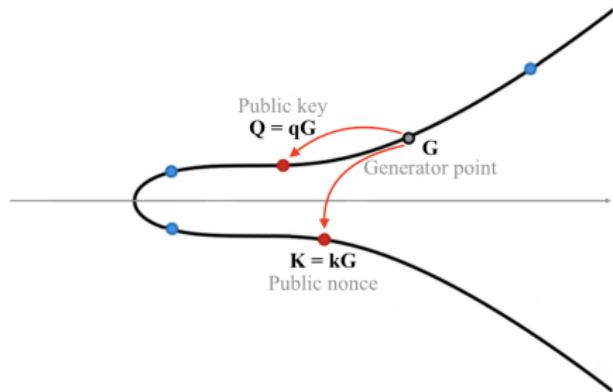
Adapted from:
<https://medium.com/cryptoadvance/how-schnorr-signatures-may-improve-bitcoin-91655bcb4744>

Elliptic curve Schnorr signature algorithm



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

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



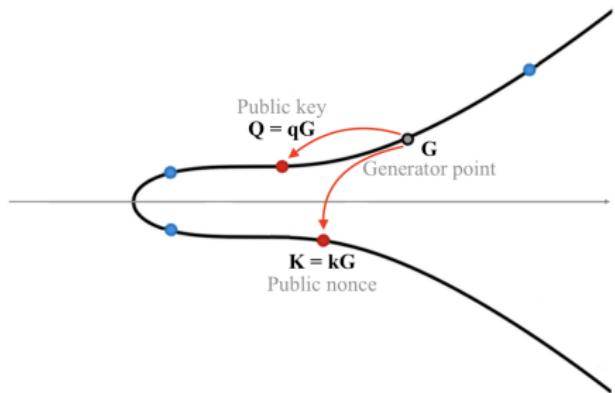
Adapted from:
<https://medium.com/cryptoadvance/how-schnorr-signatures-may-improve-bitcoin-91655bcb4744>

Elliptic curve Schnorr signature algorithm



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

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



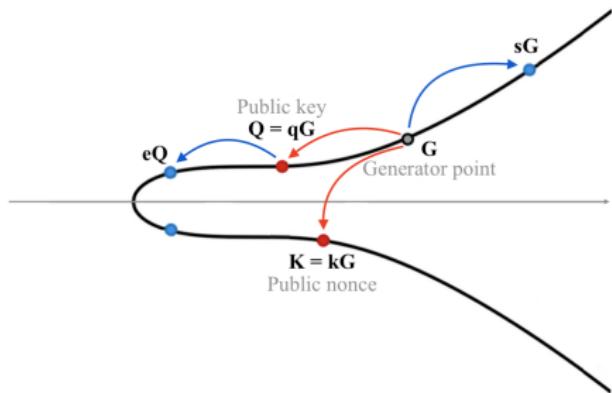
Adapted from:
<https://medium.com/cryptoadvance/how-schnorr-signatures-may-improve-bitcoin-91655bcb4744>

Elliptic curve Schnorr signature algorithm



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

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



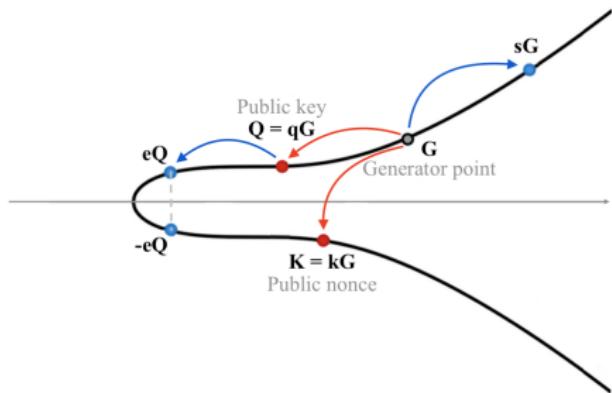
Adapted from:
<https://medium.com/cryptoadvance/how-schnorr-signatures-may-improve-bitcoin-91655bcb4744>

Elliptic curve Schnorr signature algorithm



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

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



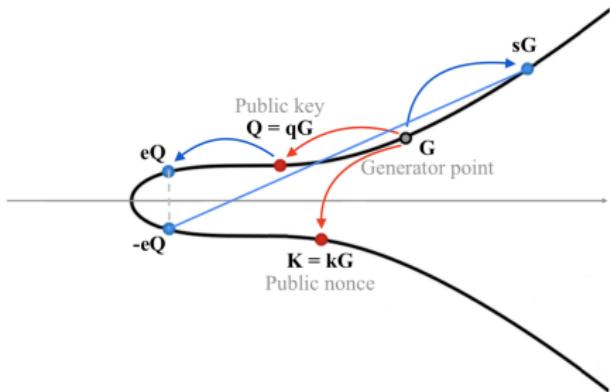
Adapted from:
<https://medium.com/cryptoadvance/how-schnorr-signatures-may-improve-bitcoin-91655bcb4744>

Elliptic curve Schnorr signature algorithm



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

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



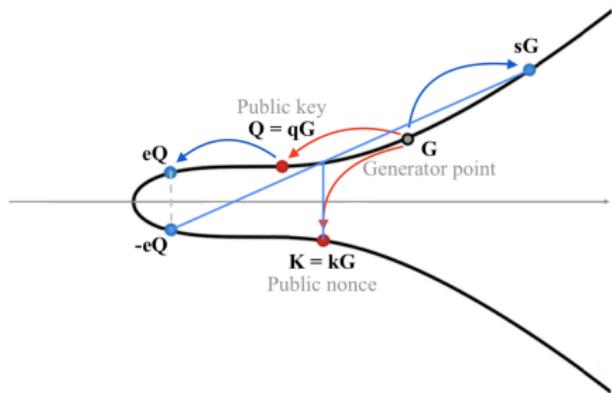
Adapted from:
<https://medium.com/cryptoadvance/how-schnorr-signatures-may-improve-bitcoin-91655bcb4744>

Elliptic curve Schnorr signature algorithm



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

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



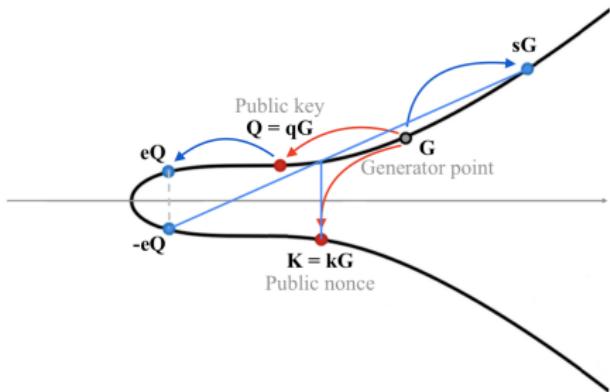
Adapted from:
<https://medium.com/cryptoadvance/how-schnorr-signatures-may-improve-bitcoin-91655bcb4744>

Elliptic curve Schnorr signature algorithm



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

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



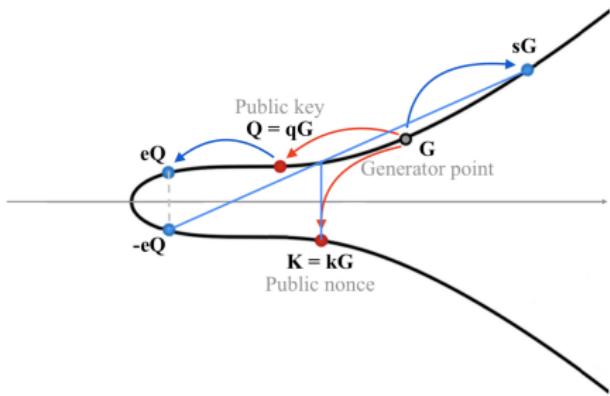
Adapted from:
<https://medium.com/cryptoadvance/how-schnorr-signatures-may-improve-bitcoin-91655bcb4744>

Elliptic curve Schnorr signature algorithm



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

1. **If** $r \notin \{1, \dots, p - 1\}$ or
 $s \notin \{1, \dots, n - 1\}$:
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 \implies 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.

Outline

Mathematical background and cryptographic primitives

Digital signature schemes

ECSSA applications

MuSig

Threshold signature scheme

Adaptor signatures

Conclusion

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: $t <\text{pubKey1}> <\text{pubKey2}> \dots <\text{pubKey}_m>$
 $m \text{ OP_CHECKMULTISIG}$
- ▶ Unlocking script: $0 <\text{sig1}> <\text{sig2}> \dots <\text{sig}_t>$

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: $t <\text{pubKey1}> <\text{pubKey2}> \dots <\text{pubKey}_m>$
 $m \text{ OP_CHECKMULTISIG}$
- ▶ Unlocking script: $0 <\text{sig1}> <\text{sig2}> \dots <\text{sig}_t>$

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: $t <\text{pubKey1}> <\text{pubKey2}> \dots <\text{pubKey}_m>$
 $m \text{ OP_CHECKMULTISIG}$
- ▶ Unlocking script: $0 <\text{sig1}> <\text{sig2}> \dots <\text{sig}_t>$

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

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



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. $Q \leftarrow \sum_{i=1}^m a_i Q_i;$



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. $k_1 \xleftarrow{\$} \{1, \dots, n - 1\};$



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. $k_1 \xleftarrow{\$} \{1, \dots, n - 1\}$;
4. $K_1 \leftarrow k_1 G$, $t_1 \leftarrow \text{hash}(K_1)$;



1: Alice



2: Bob

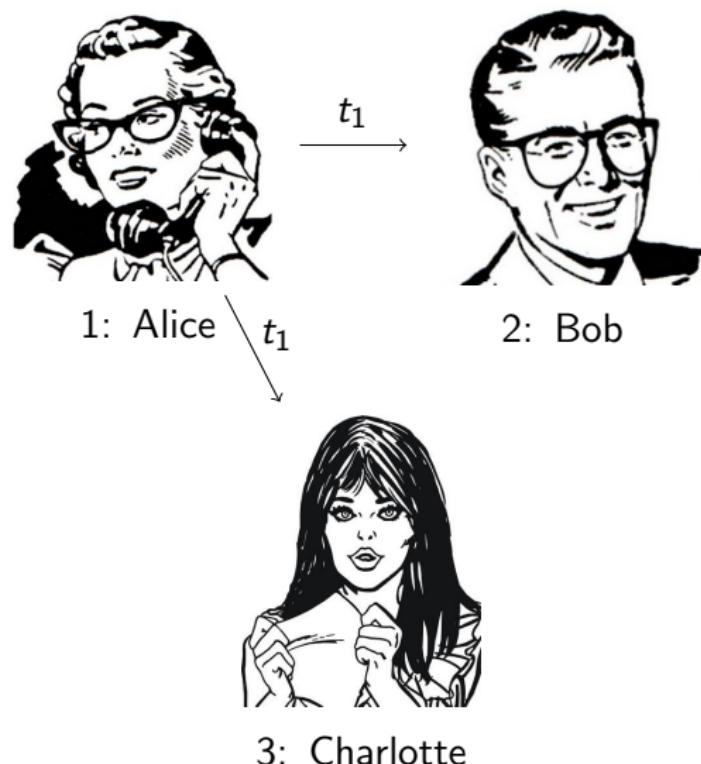


3: Charlotte

MuSig: compact m -of- m signature scheme

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

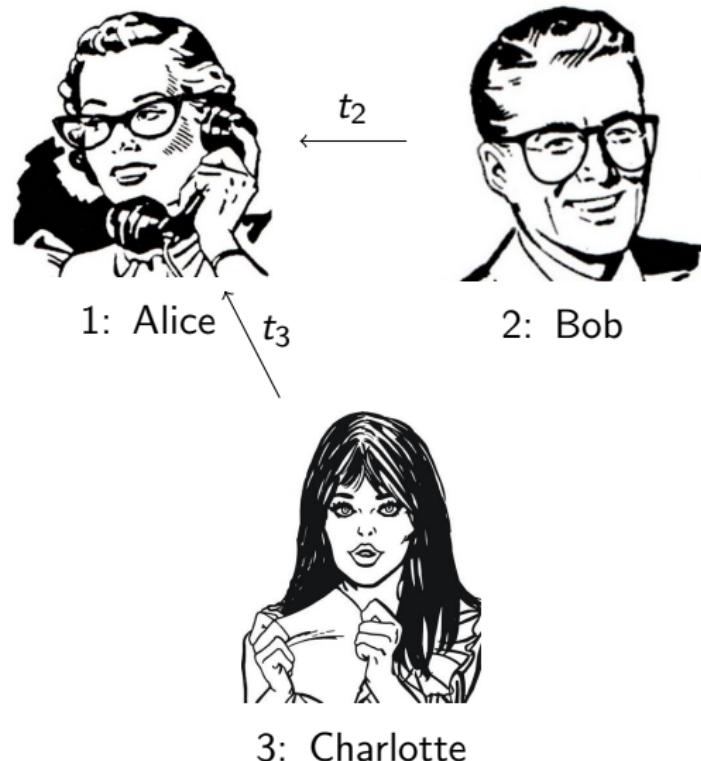
1. **for** $i \leftarrow 1, m$ **do**:
 - 1.1 $a_i \leftarrow \text{hash}(\langle L \rangle || Q_i)$;
2. $Q \leftarrow \sum_{i=1}^m a_i Q_i$;
3. $k_1 \xleftarrow{\$} \{1, \dots, n - 1\}$;
4. $K_1 \leftarrow k_1 G$, $t_1 \leftarrow \text{hash}(K_1)$;
5. **send** t_1, K_1 ;



MuSig: compact m -of- m signature scheme

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

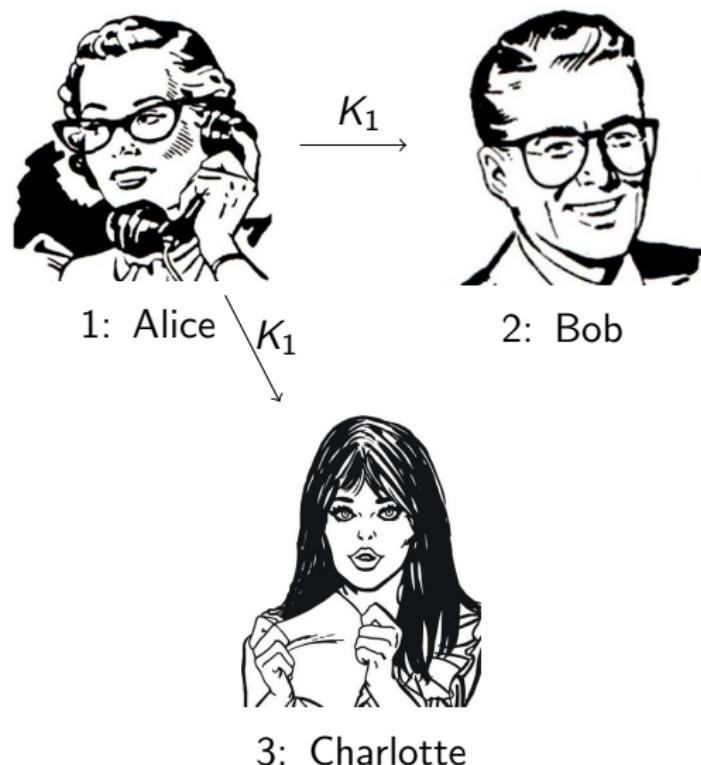
1. **for** $i \leftarrow 1, m$ **do**:
 - 1.1 $a_i \leftarrow \text{hash}(\langle L \rangle || Q_i)$;
2. $Q \leftarrow \sum_{i=1}^m a_i Q_i$;
3. $k_1 \xleftarrow{\$} \{1, \dots, n - 1\}$;
4. $K_1 \leftarrow k_1 G$, $t_1 \leftarrow \text{hash}(K_1)$;
5. **send** t_1, K_1 ;



MuSig: compact m -of- m signature scheme

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

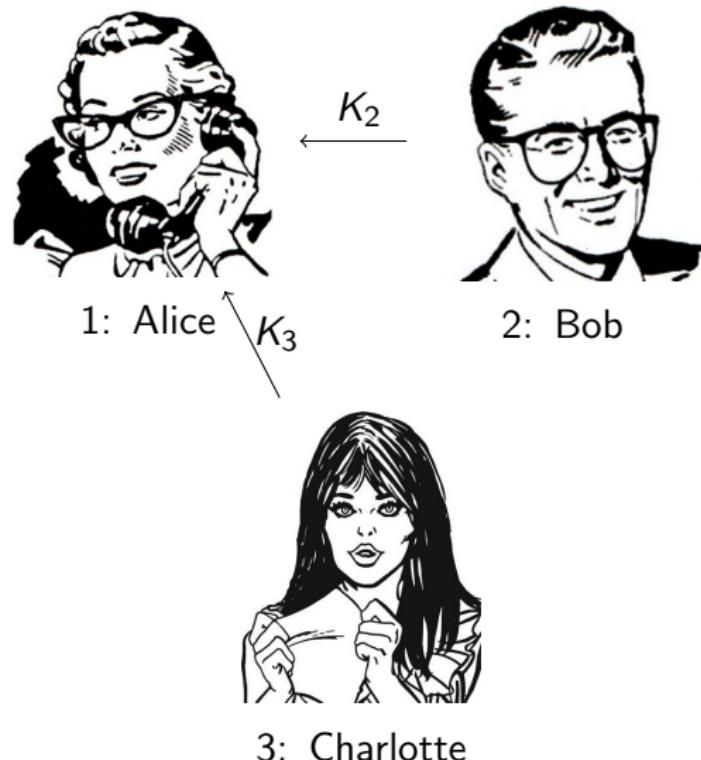
1. **for** $i \leftarrow 1, m$ **do**:
 - 1.1 $a_i \leftarrow \text{hash}(\langle L \rangle || Q_i)$;
2. $Q \leftarrow \sum_{i=1}^m a_i Q_i$;
3. $k_1 \xleftarrow{\$} \{1, \dots, n - 1\}$;
4. $K_1 \leftarrow k_1 G, t_1 \leftarrow \text{hash}(K_1)$;
5. **send** t_1, K_1 ;



MuSig: compact m -of- m signature scheme

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

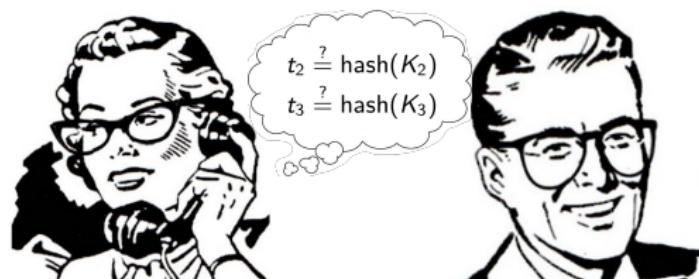
1. **for** $i \leftarrow 1, m$ **do**:
 - 1.1 $a_i \leftarrow \text{hash}(\langle L \rangle || Q_i);$
2. $Q \leftarrow \sum_{i=1}^m a_i Q_i;$
3. $k_1 \xleftarrow{\$} \{1, \dots, n - 1\};$
4. $K_1 \leftarrow k_1 G, t_1 \leftarrow \text{hash}(K_1);$
5. **send** $t_1, K_1;$



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



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. $k_1 \xleftarrow{\$} \{1, \dots, n - 1\};$
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. $k_1 \xleftarrow{\$} \{1, \dots, n - 1\}$;
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

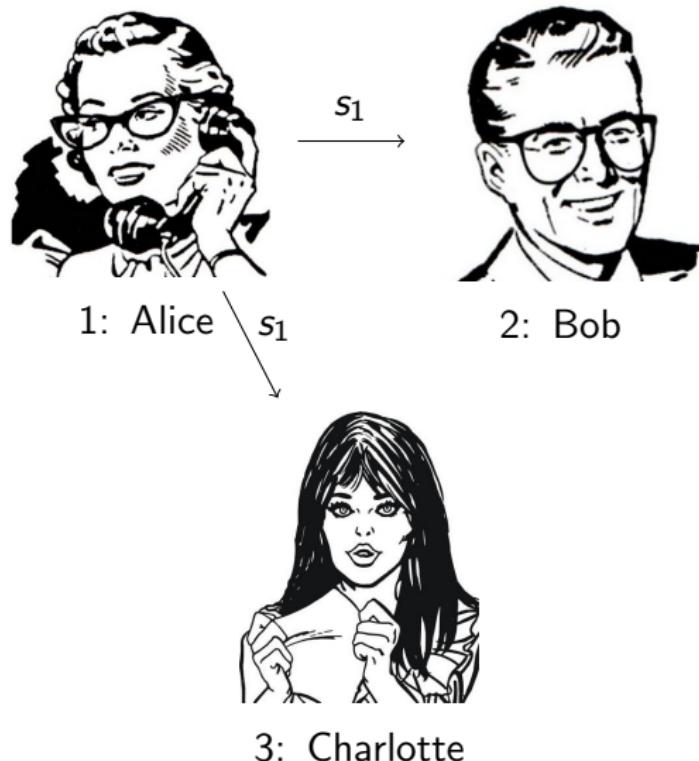


3: Charlotte

MuSig: compact m -of- m signature scheme

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

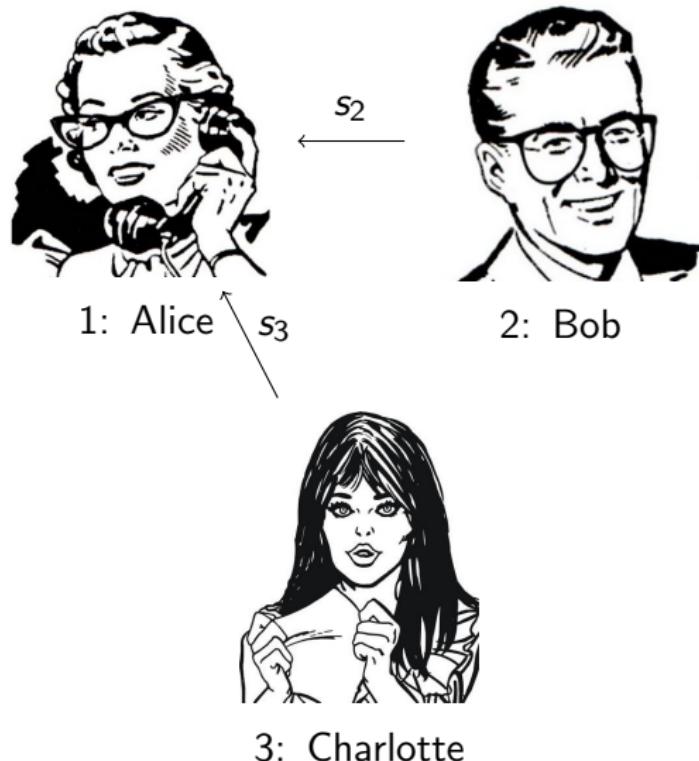
1. **for** $i \leftarrow 1, m$ **do**:
 - 1.1 $a_i \leftarrow \text{hash}(\langle L \rangle || Q_i)$;
2. $Q \leftarrow \sum_{i=1}^m a_i Q_i$;
3. $k_1 \xleftarrow{\$} \{1, \dots, n - 1\}$;
4. $K_1 \leftarrow k_1 G$, $t_1 \leftarrow \text{hash}(K_1)$;
5. **send** t_1, K_1 ;
6. $K \leftarrow \sum_{i=1}^m K_i$;
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. $k_1 \xleftarrow{\$} \{1, \dots, n - 1\}$;
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. $k_1 \xleftarrow{\$} \{1, \dots, n - 1\}$;
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. $k_1 \xleftarrow{\$} \{1, \dots, n - 1\}$;
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. $s_1 \leftarrow k_1 + ca_1 q_1 \pmod{n}$;
9. **send** s_1 ;
10. $s \leftarrow \sum_{i=1}^m s_i \pmod{n}$;
11. **return** (x_K, s) .



1: Alice



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

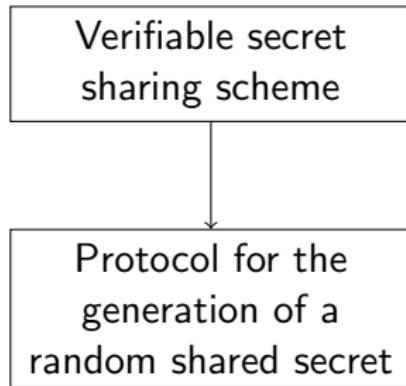


Bob



Charlotte

Threshold signature scheme (t -of- m)



Alice



Protocol for the generation of a random shared secret

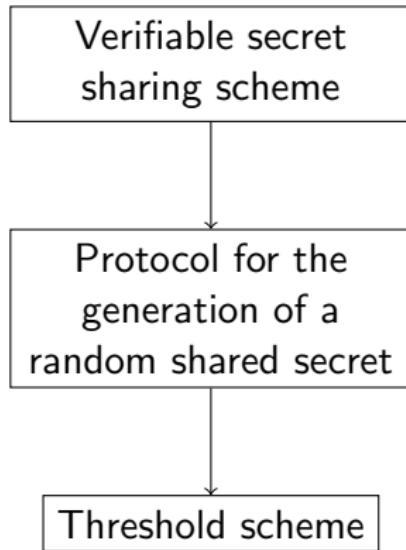
Bob



Charlotte



Threshold signature scheme (t -of- m)



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
 $s' \xleftarrow{\$} \{1, \dots, n - 1\}$;

Dealer: Alice



1: Bob



2: Charlotte

Threshold signature scheme (t -of- m)

Verifiable secret sharing scheme

The dealer:

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

Dealer: Alice



C_0 C_0



1: Bob



2: Charlotte

Threshold signature scheme (t -of- m)

Verifiable secret
sharing scheme

The dealer:

- chooses random polynomials:

$$f(u) = s + f_1 u + \dots + f_{t-1} u^{t-1},$$
$$f'(u) = s' + f'_1 u + \dots + f'_{t-1} u^{t-1},$$

$$f_j, f'_j \xleftarrow{\$} \{1, \dots, n-1\};$$

Dealer: Alice



1: Bob



2: Charlotte

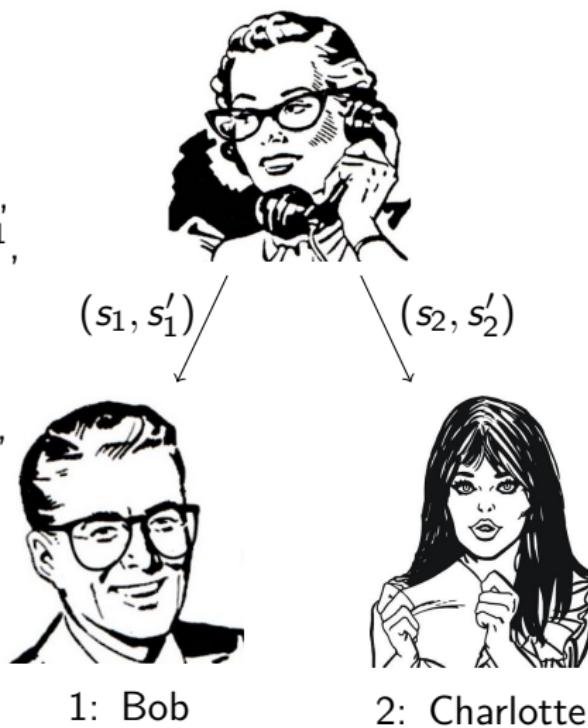
Threshold signature scheme (t -of- m)

Verifiable secret sharing scheme

The dealer:

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

Dealer: Alice



Threshold signature scheme (t -of- m)

Verifiable secret sharing scheme

The dealer:

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

Dealer: Alice



C_j



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:

$$f(u) = \sum_i f(i)\omega_i(u), \text{ where}$$

$$\omega_i(u) = \prod_{j \neq i} \frac{u-j}{i-j} \pmod{n}.$$

$$s = f(0) = \sum_i s_i \omega_i, \text{ with}$$

$$\omega_i = \omega_i(0) = \prod_{j \neq i} \frac{j}{j-i} \pmod{n}.$$

Dealer: Alice



$$\longleftrightarrow$$



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



r_3, f_3

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



r_3, f_3

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
 $(f_i(u) = \sum_{j=0}^{t-1} a_{ij} u^j, a_{i0} = r_i);$
- ▶ 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



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
 $(f_i(u) = \sum_{j=0}^{t-1} a_{ij} u^j, a_{i0} = r_i);$
- ▶ 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 $R_j = r_j G$ ($R = \sum_{j=1}^m R_j = \sum_{j=1}^m r_j G = rG$).

1: Alice



2: Bob



3: Charlotte



Threshold signature scheme (t -of- m)

Threshold scheme

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

1: Alice



3: Charlotte

Threshold signature scheme (t -of- m)

Threshold scheme

After having established a distributed key pair $(\alpha_1, \dots, \alpha_m) \xleftarrow{(t, m)} (q|Q)$ 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:
 $(\beta_1, \dots, \beta_m) \xleftarrow{(t, m)} (k|K).$

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
 $\beta_i = n - \beta_i$;

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

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



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 $\sigma = k + eq \pmod{n}$;
- ▶ the signature is (x_K, σ) .

1: Alice



3: Charlotte

ECDSA vs. ECSSA (multi-signature)

ECDSA:

- ▶ Locking script: $t <\text{pubKey1}> <\text{pubKey2}> \dots <\text{pubKeym}>$
m OP_CHECKMULTISIG
- ▶ Unlocking script: $0 <\text{sig1}> <\text{sig2}> \dots <\text{sigt}>$

ECSSA:

- ▶ Locking script: $<\text{jointPubKey}>$ OP_SCHNORR
- ▶ Unlocking script: $<\text{jointSig}>$

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 $T = tG$ but still consider k as private nonce: this results in an invalid signature, however learning t is equivalent to learning a valid signature:

$$(x_T, x_{K+T}, s'), \quad s' = k + \text{hash}(x_{K+T} || Q || m).$$

$$\text{Checking equation: } s'G = K + \text{hash}(x_{K+T} || Q || m)Q.$$

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.

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 address>

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 address>

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

 $Q_A^{\mathbb{B}}, Q_A^{\Xi}$ 

Alice

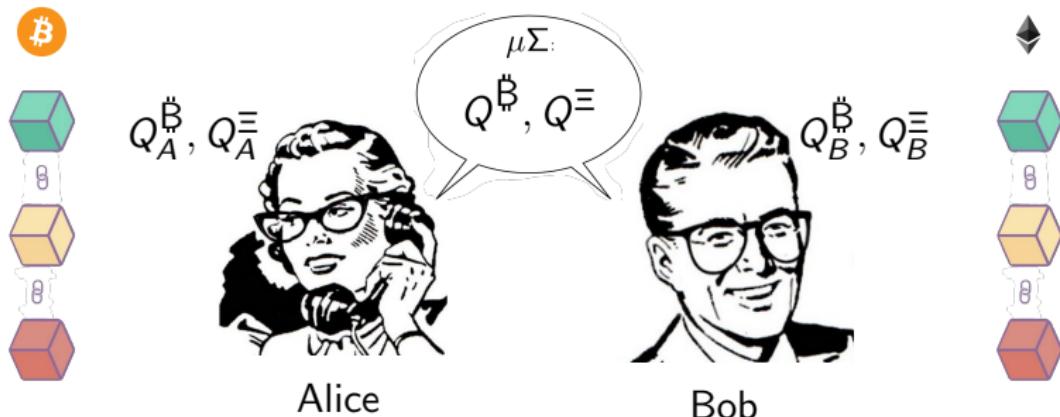
 $Q_B^{\mathbb{B}}, Q_B^{\Xi}$ 

Bob

The participants have a key pair for each chain:

 $\{q_A^{\mathbb{B}}, Q_A^{\mathbb{B}}\}, \{q_A^{\Xi}, Q_A^{\Xi}\}, \{q_B^{\mathbb{B}}, Q_B^{\mathbb{B}}\}, \{q_B^{\Xi}, Q_B^{\Xi}\}.$

Cross-chain atomic swaps via adaptor signatures



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

$$Q^{\mathbb{B}} = \text{hash}(\langle Q_A^{\mathbb{B}}, Q_B^{\mathbb{B}} \rangle || Q_A^{\mathbb{B}}) Q_A^{\mathbb{B}} + \text{hash}(\langle Q_A^{\mathbb{B}}, Q_B^{\mathbb{B}} \rangle || Q_B^{\mathbb{B}}) Q_B^{\mathbb{B}},$$
$$Q^{\Xi} = \text{hash}(\langle Q_A^{\Xi}, Q_B^{\Xi} \rangle || Q_A^{\Xi}) Q_A^{\Xi} + \text{hash}(\langle Q_A^{\Xi}, Q_B^{\Xi} \rangle || Q_B^{\Xi}) Q_B^{\Xi}.$$

Cross-chain atomic swaps via adaptor signatures


$$Q_A^{\mathbb{B}}, Q_A^{\Xi}$$
$$Q_B^{\mathbb{B}}, Q_B^{\Xi}$$


Alice


$$Q_B^{\mathbb{B}}, Q_B^{\Xi}$$
$$Q_A^{\mathbb{B}}, Q_A^{\Xi}$$

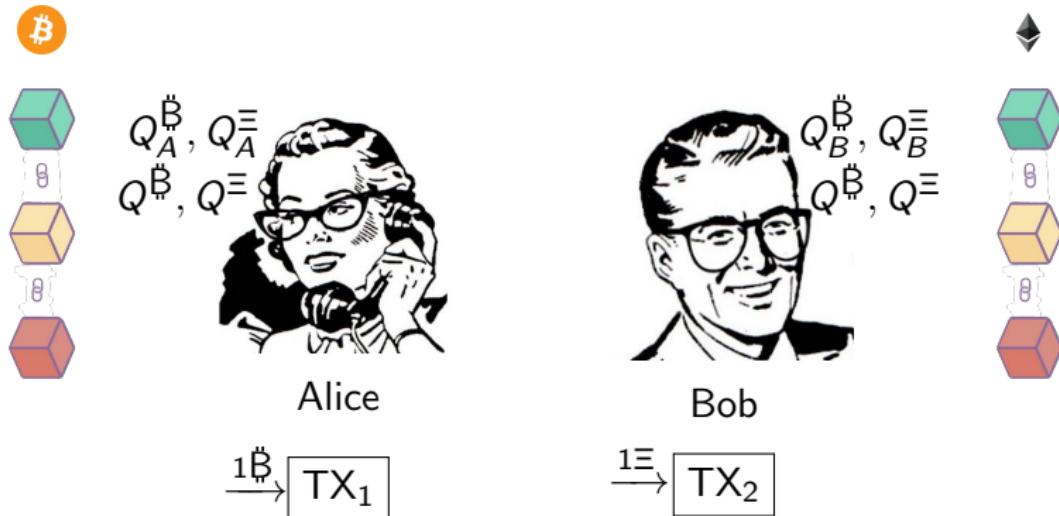

Bob

TX₁

TX₂

They start constructing transactions on the respective chain:

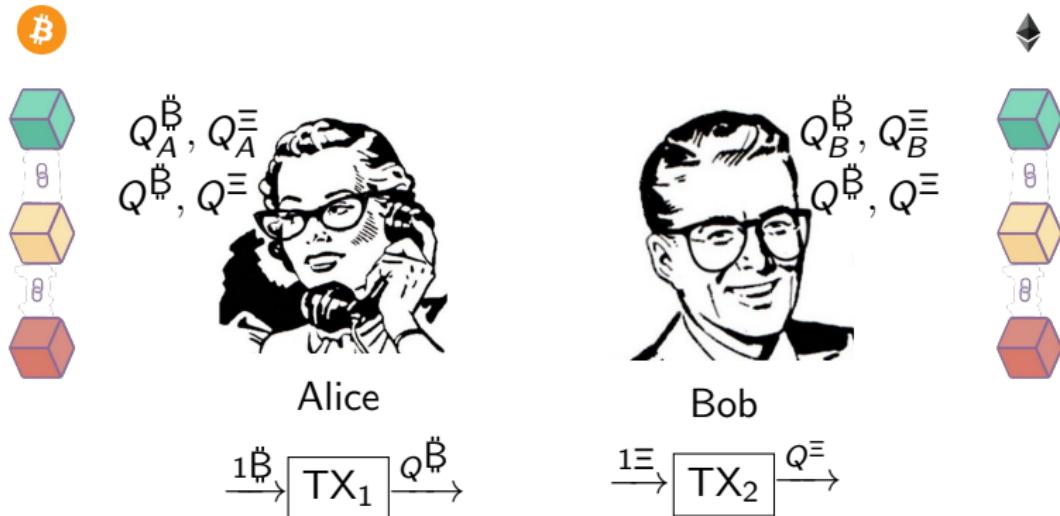
Cross-chain atomic swaps via adaptor signatures



They start constructing transactions on the respective chain:

- ▶ They agree on the exchange rate;

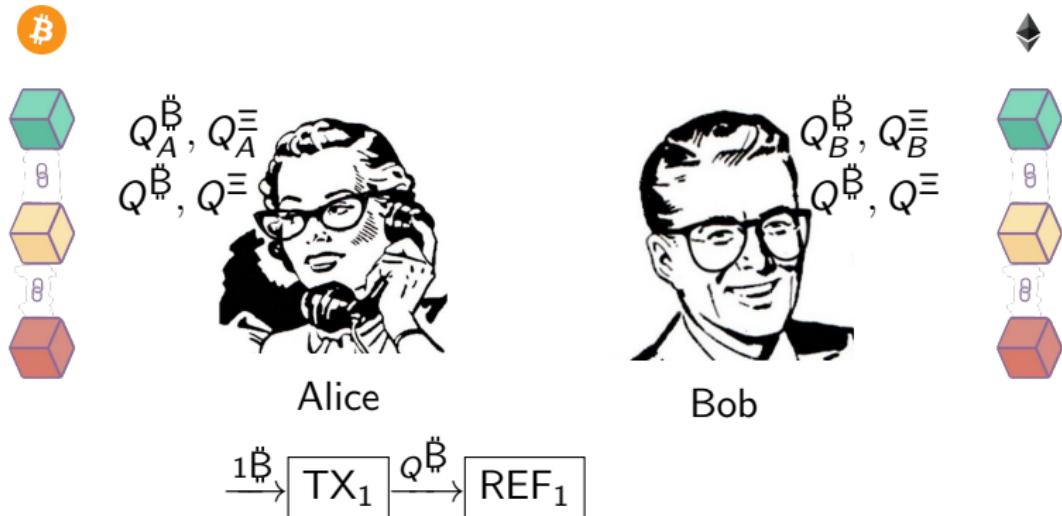
Cross-chain atomic swaps via adaptor signatures



They start constructing transactions on the respective chain:

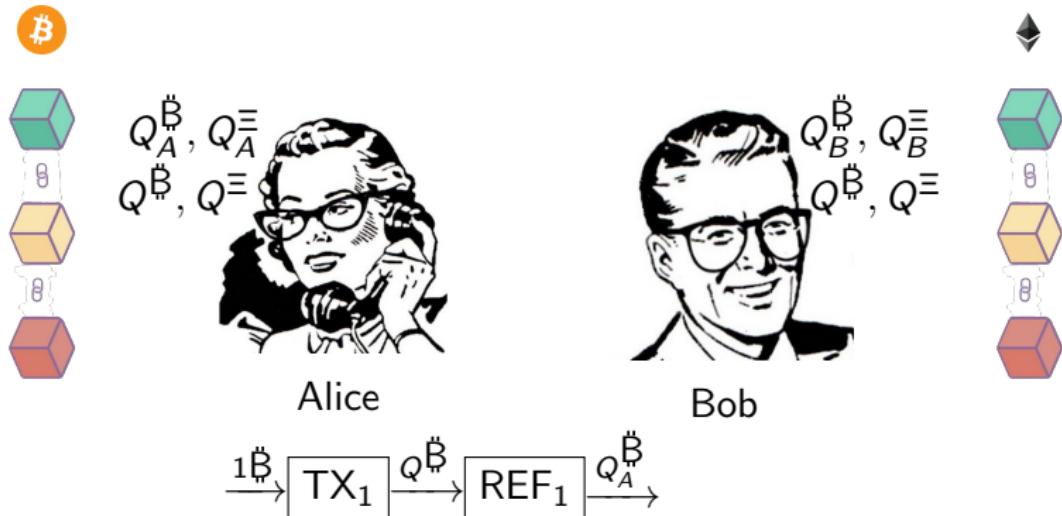
- ▶ They agree on the exchange rate;
- ▶ The funds are locked with the aggregated public keys.

Cross-chain atomic swaps via adaptor signatures



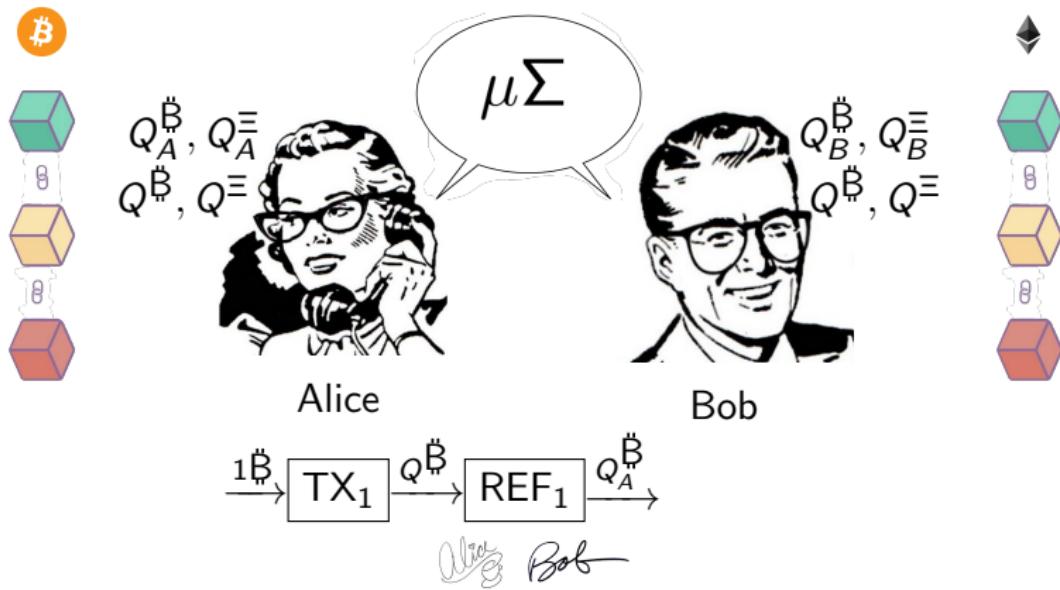
Before signing, they construct a refund transaction (here Alice's example). Both transactions have to be time locked and the time lock over Bob's transaction has to be greater.

Cross-chain atomic swaps via adaptor signatures



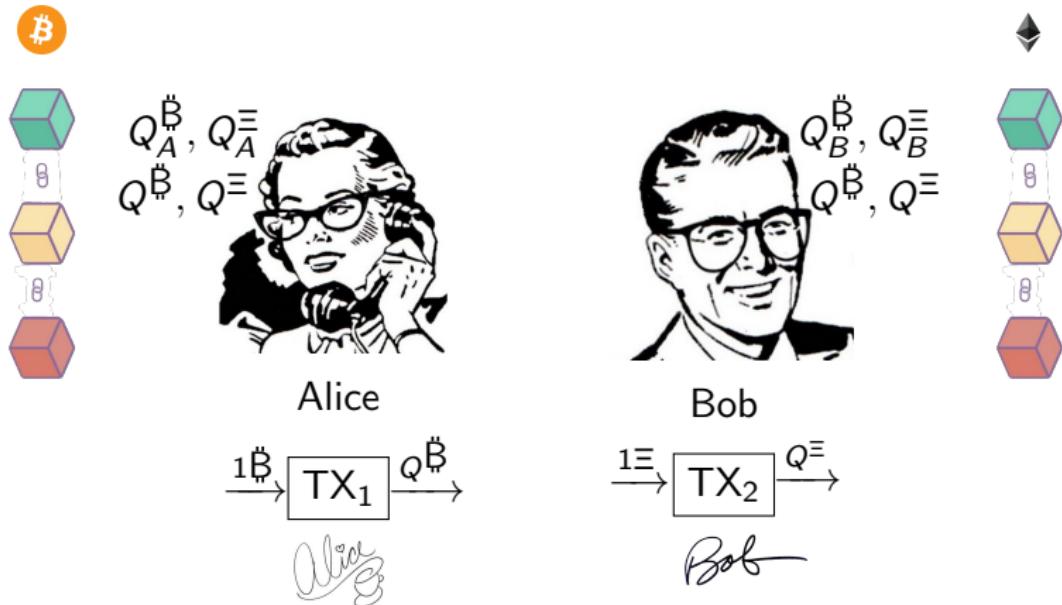
Before signing, they construct a refund transaction (here Alice's example). Both transactions have to be time locked and the time lock over Bob's transaction has to be greater.

Cross-chain atomic swaps via adaptor signatures



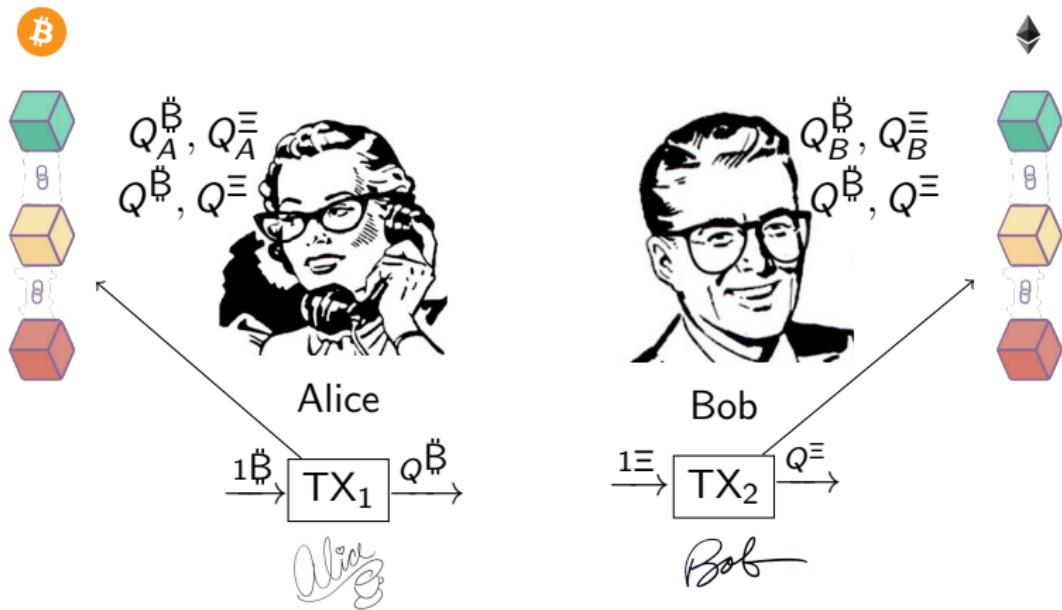
Before signing, they construct a refund transaction (here Alice's example). Both transactions have to be time locked and the time lock over Bob's transaction has to be greater. To sign it, the participants have to interact via the MuSig protocol.

Cross-chain atomic swaps via adaptor signatures



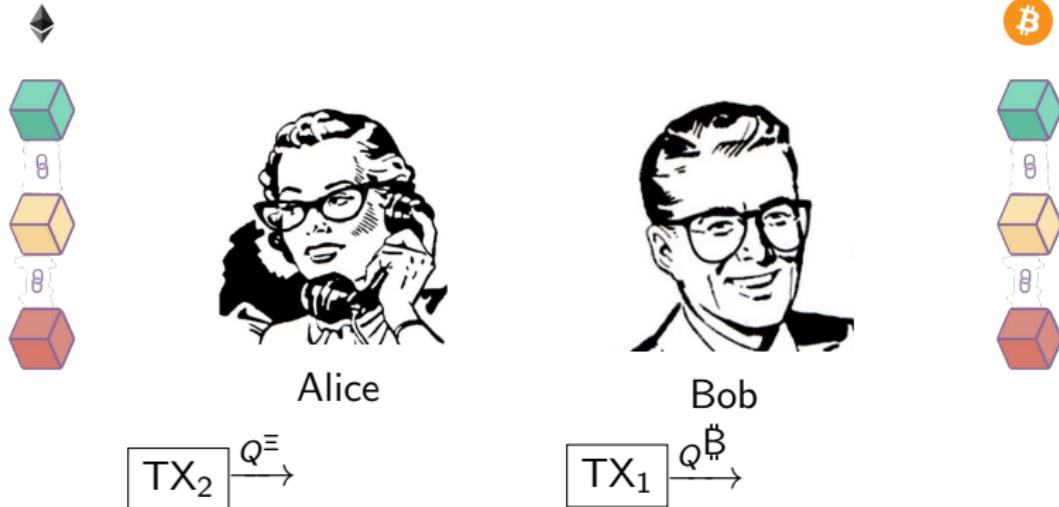
Having the refund transactions, Alice and Bob can confidently sign TX_1 and TX_2 .

Cross-chain atomic swaps via adaptor signatures



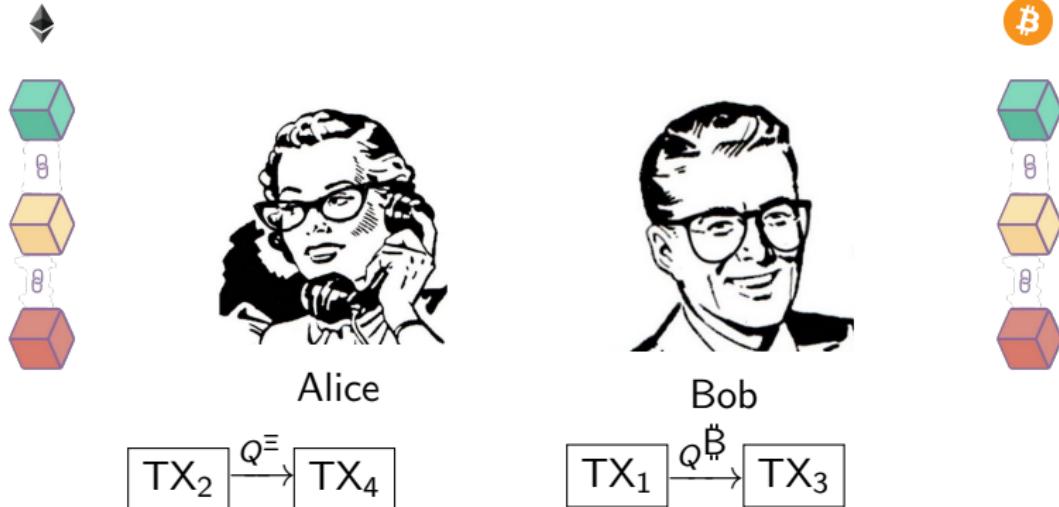
Having the refund transactions, Alice and Bob can confidently sign TX_1 and TX_2 . Then they broadcast them and wait for confirmation.

Cross-chain atomic swaps via adaptor signatures



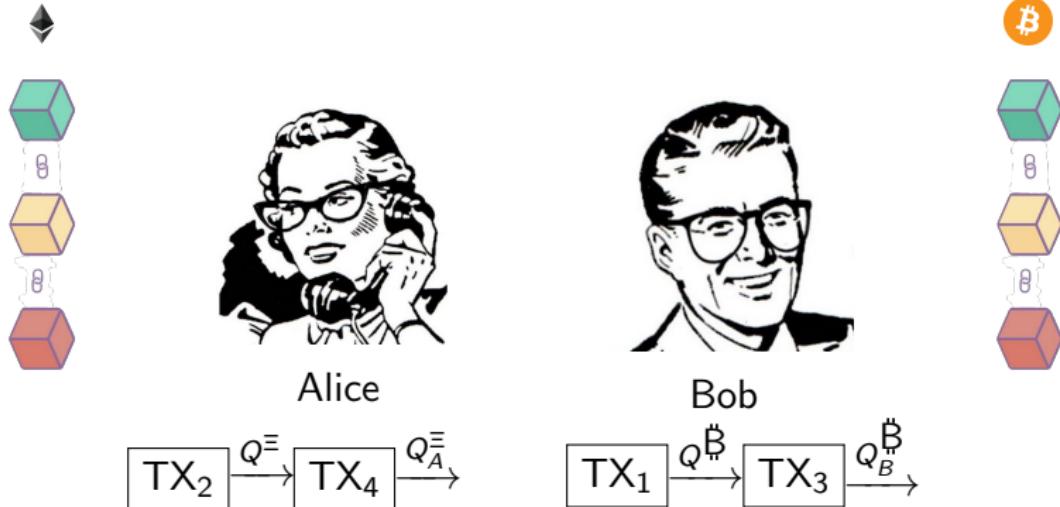
At this point they construct transactions to grab the other coin:

Cross-chain atomic swaps via adaptor signatures



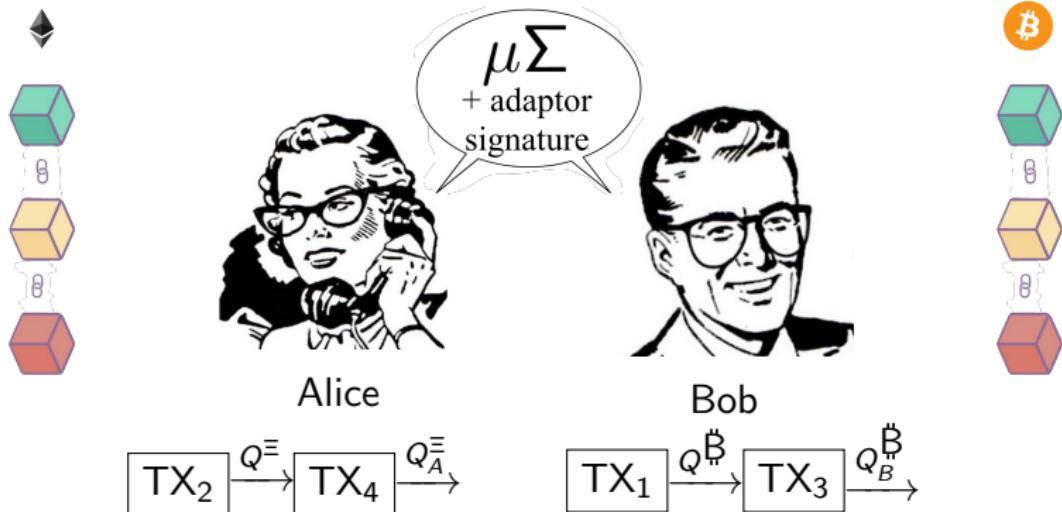
At this point they construct transactions to grab the other coin:

Cross-chain atomic swaps via adaptor signatures



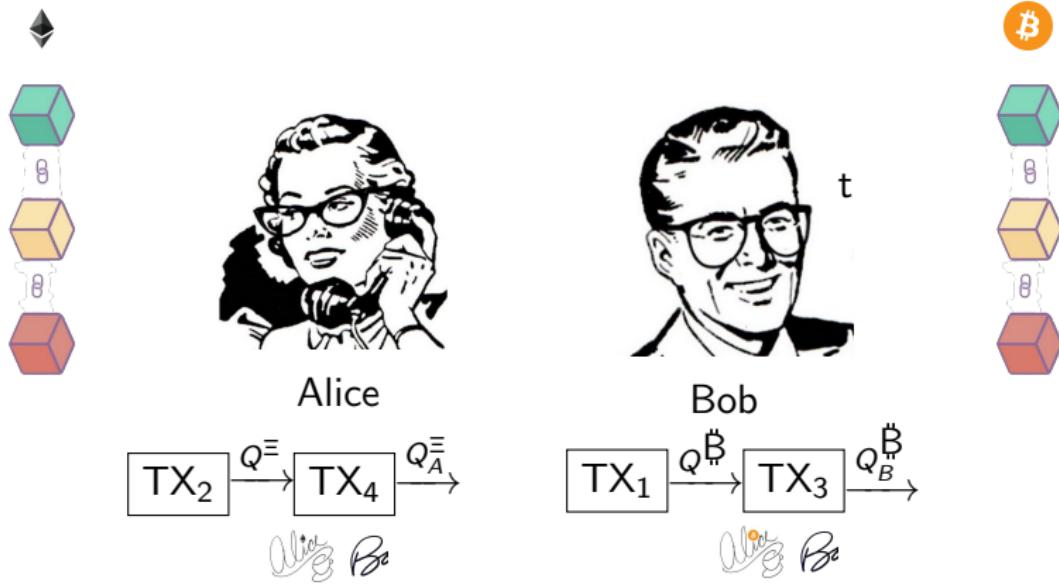
At this point they construct transactions to grab the other coin:

Cross-chain atomic swaps via adaptor signatures



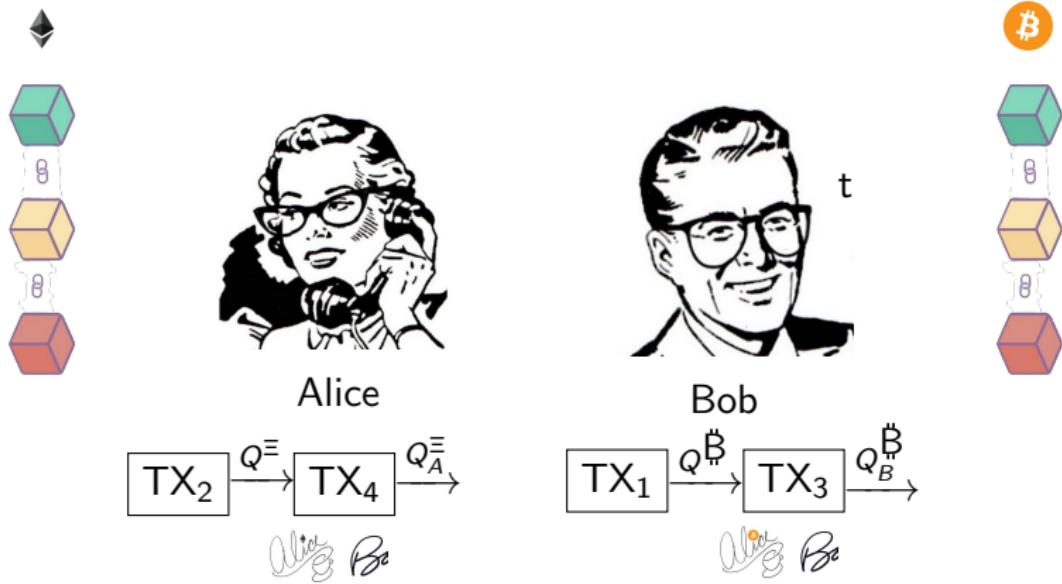
At this point they construct transactions to grab the other coin: they need to resort to the MuSig protocol to generate valid signatures; to ensure atomicity, Bob produces two adaptor signatures using the same secret value t .

Cross-chain atomic swaps via adaptor signatures



Alice 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

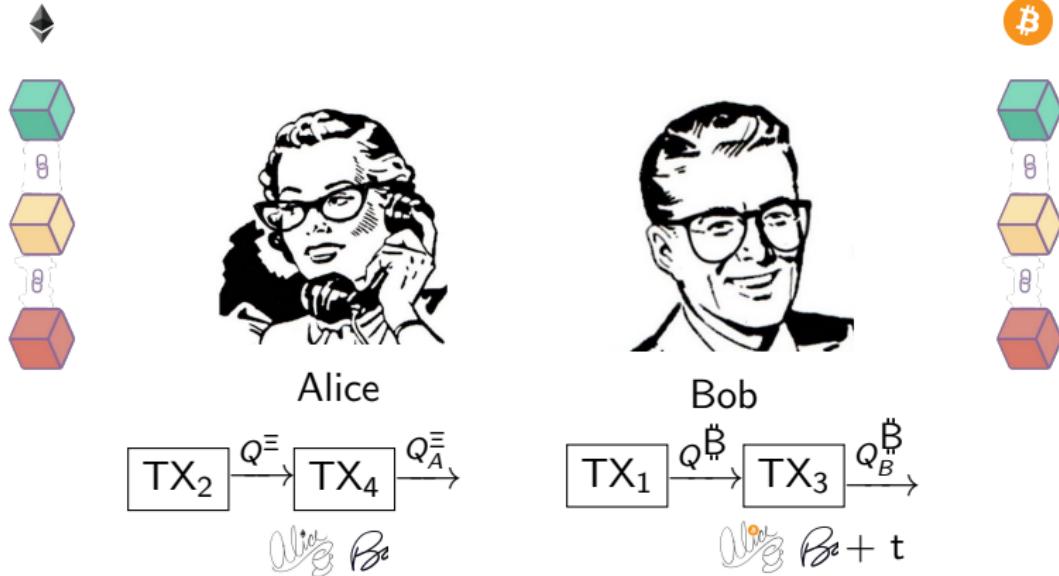


This results in two partial (invalid) signatures: $(x_{K_A^\Xi + K_B^\Xi + T}, s'^\Xi)$
and $(x_{K_A^\Xi + K_B^\Xi + T}, s'^\Xi)$, with

$$s'^\Xi = k_A^\Xi + k_B^\Xi + \text{hash}(x_{K_A^\Xi + K_B^\Xi + T} || Q^\Xi || TX_4)(q_A^\Xi + q_B^\Xi) \text{ and}$$

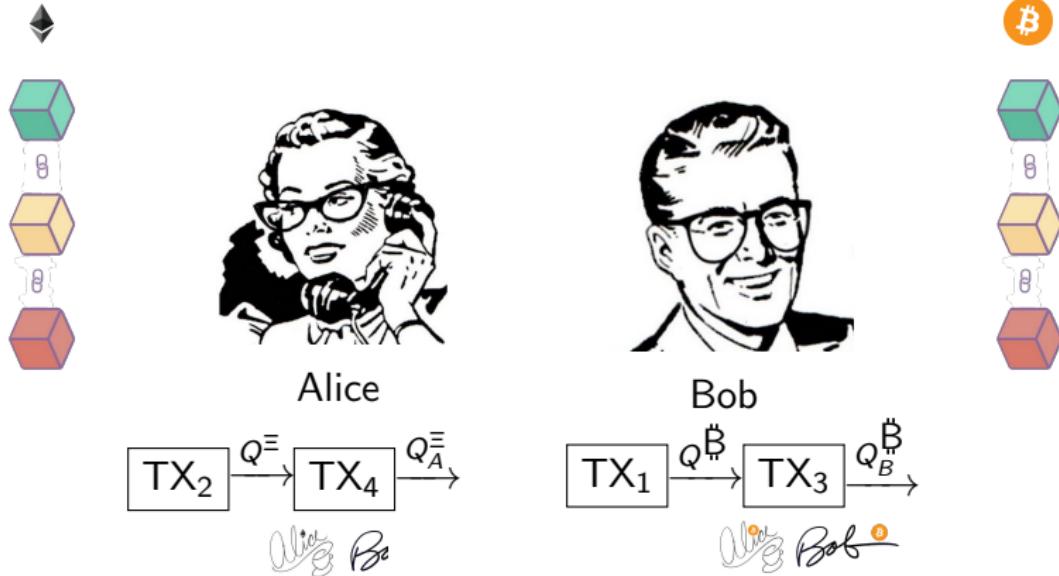
$$s'^\Xi = k_A^\Xi + k_B^\Xi + \text{hash}(x_{K_A^\Xi + K_B^\Xi + T} || Q^\Xi || TX_4)(q_A^\Xi + q_B^\Xi).$$

Cross-chain atomic swaps via adaptor signatures



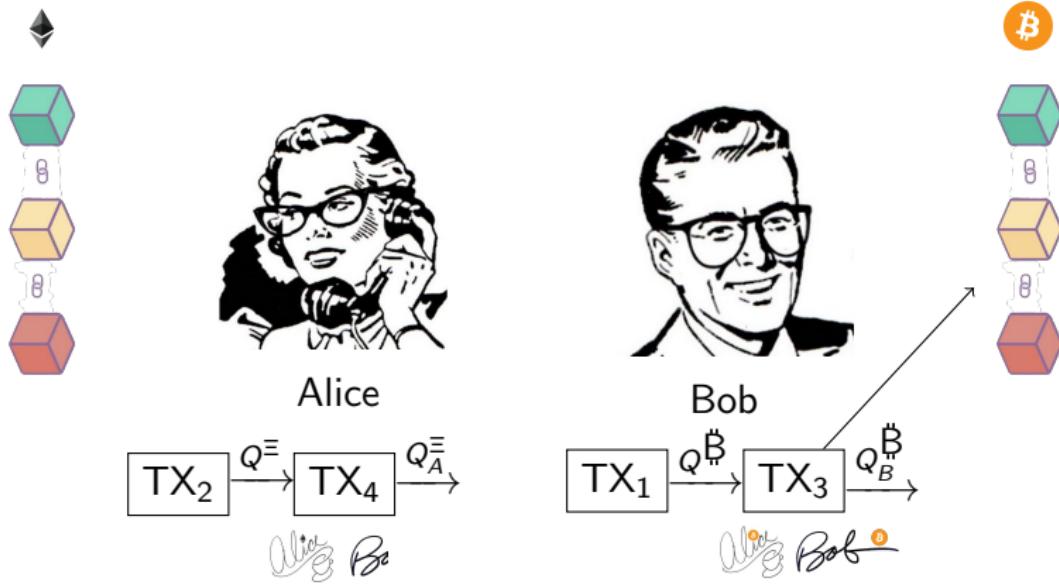
Whenever Bob wants to take the coins he can add t to the adaptor signature, making it valid: $(x_{K_A^{\mathbb{B}} + K_B^{\mathbb{B}} + T}, s'^{\mathbb{B}} + t)$.

Cross-chain atomic swaps via adaptor signatures



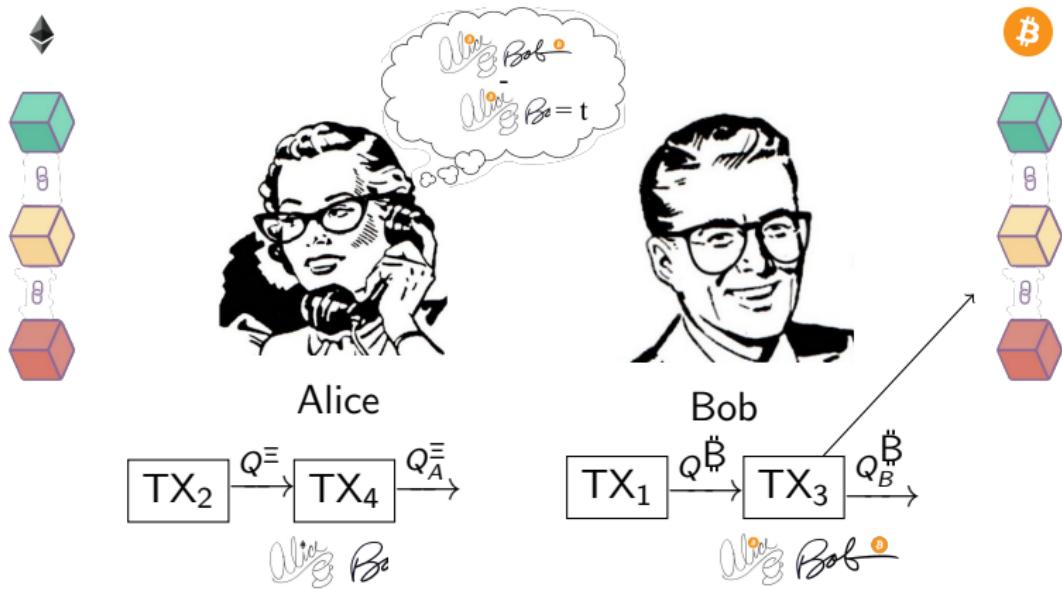
Whenever Bob wants to take the coins he can add t to the adaptor signature, making it valid: $(x_{K_A^{\mathbb{B}} + K_B^{\mathbb{B}} + T}, s'^{\mathbb{B}} + t)$.

Cross-chain atomic swaps via adaptor signatures



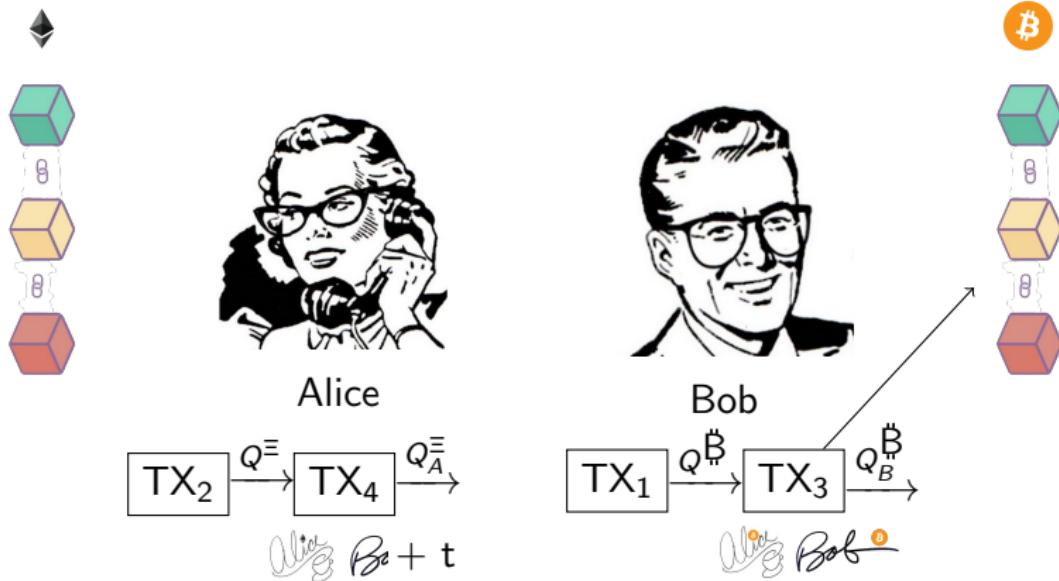
Now he can broadcast to the blockchain.

Cross-chain atomic swaps via adaptor signatures



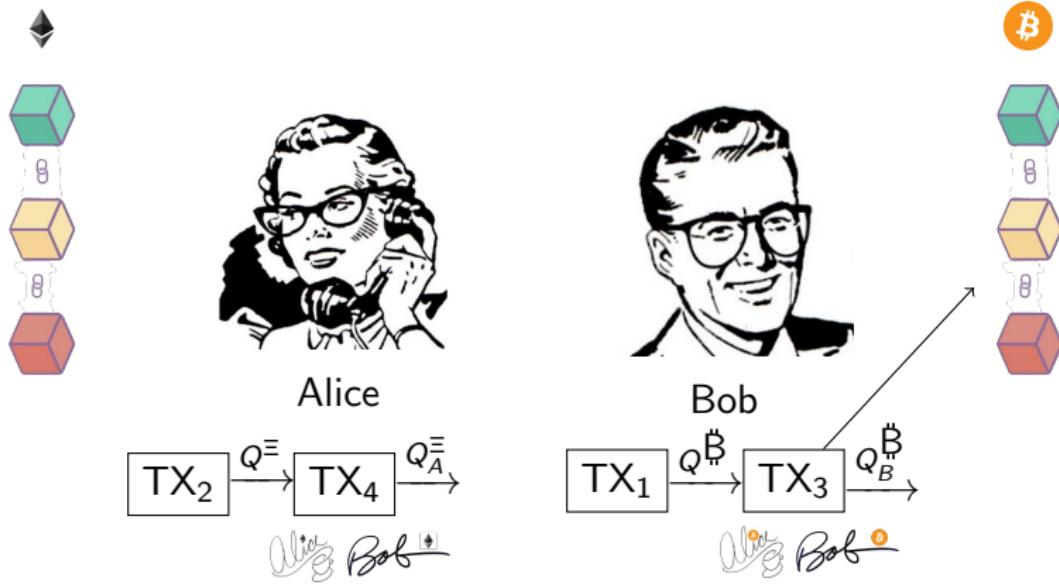
Monitoring the blockchain Alice sees the valid signature:
subtracting from it the adaptor signature she learns the secret t .

Cross-chain atomic swaps via adaptor signatures



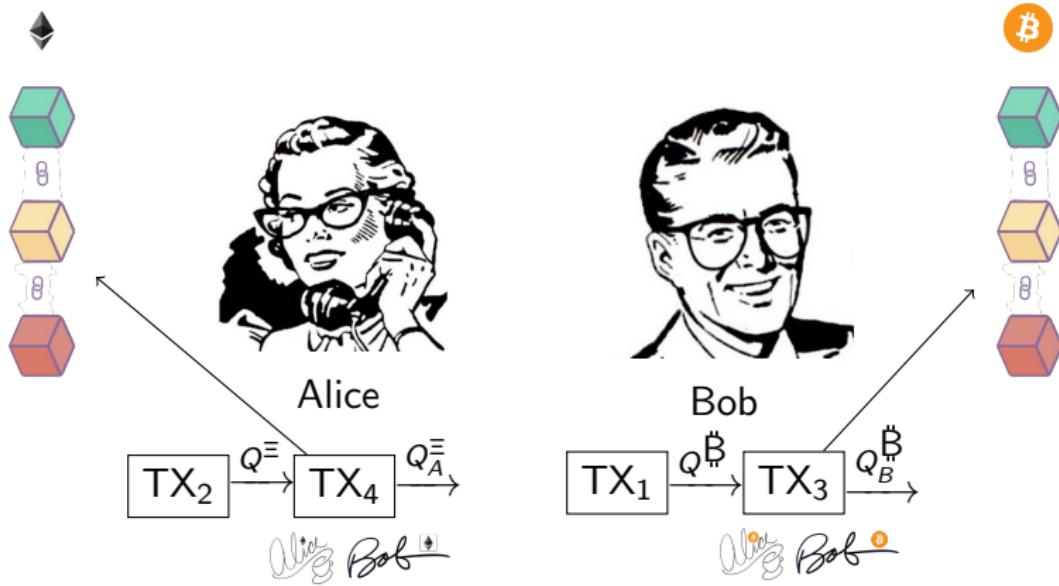
Adding it to the other adaptor signature results again in a valid signature for TX_4 .

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.

Cross-chain atomic swaps via adaptor signature

Efficiency and privacy gains:

- ▶ We were able to condense 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

Mathematical background and cryptographic primitives

Digital signature schemes

ECSSA applications

Conclusion

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;
- ▶ Also security would be affected, 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;
- ▶ Also security would be affected, 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 Merkleized 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 |

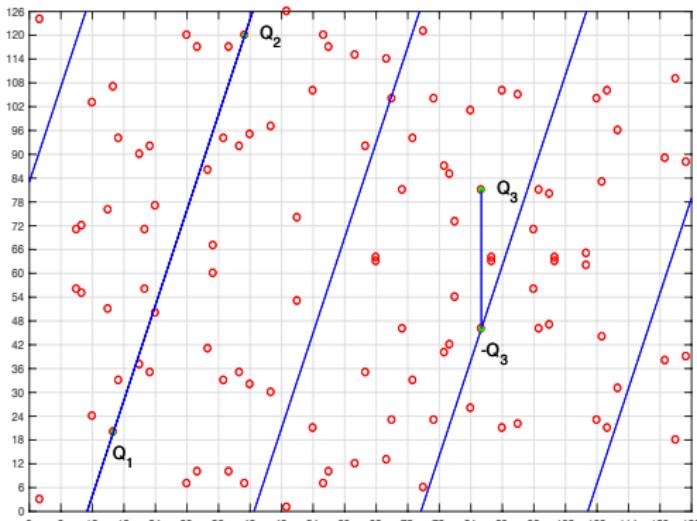
- [1] Gregory Maxwell, Andrew Poelstra, Yannick Seurin, and Pieter Wuille. Simple schnorr multi-signatures with applications to bitcoin. <https://eprint.iacr.org/2018/068.pdf>, 2018.
- [2] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. <https://bitcoin.org/bitcoin.pdf>, 2008.
- [3] 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 $q \rightarrow Q$ can be made efficiently (i.e. there exist some polynomial time algorithms);
- ▶ The inverse operation $Q \rightarrow q$ 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 $K = sG - \text{hash}(x_K \parallel Q \parallel m)Q$.
Thus, two valid signatures (K_0, s_0) and (K_1, s_1) satisfies:

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

Insecure: introduction of random factors.

$$\begin{aligned} a_0 K_0 + a_1 K_1 &= \\ &= (a_0 s_0 + a_1 s_1)G - a_0 \text{hash}(x_{K_0} \parallel Q_0 \parallel m_0)Q_0 - a_1 \text{hash}(x_{K_1} \parallel Q_1 \parallel m_1)Q_1. \end{aligned}$$

Batch validation - Bos-Coster's algorithm

$$\begin{aligned} & a_0 K_0 + a_1 K_1 = \\ & = (a_0 - a_1) K_0 + a_1 (K_0 + K_1). \end{aligned}$$

- ▶ 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) .

