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

[1]Fungibility is the property of a good or a commodity whose individual units are indistinguishable, thus interchangeable.

Outline

Hash functions ($\simeq$ Random oracles)

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

Elliptic curve cryptography

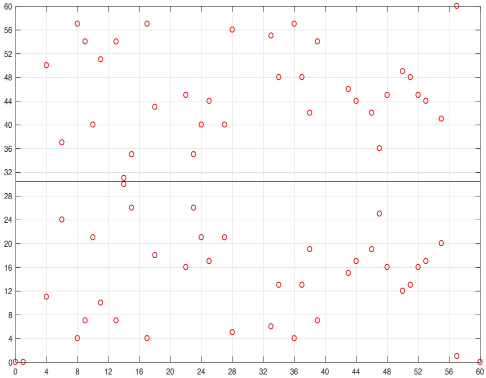
An elliptic curve over a finite field is defined by the equation:

$$E: \; y^2 = x^3 + ax + b \pmod{p}.$$

Bitcoin: $a = 0, b = 7, p \simeq 2^{256}$.

It is possible to define:

- Addition:
  $Q_3 := Q_1 + Q_2$,
  $\forall Q_1, Q_2 \in E$;
- Scalar multiplication:
  $qG := G + \ldots + G, \forall G \in E$,
  $\forall q \in \mathbb{N}$.



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

Discrete logarithm problem

Fixed $G \in E$, we can define $Q = qG \ \ \forall q \in \{1, ..., n-1\}$, 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

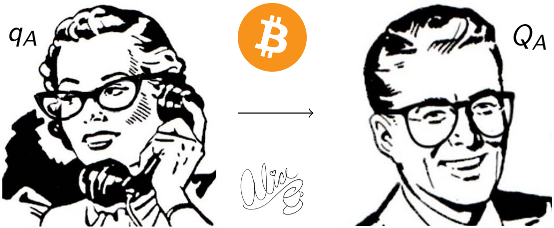
Digital signature

Alice Bob

Digital signature

Alice Bob

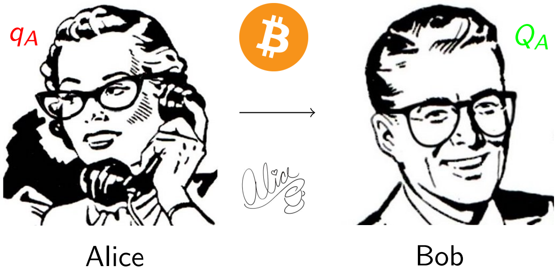
Digital signature



Alice Bob

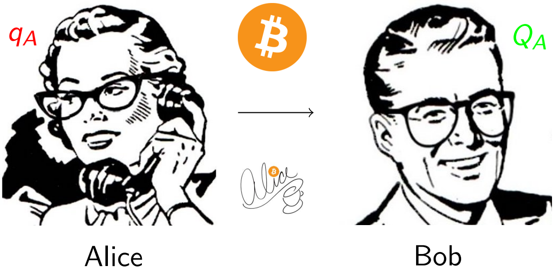
- Authentication: the recipient is confident that the funds come from the owner;

Digital signature



- Authentication: the recipient is confident that the funds come from the owner;
- Non repudiation: the sender cannot deny having sent the funds;

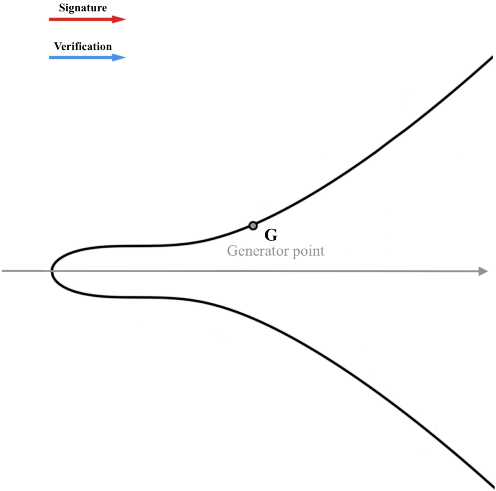
Digital signature



- Authentication: the recipient is confident that the funds come from the owner;
- Non repudiation: the sender cannot deny having sent the funds;
- Integrity: ensures that the transaction has not been altered during transmission.

Elliptic curve digital signature algorithm

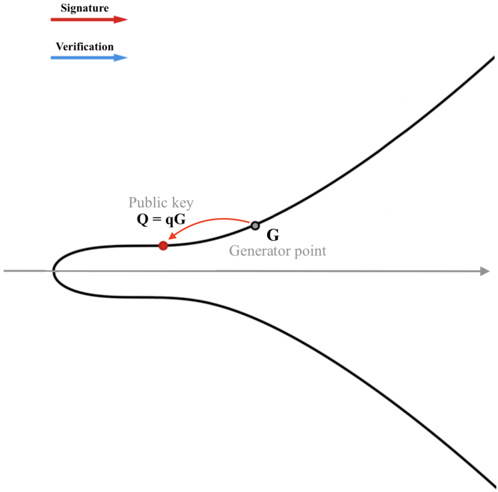
ECDSA_SIG(m, q):



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

Elliptic curve digital signature algorithm

ECDSA_SIG(m, q):

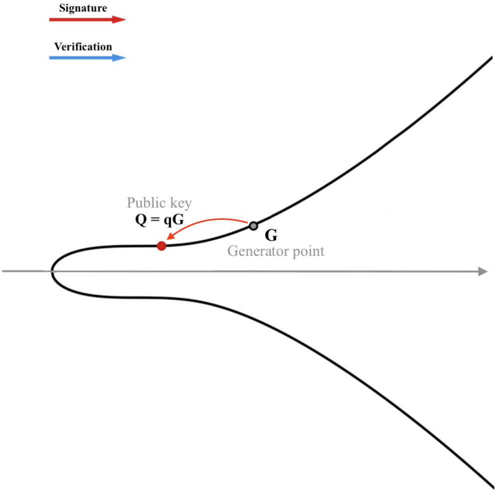


Adapted from:
`https://medium.com/cryptoadvance/how-schnorr-signatures-may-improve-bitcoin-91655bcb4744`

Elliptic curve digital signature algorithm

ECDSA_SIG(m, q):

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

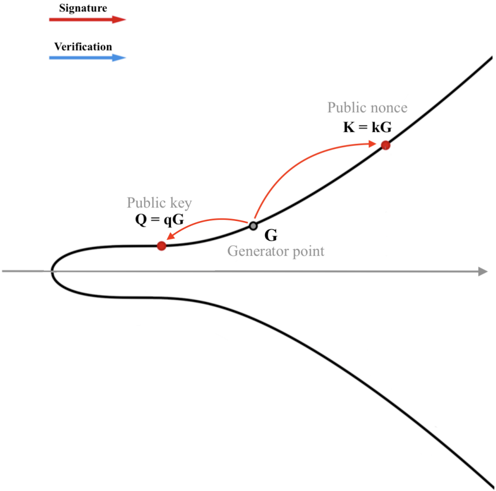


Adapted from:
`https://medium.com/cryptoadvance/how-schnorr-signatures-may-improve-bitcoin-91655bcb4744`

Elliptic curve digital signature algorithm

ECDSA_SIG(m, q):

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

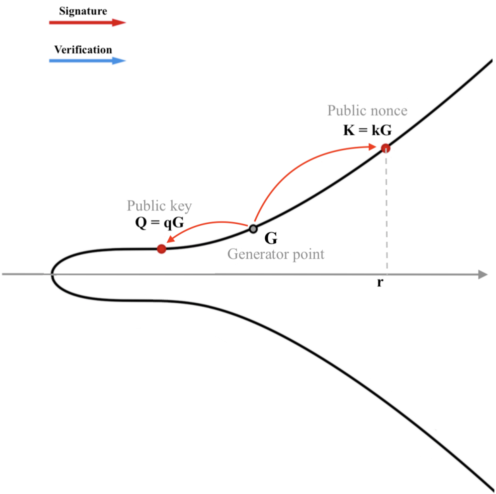


Adapted from:
`https://medium.com/cryptoadvance/how-schnorr-signatures-may-improve-bitcoin-91655bcb4744`

Elliptic curve digital signature algorithm

ECDSA_SIG(m, q):

1. $k \xleftarrow{\$} \{1, \ldots, n-1\}$;
2. $K \leftarrow kG$;
3. $r \leftarrow x_K \pmod{n}$;

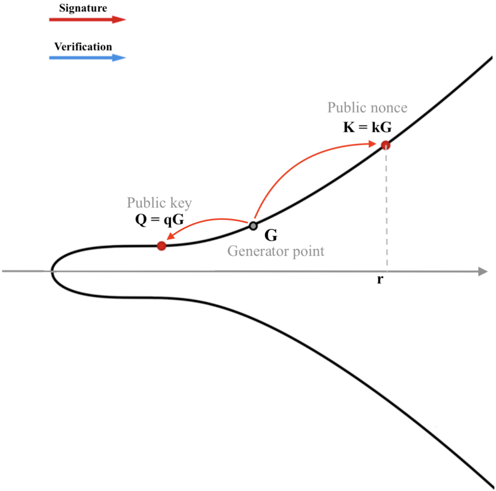


Adapted from:
`https://medium.com/cryptoadvance/how-schnorr-signatures-may-improve-bitcoin-91655bcb4744`

Elliptic curve digital signature algorithm

ECDSA_SIG(m, q):

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

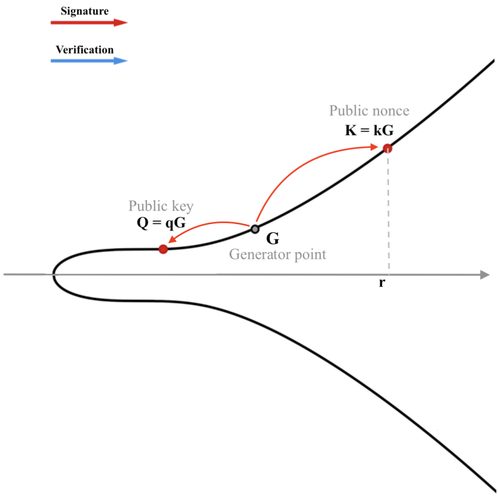


Adapted from:
`https://medium.com/cryptoadvance/how-schnorr-signatures-may-improve-bitcoin-91655bcb4744`

Elliptic curve digital signature algorithm

ECDSA_SIG(m, q):

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

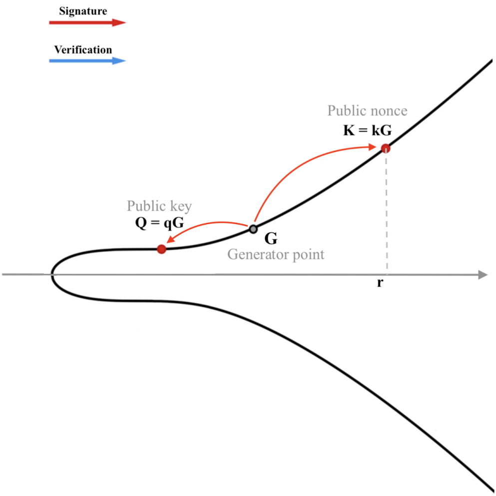


Adapted from:
`https://medium.com/cryptoadvance/how-schnorr-signatures-may-improve-bitcoin-91655bcb4744`

Elliptic curve digital signature algorithm

ECDSA_SIG(m, q):

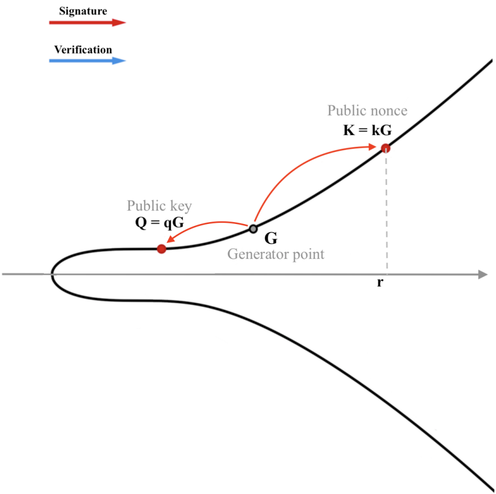
1. $k \xleftarrow{\$} \{1, \ldots, n-1\}$;
2. $K \leftarrow kG$;
3. $r \leftarrow x_K \pmod{n}$;
4. $e \leftarrow \text{hash}(m)$;
5. $s \leftarrow k^{-1}(e + rq) \pmod{n}$;
6. **return** (r, s).



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

Elliptic curve digital signature algorithm

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

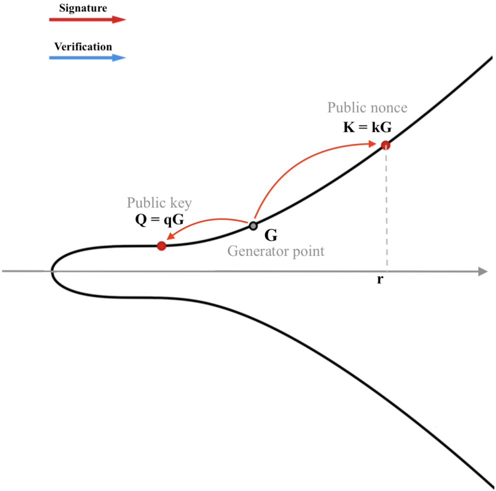


Adapted from:
`https://medium.com/cryptoadvance/how-schnorr-signatures-may-improve-bitcoin-91655bcb4744`

Elliptic curve digital signature algorithm

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

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

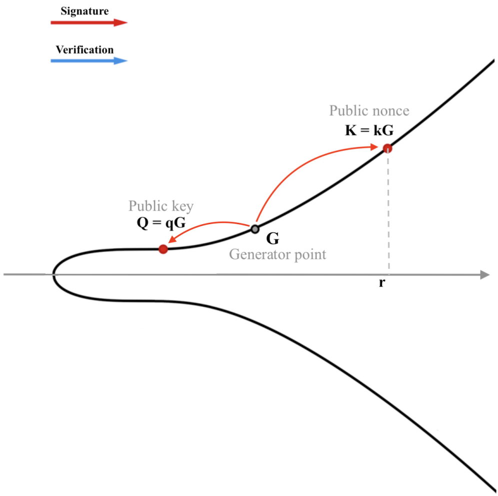


Adapted from:
`https://medium.com/cryptoadvance/how-schnorr-signatures-may-improve-bitcoin-91655bcb4744`

Elliptic curve digital signature algorithm

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

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

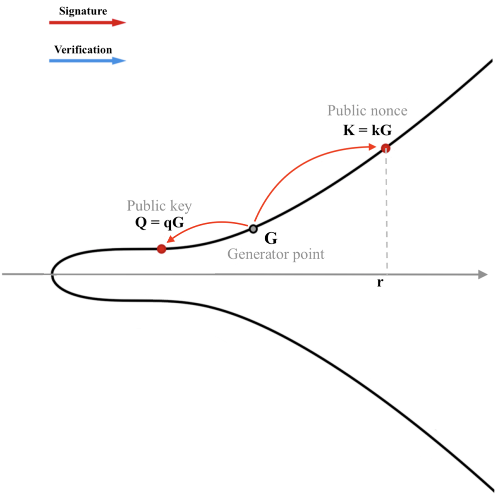


Adapted from:
`https://medium.com/cryptoadvance/how-schnorr-signatures-may-improve-bitcoin-91655bcb4744`

Elliptic curve digital signature algorithm

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

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

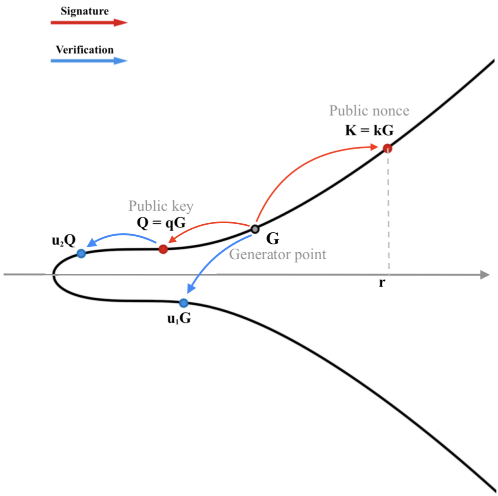


Adapted from: `https://medium.com/cryptoadvance/how-schnorr-signatures-may-improve-bitcoin-91655bcb4744`

Elliptic curve digital signature algorithm

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

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

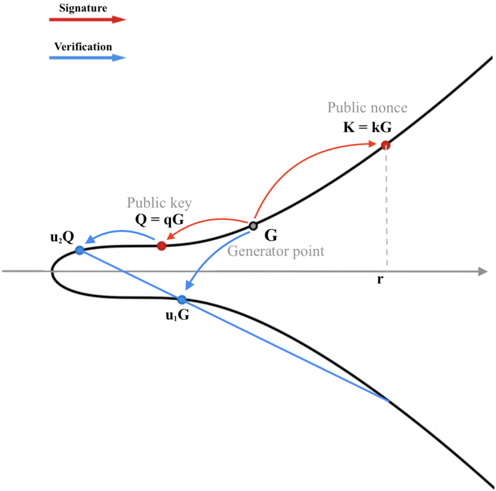


Adapted from:
`https://medium.com/cryptoadvance/how-schnorr-signatures-may-improve-bitcoin-91655bcb4744`

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. $e \leftarrow \text{hash}(m)$;
3. $u_1 \leftarrow es^{-1} \pmod{n}$,
   $u_2 \leftarrow rs^{-1} \pmod{n}$;
4. $K \leftarrow u_1 G + u_2 Q$;

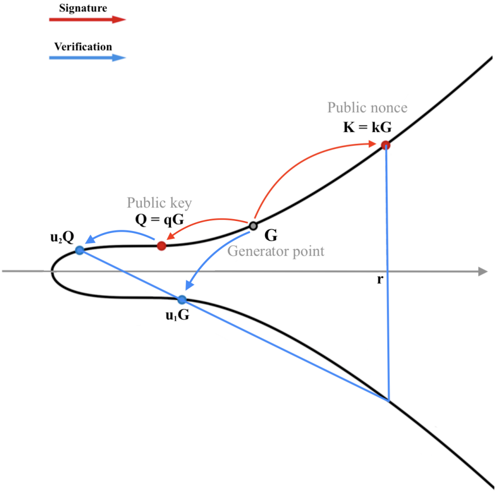


Adapted from:
`https://medium.com/cryptoadvance/how-schnorr-signatures-may-improve-bitcoin-91655bcb4744`

Elliptic curve digital signature algorithm

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

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

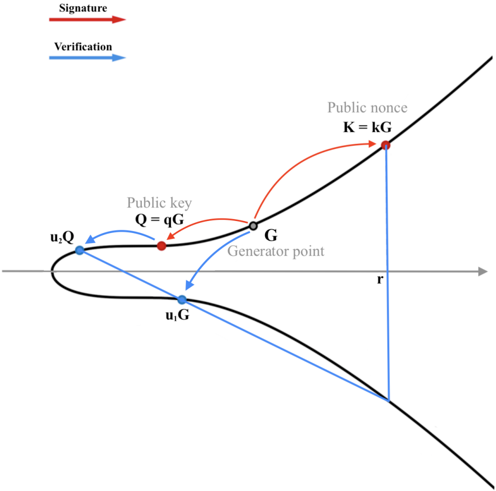


Adapted from:
`https://medium.com/cryptoadvance/how-schnorr-signatures-may-improve-bitcoin-91655bcb4744`

Elliptic curve digital signature algorithm

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

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

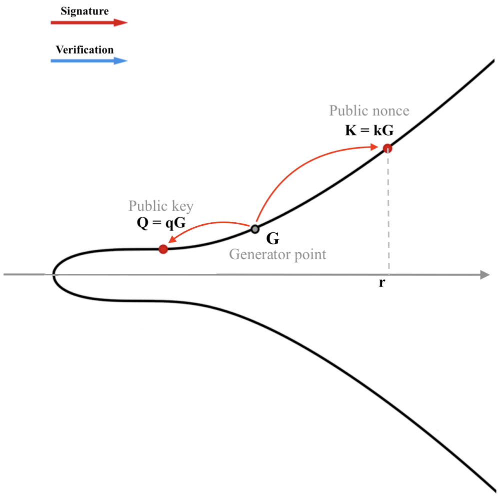


Adapted from:
`https://medium.com/cryptoadvance/how-schnorr-signatures-may-improve-bitcoin-91655bcb4744`

Elliptic curve Schnorr signature algorithm

ECDSA_SIG(m, q):

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

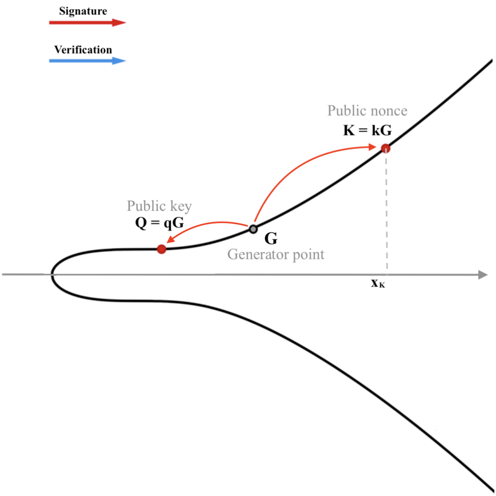


Adapted from:
`https://medium.com/cryptoadvance/how-schnorr-signatures-may-improve-bitcoin-91655bcb4744`

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. $e \leftarrow \text{hash}(m)$;
5. $s \leftarrow k^{-1}(e + rq) \pmod{n}$;
6. **return** (x_K, s).

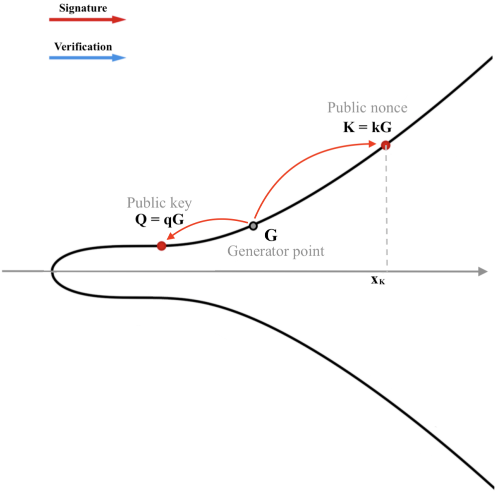


Adapted from:
https://medium.com/cryptoadvance/how-schnorr-signatures-may-improve-bitcoin-91655bcb4744

Elliptic curve Schnorr signature algorithm

ECSSA_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. $s \leftarrow k^{-1}(e + rq) \pmod{n}$;
6. **return** (x_K, s).

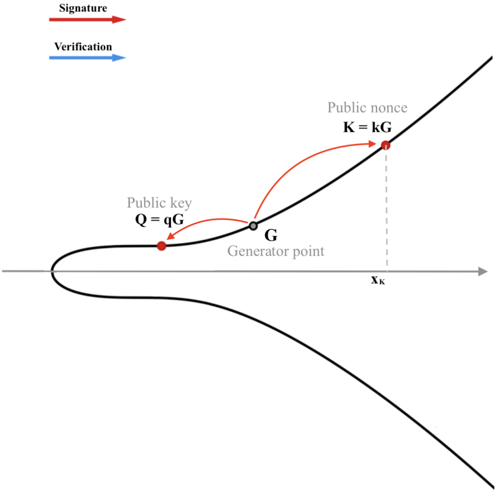


Adapted from:
`https://medium.com/cryptoadvance/how-schnorr-signatures-may-improve-bitcoin-91655bcb4744`

Elliptic curve Schnorr signature algorithm

ECSSA_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).

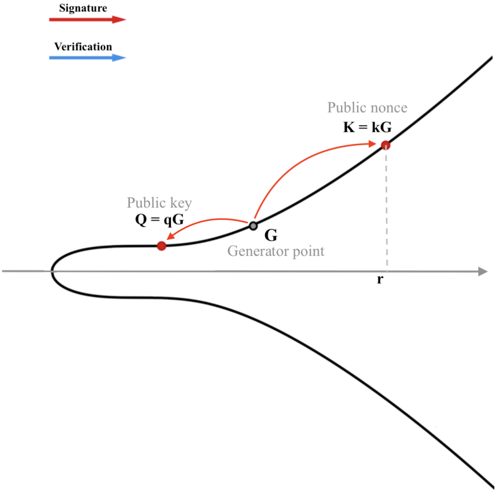


Adapted from:
https://medium.com/cryptoadvance/how-schnorr-signatures-may-improve-bitcoin-91655bcb4744

Elliptic curve Schnorr signature algorithm

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

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

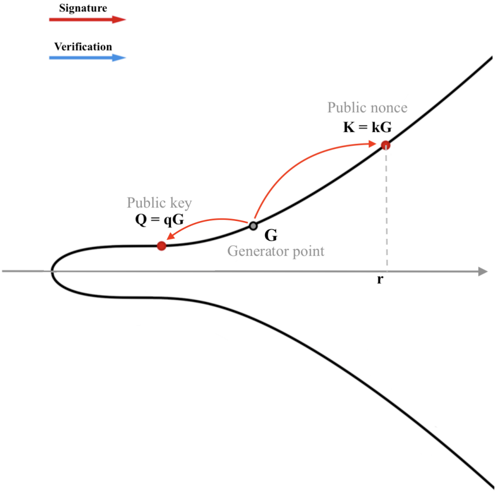


Adapted from:
`https://medium.com/cryptoadvance/how-schnorr-signatures-may-improve-bitcoin-91655bcb4744`

Elliptic curve Schnorr signature algorithm

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

1. **If** $r \notin \{1, ..., p-1\}$ **or** $s \notin \{1, ..., n-1\}$:
   return False;
2. $e \leftarrow \mathrm{hash}(m)$;
3. $K \leftarrow es^{-1}G + rs^{-1}Q$;
4. **return** $r = x_K \pmod{n}$.

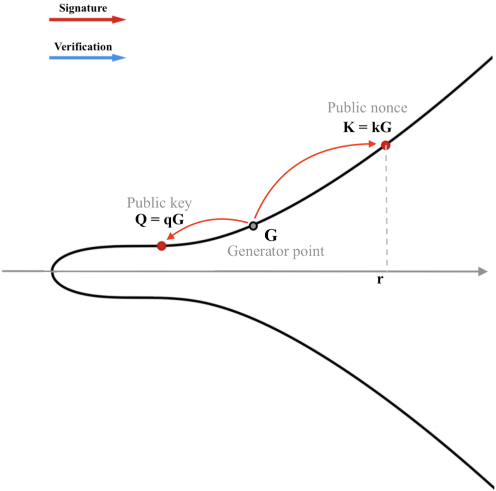


Adapted from:
https://medium.com/cryptoadvance/how-schnorr-signatures-may-improve-bitcoin-91655bcb4744

Elliptic curve Schnorr signature algorithm

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

1. **If** $r \notin \{1, \ldots, p-1\}$ **or** $s \notin \{1, \ldots, n-1\}$:
   return False;
2. $e \leftarrow \text{hash}(r||Q||m) \pmod{n}$;
3. $K \leftarrow es^{-1}G + rs^{-1}Q$;
4. **return** $r = x_K \pmod{n}$.

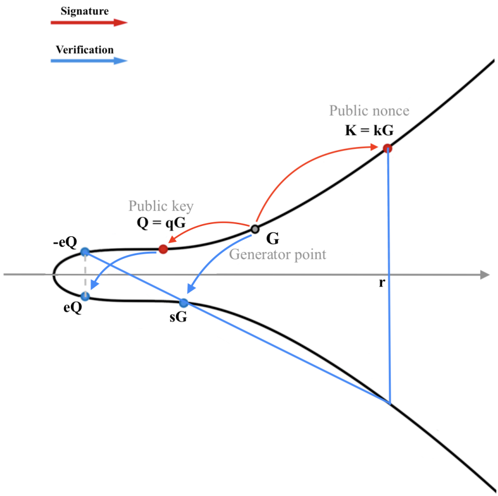


Adapted from:
`https://medium.com/cryptoadvance/how-schnorr-signatures-may-improve-bitcoin-91655bcb4744`

Elliptic curve Schnorr signature algorithm

ECSSA_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. **return** $r = x_K \pmod n$.

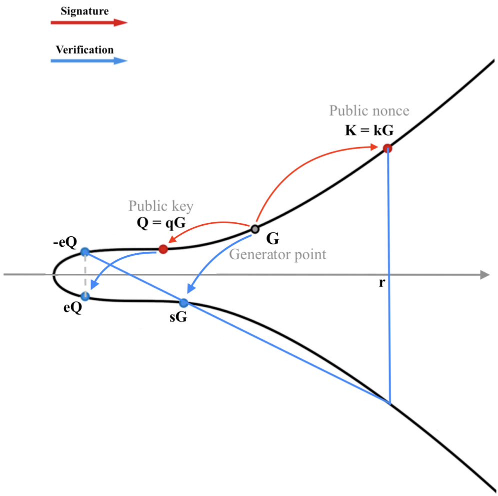


Adapted from:
`https://medium.com/cryptoadvance/how-schnorr-signatures-may-improve-bitcoin-91655bcb4744`

Elliptic curve Schnorr signature algorithm

ECSSA_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. **return** $r = x_K$ **and** $\text{jacobi}(y_K) = 1$.



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 72 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 72 bytes;
- Chosen standardization forbid batch validation;

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 72 bytes;
- Chosen standardization forbid batch validation;
- 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;
- 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 embeds the spending conditions.

In conjunction with signatures, they enforce the property right in the digital realm: this is why signatures are typically necessary to spend bitcoins and are required by the unlocking sripts.

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.

ECDSA multi-signature (*t*-of-*m*)

- 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.

ECDSA multi-signature (t-of-m)

- 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 ($\{q_B, Q_B\}$) generate 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: $s_i = k_i + \text{hash}(x_K || Q || msg) q_i \pmod{n}, \ i \in \{A, B\}$;
- The signature $(x_K, s_A + s_B \pmod{n})$ is valid for msg and 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.

ECDSA multi-signature (t-of-m)

- 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 ($\{q_B, Q_B\}$) generate 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: $s_i = k_i + \text{hash}(x_K || Q || msg) q_i \pmod n$, $i \in \{A, B\}$;
- The signature $(x_K, s_A + s_B \pmod n)$ is valid for msg and 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) assumption, the idea is to introduce a “random factor” (ROM) $a_i = \text{hash}(\{Q_1, ..., Q_m\}||Q_i)$ 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) assumption, the idea is to introduce a "random factor" (ROM) $a_i = \text{hash}(\{Q_1, ..., Q_m\} || Q_i)$ 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) assumption, the idea is to introduce a "random factor" (ROM) $a_i = \text{hash}(\{Q_1, ..., Q_m\}||Q_i)$ 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 $r \in \{1, ..., n-1\}$;
  - He embeds it in a random polynomial of degree $t-1$: $f(u) = r + f_1 u + ... + f_{t-1} u^{t-1} \pmod{n}$, $f_1, ..., f_{t-1} \xleftarrow{\$} \{0, ..., n-1\}$;
  - Each participant $i = 1, ..., m$ of the scheme receives the share $s_i = f(i)$;
  - The secret is reconstructed by any coalition $\mathcal{P}$ with $|\mathcal{P}| = t$ via Lagrange's interpolation formula: $r = f(0) = \sum_{i \in \mathcal{P}} s_i \prod_{h \in \mathcal{P}, h \neq i} \frac{h}{h-i} \pmod{n}$.

Threshold signature scheme (t-of-m)

- Pedersen verifiable secret sharing scheme:
  - A dealer chooses secret $r \in \{1, ..., n-1\}$;
  - He embeds it in a random polynomial of degree $t-1$:
    $f(u) = r + f_1 u + ... + f_{t-1} u^{t-1} \pmod{n}$,
    $f_1, ..., f_{t-1} \xleftarrow{\$} \{0, ..., n-1\}$;
  - Each participant $i = 1, ..., m$ of the scheme receives the share $s_i = f(i)$;
  - The secret is reconstructed by any coalition $\mathcal{P}$ with $|\mathcal{P}| = t$ via Lagrange's interpolation formula:
    $r = f(0) = \sum_{i \in \mathcal{P}} s_i \prod_{h \in \mathcal{P}, h \neq i} \frac{h}{h-i} \pmod{n}$.
- Protocol for the generation of a shared random secret: every participant acts as the dealer in the previous protocol with secret r_i and sharing polynomial $f_i(u)$:
  - Shared secret: $r = \sum_{i=1}^{m} r_i \pmod{n} \implies \{r, R\}$;
  - Global sharing polynomial: $F(u) = \sum_{i=1}^{m} f_i(u) \pmod{n}$;
  - Share of the secret belonging to i:
    $s_i = \sum_{j=1}^{m} f_j(i) \pmod{n} = F(i)$.

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 $\alpha_i = F_1(i)$, $i = 1, ..., m$) and a nonce pair $\{k, K\}$ (shares $\beta_i = F_2(i)$, $i = 1, ..., t$);
- Each signer $i = 1, ..., t$ computes his partial signature as: $\gamma_i = \beta_i + e\alpha_i \pmod{n}$, $e = \text{hash}(x_K||Q||msg) \pmod{n}$;
- The signature is (x_K, s), with $s = \sum_{i=1}^{t} \gamma_i \prod_{h \neq i} \frac{h}{h-i} \pmod{n}$.

s computed in this way satisfies $s = k + eq \pmod{n}$:

$$F_3(u) := F_2(u) + eF_1(u) \implies$$

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

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

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.

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, and hidden policies;
- Improved security, being Schnorr 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, and hidden policies;
- Improved security, being Schnorr provably secure.

But there is even more:

- Adaptor signatures: they aim at reducing the verbose scripting semantics to a fixed size signature with huge benefits for privacy and efficiency in protocols like atomic swaps and the Lightning Network;
- 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 to look the same as a single signer transaction in the cooperative case.

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.

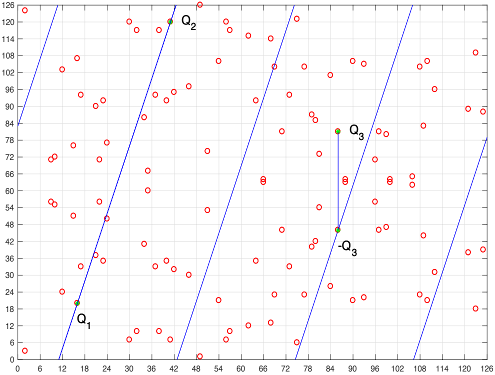
References II

[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 $q \mapsto Q$ can be made efficiently (i.e. there exist some polynomial time algorithms);
- The inverse operation $Q \mapsto q$ 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 $K = sG - \text{hash}(x_K \mid\mid Q \mid\mid m)Q$.
Thus, two valid signatures (K_0, s_0) and (K_1, s_1) satisfies:

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

Insecure (cancelation attack): introduction of random factors.

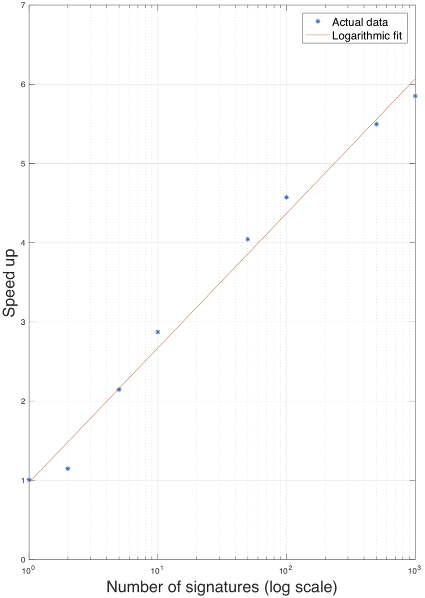
$$a_0 K_0 + a_1 K_1 =$$

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

Batch validation - Bos-Coster's algorithm

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

- Sort the tuples (a_i, K_i) 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).



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:

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

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

Adaptor signatures can be used in conjunction with the MuSig protocol: a signer generates $s_i' = k_i + \mathrm{hash}(x_K||Q||msg)a_i q_i \pmod{n}$ 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 achieved via Hashed TimeLock Contract:

HTLC

- 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 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.