Multi-Signatures for Blockchains

Yannick Seurin

Agence nationale de la sécurité des systèmes d'information

June 12, 2019 — LINCS Blockchain Day

Uses of cryptography in blockchains

- define valid transactions
  - signatures
  - multi-, threshold, aggregate, ... signatures
- achieve distributed consensus on the state of the ledger
  - proof of work: hash functions
  - proof of stake:
    - verifiable random functions (VRFs)
    - verifiable delay functions (VDFs)
  - proof of space
- provide privacy
  - ring signatures, stealth addresses (Monero)
  - confidential transactions (homomorphic commitments, range proofs)
  - zero-knowledge proofs / ZK-SNARKs (Zcash)

Uses of cryptography in blockchains

- define valid transactions
  - signatures
  - multi-, threshold, aggregate, ... signatures
- achieve distributed consensus on the state of the ledger
  - proof of work: hash functions
  - proof of stake:
    - verifiable random functions (VRFs)
    - verifiable delay functions (VDFs)
  - proof of space
- provide privacy
  - ring signatures, stealth addresses (Monero)
  - confidential transactions (homomorphic commitments, range proofs)
  - zero-knowledge proofs / ZK-SNARKs (Zcash)

Uses of cryptography in blockchains

- define valid transactions
  - signatures
  - multi-, threshold, aggregate, . . . signatures
- achieve distributed consensus on the state of the ledger
  - proof of work: hash functions
  - proof of stake:
    - verifiable random functions (VRFs)
    - verifiable delay functions (VDFs)
  - proof of space
- provide privacy
  - ring signatures, stealth addresses (Monero)
  - confidential transactions (homomorphic commitments, range proofs)
  - zero-knowledge proofs / ZK-SNARKs (Zcash)

Uses of cryptography in blockchains

- define valid transactions
  - signatures
  - multi-, threshold, aggregate, ... signatures
- achieve distributed consensus on the state of the ledger
  - proof of work: hash functions
  - proof of stake:
    - verifiable random functions (VRFs)
    - verifiable delay functions (VDFs)
  - proof of space
- provide privacy
  - ring signatures, stealth addresses (Monero)
  - confidential transactions (homomorphic commitments, range proofs)
  - zero-knowledge proofs / ZK-SNARKs (Zcash)

Uses of cryptography in blockchains

- define valid transactions
  - signatures
  - multi-, threshold, aggregate, ... signatures
- achieve distributed consensus on the state of the ledger
  - proof of work: hash functions
  - proof of stake:
    - verifiable random functions (VRFs)
    - verifiable delay functions (VDFs)
  - proof of space
- provide privacy
  - ring signatures, stealth addresses (Monero)
  - confidential transactions (homomorphic commitments, range proofs)
  - zero-knowledge proofs / ZK-SNARKs (Zcash)

Uses of cryptography in blockchains

- define valid transactions
  - signatures
  - multi-, threshold, aggregate, ... signatures
- achieve distributed consensus on the state of the ledger
  - proof of work: hash functions
  - proof of stake:
    - verifiable random functions (VRFs)
    - verifiable delay functions (VDFs)
  - proof of space
- provide privacy
  - ring signatures, stealth addresses (Monero)
  - confidential transactions (homomorphic commitments, range proofs)
  - zero-knowledge proofs / ZK-SNARKs (Zcash)

Uses of cryptography in blockchains

- define valid transactions
  - signatures
  - multi-, threshold, aggregate, ... signatures
- achieve distributed consensus on the state of the ledger
  - proof of work: hash functions
  - proof of stake:
    - verifiable random functions (VRFs)
    - verifiable delay functions (VDFs)
  - proof of space
- provide privacy
  - ring signatures, stealth addresses (Monero)
  - confidential transactions (homomorphic commitments, range proofs)
  - zero-knowledge proofs / ZK-SNARKs (Zcash)

Uses of cryptography in blockchains

- define valid transactions
  - signatures
  - multi-, threshold, aggregate, . . . signatures
- achieve distributed consensus on the state of the ledger
  - proof of work: hash functions
  - proof of stake:
    - verifiable random functions (VRFs)
    - verifiable delay functions (VDFs)
  - proof of space
- provide privacy
  - ring signatures, stealth addresses (Monero)
  - confidential transactions (homomorphic commitments, range proofs)
  - zero-knowledge proofs / ZK-SNARKs (Zcash)

Uses of cryptography in blockchains

- define valid transactions
  - signatures
  - multi-, threshold, aggregate, . . . signatures
- achieve distributed consensus on the state of the ledger
  - proof of work: hash functions
  - proof of stake:
    - verifiable random functions (VRFs)
    - verifiable delay functions (VDFs)
  - proof of space
- provide privacy
  - ring signatures, stealth addresses (Monero)
  - confidential transactions (homomorphic commitments, range proofs)
  - zero-knowledge proofs / ZK-SNARKs (Zcash)

Uses of cryptography in blockchains

- define valid transactions
  - signatures
  - multi-, threshold, aggregate, ... signatures
- achieve distributed consensus on the state of the ledger
  - proof of work: hash functions
  - proof of stake:
    - verifiable random functions (VRFs)
    - verifiable delay functions (VDFs)
  - proof of space
- provide privacy
  - ring signatures, stealth addresses (Monero)
  - confidential transactions (homomorphic commitments, range proofs)
  - zero-knowledge proofs / ZK-SNARKs (Zcash)

Uses of cryptography in blockchains

- define valid transactions
  - signatures
  - multi-, threshold, aggregate, ... signatures
- achieve distributed consensus on the state of the ledger
  - proof of work: hash functions
  - proof of stake:
    - verifiable random functions (VRFs)
    - verifiable delay functions (VDFs)
  - proof of space
- provide privacy
  - ring signatures, stealth addresses (Monero)
  - confidential transactions (homomorphic commitments, range proofs)
  - zero-knowledge proofs / ZK-SNARKs (Zcash)

Uses of cryptography in blockchains

- define valid transactions
  - signatures
  - multi-, threshold, aggregate, . . . signatures
- achieve distributed consensus on the state of the ledger
  - proof of work: hash functions
  - proof of stake:
    - verifiable random functions (VRFs)
    - verifiable delay functions (VDFs)
  - proof of space
- provide privacy
  - ring signatures, stealth addresses (Monero)
  - confidential transactions (homomorphic commitments, range proofs)
  - zero-knowledge proofs / ZK-SNARKs (Zcash)

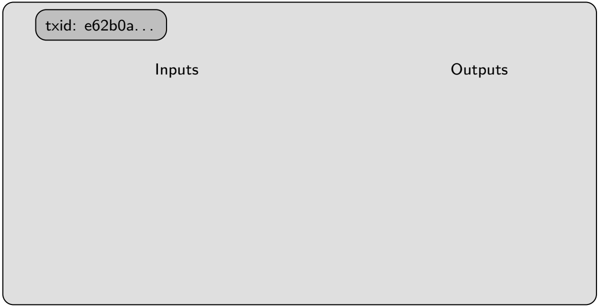
Uses of cryptography in blockchains

- define valid transactions
  - signatures
  - multi-, threshold, aggregate, ... signatures
- achieve distributed consensus on the state of the ledger
  - proof of work: hash functions
  - proof of stake:
    - verifiable random functions (VRFs)
    - verifiable delay functions (VDFs)
  - proof of space
- provide privacy
  - ring signatures, stealth addresses (Monero)
  - confidential transactions (homomorphic commitments, range proofs)
  - zero-knowledge proofs / ZK-SNARKs (Zcash)

Bitcoin transactions

A Bitcoin transaction spends inputs and creates outputs:

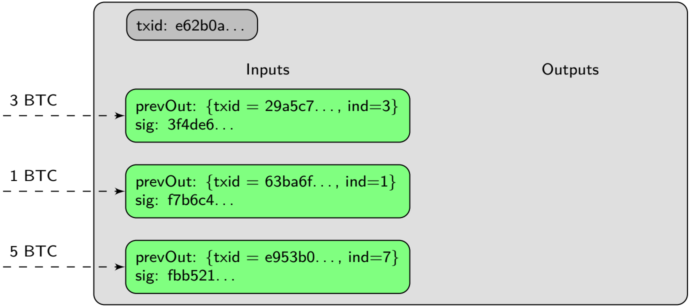
- an input consists of a reference to an output of a previous transaction and a signature authorizing spending of this output
- an output consists of an amount and a public key



Bitcoin transactions

A Bitcoin transaction spends inputs and creates outputs:

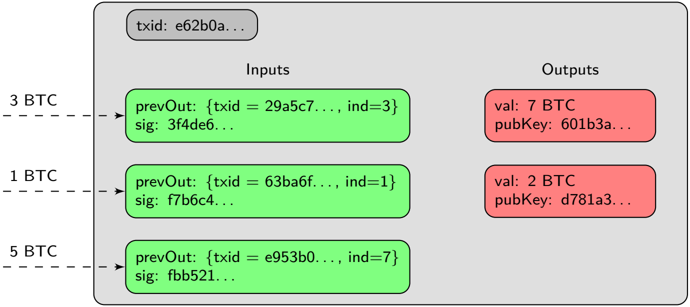
- an input consists of a reference to an output of a previous transaction and a signature authorizing spending of this output
- an output consists of an amount and a public key



Bitcoin transactions

A Bitcoin transaction spends inputs and creates outputs:

- an input consists of a reference to an output of a previous transaction and a signature authorizing spending of this output
- an output consists of an amount and a public key



Signatures in Bitcoin

- Bitcoin (and $\sim$ all blockchains) use ECDSA (curve secp256k1)
- size of an ECDSA public key: 33 bytes
- typical size of an ECDSA signature: 72 bytes
  (two 32-bytes integers + 6 bytes DER encoding)
- 420 000 000 transactions in the blockchain, $\sim$ 2 inputs/tx
  $\Rightarrow$ $\simeq$ 88 GB of pk+sig data (40% blockchain size)

Signatures in Bitcoin

- Bitcoin (and $\sim$ all blockchains) use ECDSA (curve secp256k1)
- size of an ECDSA public key: 33 bytes
- typical size of an ECDSA signature: 72 bytes
  (two 32-bytes integers + 6 bytes DER encoding)
- 420 000 000 transactions in the blockchain, $\sim$ 2 inputs/tx
  $\Rightarrow \simeq$ 88 GB of pk+sig data (40% blockchain size)

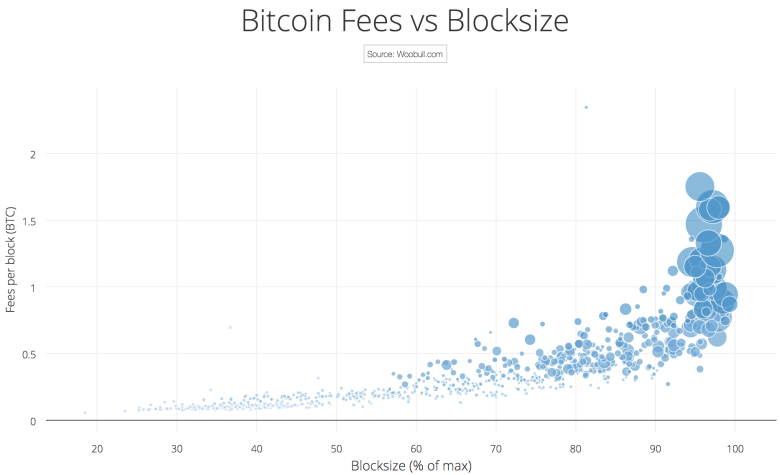
Signatures in Bitcoin

- Bitcoin (and $\sim$ all blockchains) use ECDSA (curve secp256k1)
- size of an ECDSA public key: 33 bytes
- typical size of an ECDSA signature: 72 bytes
  (two 32-bytes integers + 6 bytes DER encoding)
- 420 000 000 transactions in the blockchain, $\sim$ 2 inputs/tx
  $\Rightarrow \simeq$ 88 GB of pk+sig data (40% blockchain size)

Signatures in Bitcoin

- Bitcoin (and $\sim$ all blockchains) use ECDSA (curve secp256k1)
- size of an ECDSA public key: 33 bytes
- typical size of an ECDSA signature: 72 bytes
(two 32-bytes integers + 6 bytes DER encoding)
- 420 000 000 transactions in the blockchain, $\sim$ 2 inputs/tx
$\Rightarrow \simeq$ 88 GB of pk+sig data (40% blockchain size)

Optimizing transaction size matters



Miners fee per block vs block size (% of maximum), samples grouped by day. Bubble size denotes mempool size (for records Apr 2016 onwards).

Signature scheme: definition

A signature scheme consists of three algorithms:

1. key generation algorithm $\texttt{KeyGen}$:
   - returns a public/secret key pair (pk, sk)
2. signature algorithm $\texttt{Sign}$:
   - takes as input a secret key sk and a message m
   - returns a signature σ
3. verification algorithm $\texttt{Ver}$:
   - takes as input a public key pk, a message m, and a signature σ
   - returns 1 if the signature is valid and 0 otherwise

Correctness property:

$$\forall (pk, sk) \leftarrow \texttt{KeyGen},\ \forall m,\ \texttt{Ver}(pk, m, \texttt{Sign}(sk, m)) = 1$$

Signature scheme: definition

A signature scheme consists of three algorithms:

1. key generation algorithm $\texttt{KeyGen}$:
   - returns a public/secret key pair (pk, sk)
2. signature algorithm $\texttt{Sign}$:
   - takes as input a secret key sk and a message m
   - returns a signature σ
3. verification algorithm $\texttt{Ver}$:
   - takes as input a public key pk, a message m, and a signature σ
   - returns 1 if the signature is valid and 0 otherwise

Correctness property:

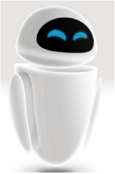
$$\forall (pk, sk) \leftarrow \texttt{KeyGen},\ \forall m,\ \texttt{Ver}(pk, m, \texttt{Sign}(sk, m)) = 1$$

Signature scheme: security

- "gold" security notion: Existential Unforgeability against Chosen Message Attacks (EUF-CMA)

sk_A

pk_A



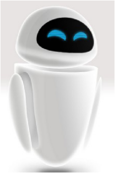
Signature scheme: security

- "gold" security notion: Existential Unforgeability against Chosen Message Attacks (EUF-CMA)

sk_A

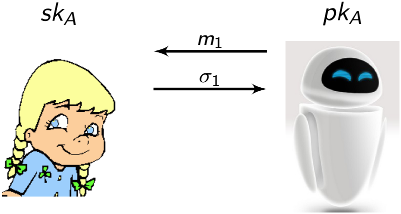
$\xleftarrow{m_1}$

pk_A



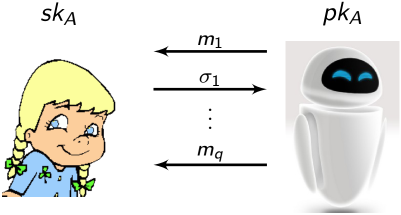
Signature scheme: security

- "gold" security notion: Existential Unforgeability against Chosen Message Attacks (EUF-CMA)



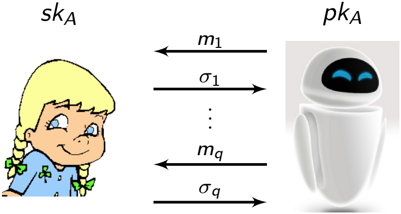
Signature scheme: security

- "gold" security notion: Existential Unforgeability against Chosen Message Attacks (EUF-CMA)



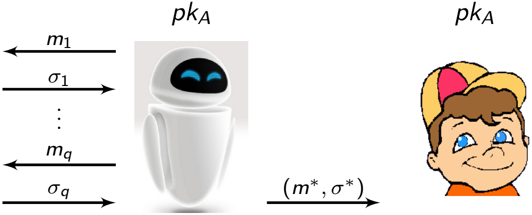
Signature scheme: security

- "gold" security notion: Existential Unforgeability against Chosen Message Attacks (EUF-CMA)



Signature scheme: security

- "gold" security notion: Existential Unforgeability against Chosen Message Attacks (EUF-CMA)



Signature scheme: security

- "gold" security notion: Existential Unforgeability against Chosen Message Attacks (EUF-CMA)

pk_A $\qquad\qquad$ pk_A

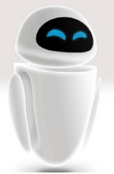
$\xleftarrow{m_1}$

$\xrightarrow{\sigma_1}$

$\vdots$

$\xleftarrow{m_q}$

$\xrightarrow{\sigma_q}$



$\xrightarrow{(m^*, \sigma^*)}$

$$\boxed{\begin{array}{c} m^* \neq m_1, \ldots, m_q \\ \texttt{Ver}(pk_A, m^*, \sigma^*) = 1 \end{array}}$$

Multi-signatures

- often, transactions must be authorized by multiple parties (shared wallet, escrow, payment channel, atomic swap, ...)
- currently in Bitcoin: trivial solution (concatenation of pks/sigs)
- better: one signature, independently of the number of signers
- even better: one public key, independently of the number of signers
- difficulty: rogue-key attacks (*plain public-key model*: no CA)

(sk_A, pk_A)

(sk_B, pk_B)

Multi-signatures

- often, transactions must be authorized by multiple parties (shared wallet, escrow, payment channel, atomic swap, ...)
- currently in Bitcoin: trivial solution (concatenation of pks/sigs)
- better: one signature, independently of the number of signers
- even better: one public key, independently of the number of signers
- difficulty: rogue-key attacks (*plain public-key model*: no CA)

(sk_A, pk_A)

$\downarrow$

σ_A

(sk_B, pk_B)

$\downarrow$

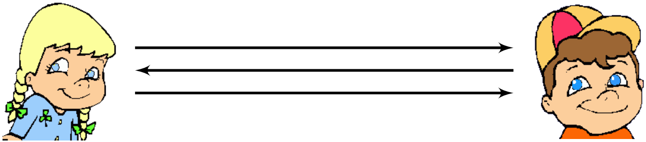
σ_B

$$\mathtt{Ver}(pk_A, m, \sigma_A) = 1$$
$$\mathtt{Ver}(pk_B, m, \sigma_B) = 1$$

Multi-signatures

- often, transactions must be authorized by multiple parties (shared wallet, escrow, payment channel, atomic swap, ...)
- currently in Bitcoin: trivial solution (concatenation of pks/sigs)
- better: one signature, independently of the number of signers
- even better: one public key, independently of the number of signers
- difficulty: rogue-key attacks (*plain public-key model*: no CA)

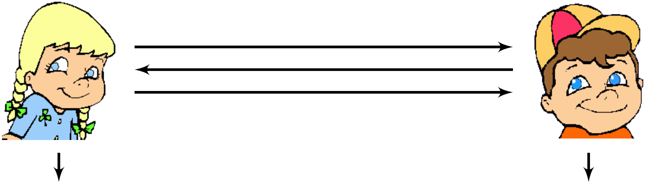
(sk_A, pk_A) (sk_B, pk_B)



Multi-signatures

- often, transactions must be authorized by multiple parties (shared wallet, escrow, payment channel, atomic swap, . . .)
- currently in Bitcoin: trivial solution (concatenation of pks/sigs)
- better: one signature, independently of the number of signers
- even better: one public key, independently of the number of signers
- difficulty: rogue-key attacks (*plain public-key model*: no CA)

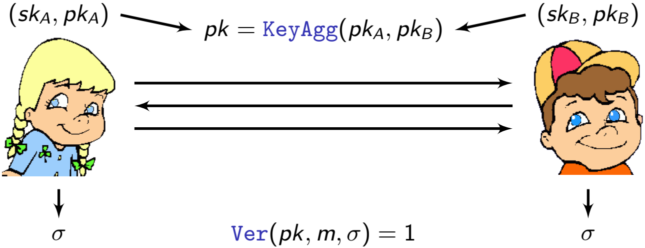
(sk_A, pk_A) (sk_B, pk_B)



σ $\texttt{Ver}(\{pk_A, pk_B\}, m, \sigma) = 1$ σ

Multi-signatures

- often, transactions must be authorized by multiple parties (shared wallet, escrow, payment channel, atomic swap, ...)
- currently in Bitcoin: trivial solution (concatenation of pks/sigs)
- better: one signature, independently of the number of signers
- even better: one public key, independently of the number of signers
- difficulty: rogue-key attacks (*plain public-key model*: no CA)



Multi-signatures

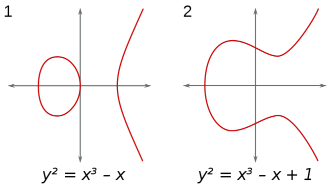
- often, transactions must be authorized by multiple parties (shared wallet, escrow, payment channel, atomic swap, ...)
- currently in Bitcoin: trivial solution (concatenation of pks/sigs)
- better: one signature, independently of the number of signers
- even better: one public key, independently of the number of signers
- difficulty: rogue-key attacks (*plain public-key model*: no CA)

$(sk_A, pk_A) \longrightarrow pk = \mathtt{KeyAgg}(pk_A, pk_B) \longleftarrow (sk_B, pk_B)$

σ $\qquad \mathtt{Ver}(pk, m, \sigma) = 1 \qquad$ σ

Elliptic curves

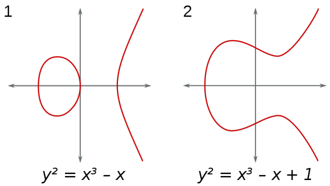
- defined over a finite field
- points on the curve can be added
  $\Rightarrow$ group $\mathbb{G}$
- order p, generator G
- $nG = \underbrace{G + \cdots + G}_{n \text{ times}}$
- can be computed in $O(\log n)$ time
  (*double-and-add*)



- discrete logarithm problem:
  Given $H \in \mathbb{G}$, find $n \in \{0, \dots, p-1\}$ such that $H = nG$

Elliptic curves

- defined over a finite field
- points on the curve can be added
  $\Rightarrow$ group $\mathbb{G}$
- order p, generator G
- $nG = \underbrace{G + \cdots + G}_{n \text{ times}}$
- can be computed in $O(\log n)$ time (*double-and-add*)



- discrete logarithm problem:
  Given $H \in \mathbb{G}$, find $n \in \{0, \ldots, p-1\}$ such that $H = nG$

History of discrete log-based signature schemes

- 1984: ElGamal signatures
- 1985: Elliptic Curve Cryptography proposed by Koblitz and Miller
- 1989: Schnorr signatures, U.S. Patent 4,995,082
- 1991: DSA (*Digital Signature Algorithm*) proposed by NIST
- 1992: ECDSA (*Elliptic Curve DSA*) proposed by Vanstone
- 1993: DSA standardized by NIST as FIPS 186
- 2000: ECDSA included in FIPS 186-2
- 2008: Schnorr's patent expires
- 2009: Bitcoin is launched; uses ECDSA

C.P. Schnorr

Schnorr signatures [Sch90, Sch91]

- secret key: $x \leftarrow_{\$} \mathbb{Z}_p$ public key: $X = xG$
- signature:

$$r \leftarrow_{\$} \mathbb{Z}_p \qquad R := rG$$
$$s := r + H(X, R, m)x \mod p$$
$$\sigma := (R, s)$$

- verification:

$$sG \stackrel{?}{=} R + H(X, R, m)X$$

- provably secure under the DL assumption in the random oracle model for H [PS00]

Schnorr signatures [Sch90, Sch91]

- secret key: $x \leftarrow_\$ \mathbb{Z}_p$ public key: $X = xG$
- signature:

$$r \leftarrow_\$ \mathbb{Z}_p \qquad R := rG$$
$$s := r + H(X, R, m)x \mod p$$
$$\sigma := (R, s)$$

- verification:

$$sG \stackrel{?}{=} R + H(X, R, m)X$$

- provably secure under the DL assumption in the random oracle model for H [PS00]

Schnorr signatures [Sch90, Sch91]

- secret key: $x \leftarrow_{\$} \mathbb{Z}_p$ public key: $X = xG$
- signature:

$$
\begin{aligned}
& r \leftarrow_{\$} \mathbb{Z}_p \qquad R := rG \\
& s := r + H(X, R, m)x \mod p \\
& \sigma := (R, s)
\end{aligned}
$$

- verification:

$$sG \stackrel{?}{=} R + H(X, R, m)X$$

- provably secure under the DL assumption in the random oracle model for H [PS00]

Schnorr signatures [Sch90, Sch91]

- secret key: $x \leftarrow_{\$} \mathbb{Z}_p$ public key: $X = xG$
- signature:

$$r \leftarrow_{\$} \mathbb{Z}_p \qquad R := rG$$
$$s := r + H(X, R, m)x \mod p$$
$$\sigma := (R, s)$$

- verification:

$$sG \stackrel{?}{=} R + H(X, R, m)X$$

- provably secure under the DL assumption in the random oracle model for H [PS00]

Schnorr signatures [Sch90, Sch91]

- secret key: $x \leftarrow_\$ \mathbb{Z}_p$ public key: $X = xG$
- signature:

$$r \leftarrow_\$ \mathbb{Z}_p \qquad R := rG$$
$$s := r + H(X, R, m)x \mod p$$
$$\sigma := (R, s)$$

- verification:

$$sG \stackrel{?}{=} R + H(X, R, m)X$$

- provably secure under the DL assumption in the random oracle model for H [PS00]

“Naive” Schnorr multi-signatures

$X_A = x_A G$

$X_B = x_B G$

- rogue-key attack: Bob sets $X_B = xG - X_A$
  $\Rightarrow X = xG$ and Bob can compute signatures without Alice

"Naive" Schnorr multi-signatures

$$X_A = x_A G \longrightarrow X = X_A + X_B = (x_A + x_B)G \longleftarrow X_B = x_B G$$

- rogue-key attack: Bob sets $X_B = xG - X_A$
  $\Rightarrow X = xG$ and Bob can compute signatures without Alice

"Naive" Schnorr multi-signatures

$X_A = x_A G \longrightarrow X = X_A + X_B = (x_A + x_B)G \longleftarrow X_B = x_B G$

$$R_A = r_A G \longrightarrow$$

$$\longleftarrow R_B = r_B G$$

$R = R_A + R_B$

- rogue-key attack: Bob sets $X_B = xG - X_A$
  $\Rightarrow X = xG$ and Bob can compute signatures without Alice

"Naive" Schnorr multi-signatures

$X_A = x_A G \rightarrow X = X_A + X_B = (x_A + x_B)G \leftarrow X_B = x_B G$

$$R_A = r_A G \longrightarrow$$

$$\longleftarrow R_B = r_B G$$

$$R = R_A + R_B$$

$s_A = r_A + H(X, R, m)x_A \rightarrow s = s_A + s_B \leftarrow s_B = r_B + H(X, R, m)x_B$

- rogue-key attack: Bob sets $X_B = xG - X_A$
  $\Rightarrow X = xG$ and Bob can compute signatures without Alice

"Naive" Schnorr multi-signatures

$X_A = x_A G \rightarrow X = X_A + X_B = (x_A + x_B)G \leftarrow X_B = x_B G$

$$R_A = r_A G \; (\text{Alice} \rightarrow \text{Bob})$$

$$R_B = r_B G \; (\text{Bob} \rightarrow \text{Alice})$$

$$R = R_A + R_B$$

$s_A = r_A + H(X, R, m)x_A \rightarrow s = s_A + s_B \leftarrow s_B = r_B + H(X, R, m)x_B$

$$\texttt{Ver}(X, m, (R, s)) = 1$$

- rogue-key attack: Bob sets $X_B = xG - X_A$
  $\Rightarrow X = xG$ and Bob can compute signatures without Alice

“Naive” Schnorr multi-signatures

$X_A = x_A G \rightarrow X = X_A + X_B = (x_A + x_B)G \leftarrow X_B = x_B G$

$R_A = r_A G \longrightarrow$

$\longleftarrow R_B = r_B G$

$R = R_A + R_B$

$s_A = r_A + H(X, R, m)x_A \rightarrow s = s_A + s_B \leftarrow s_B = r_B + H(X, R, m)x_B$

$\mathtt{Ver}(X, m, (R, s)) = 1$

- rogue-key attack: Bob sets $X_B = xG - X_A$
  $\Rightarrow X = xG$ and Bob can compute signatures without Alice

Schnorr multi-signatures: MuSig [MPSW19, BDN18]

- "delinearized" aggregate key → thwarts rogue-key attacks

$$X = \quad X_A + \quad X_B$$

- partial signature $s_A = r_A + \mu_A H(X, R, m) x_A$
- improves efficiency (n-of-n multisig: 1 pk, 1 sig)
- improves privacy (n-of-n multisig output indistinguishable from "standard" single sig output)
- could be extended to multi-input transactions

Schnorr multi-signatures: MuSig [MPSW19, BDN18]

- "delinearized" aggregate key → thwarts rogue-key attacks

$$X = \mu_A X_A + \mu_B X_B$$
$$\mu_A = H(\{X_A, X_B\}, 1), \qquad \mu_B = H(\{X_A, X_B\}, 2)$$

- partial signature $s_A = r_A + \mu_A H(X, R, m) x_A$
- improves efficiency (n-of-n multisig: 1 pk, 1 sig)
- improves privacy (n-of-n multisig output indistinguishable from "standard" single sig output)
- could be extended to multi-input transactions

Schnorr multi-signatures: MuSig [MPSW19, BDN18]

- "delinearized" aggregate key $\rightarrow$ thwarts rogue-key attacks

$$X = \mu_A X_A + \mu_B X_B$$
$$\mu_A = H(\{X_A, X_B\}, 1), \qquad \mu_B = H(\{X_A, X_B\}, 2)$$

- partial signature $s_A = r_A + \mu_A H(X, R, m) x_A$
- improves efficiency (n-of-n multisig: 1 pk, 1 sig)
- improves privacy (n-of-n multisig output indistinguishable from "standard" single sig output)
- could be extended to multi-input transactions

Schnorr multi-signatures: MuSig [MPSW19, BDN18]

- "delinearized" aggregate key → thwarts rogue-key attacks

$$X = \mu_A X_A + \mu_B X_B$$
$$\mu_A = H(\{X_A, X_B\}, 1), \qquad \mu_B = H(\{X_A, X_B\}, 2)$$

- partial signature $s_A = r_A + \mu_A H(X, R, m) x_A$
- improves efficiency (n-of-n multisig: 1 pk, 1 sig)
- improves privacy (n-of-n multisig output indistinguishable from "standard" single sig output)
- could be extended to multi-input transactions

Schnorr multi-signatures: MuSig [MPSW19, BDN18]

- “delinearized” aggregate key $\rightarrow$ thwarts rogue-key attacks

$$X = \mu_A X_A + \mu_B X_B$$
$$\mu_A = H(\{X_A, X_B\}, 1), \qquad \mu_B = H(\{X_A, X_B\}, 2)$$

- partial signature $s_A = r_A + \mu_A H(X, R, m) x_A$
- improves efficiency (n-of-n multisig: 1 pk, 1 sig)
- improves privacy (n-of-n multisig output indistinguishable from “standard” single sig output)
- could be extended to multi-input transactions

(Non-interactive) aggregate signatures

- similar to multi-signatures but for different messages
- Schnorr signatures: requires interaction
- possible using pairings: $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_t$ such that

$$e(aX, bY) = e(X, Y)^{ab}$$

- BLS signatures [BLS01, BGLS03]:
  - secret key: $x \leftarrow_{\$} \mathbb{Z}_p$ public key: $X = xG$
  - signature: $\sigma = xH(m)$ $(H : \{0,1\}^* \rightarrow \mathbb{G}_2)$
  - verification: $e(G, \sigma) \stackrel{?}{=} e(X, H(m))$
- aggregation can be done publicly after signatures have been computed (e.g. by miners)
- would allow to aggregate all signatures in the blockchain into a single one

(Non-interactive) aggregate signatures

- similar to multi-signatures but for different messages
- Schnorr signatures: requires interaction
- possible using pairings: $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_t$ such that

$$e(aX, bY) = e(X, Y)^{ab}$$

- BLS signatures [BLS01, BGLS03]:
  - secret key: $x \leftarrow_\$ \mathbb{Z}_p$ public key: $X = xG$
  - signature: $\sigma = xH(m)$ $(H : \{0,1\}^* \to \mathbb{G}_2)$
  - verification: $e(G, \sigma) \stackrel{?}{=} e(X, H(m))$
- aggregation can be done publicly after signatures have been computed (e.g. by miners)
- would allow to aggregate all signatures in the blockchain into a single one

(Non-interactive) aggregate signatures

- similar to multi-signatures but for different messages
- Schnorr signatures: requires interaction
- possible using pairings: $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_t$ such that

$$e(aX, bY) = e(X, Y)^{ab}$$

- BLS signatures [BLS01, BGLS03]:
  - secret key: $x \leftarrow_\$ \mathbb{Z}_p$ public key: $X = xG$
  - signature: $\sigma = xH(m)$ $(H : \{0,1\}^* \to \mathbb{G}_2)$
  - verification: $e(G, \sigma) \stackrel{?}{=} e(X, H(m))$
- aggregation can be done publicly after signatures have been computed (e.g. by miners)
- would allow to aggregate all signatures in the blockchain into a single one

(Non-interactive) aggregate signatures

- similar to multi-signatures but for different messages
- Schnorr signatures: requires interaction
- possible using pairings: $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_t$ such that

$$e(aX, bY) = e(X, Y)^{ab}$$

- BLS signatures [BLS01, BGLS03]:
  - secret key: $x \leftarrow_\$ \mathbb{Z}_p$ public key: $X = xG$
  - signature: $\sigma = xH(m)$ $(H : \{0,1\}^* \to \mathbb{G}_2)$
  - verification: $e(G, \sigma) \stackrel{?}{=} e(X, H(m))$
- aggregation can be done publicly after signatures have been computed (e.g. by miners)
- would allow to aggregate all signatures in the blockchain into a single one

(Non-interactive) aggregate signatures

- similar to multi-signatures but for different messages
- Schnorr signatures: requires interaction
- possible using pairings: $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_t$ such that

$$e(aX, bY) = e(X, Y)^{ab}$$

- BLS signatures [BLS01, BGLS03]:
  - secret key: $x \leftarrow_\$ \mathbb{Z}_p$ public key: $X = xG$
  - signature: $\sigma = xH(m)$ $(H : \{0,1\}^* \to \mathbb{G}_2)$
  - verification: $e(G, \sigma) \stackrel{?}{=} e(X, H(m))$
- aggregation can be done publicly after signatures have been computed (e.g. by miners)
- would allow to aggregate all signatures in the blockchain into a single one

(Non-interactive) aggregate signatures

- similar to multi-signatures but for different messages
- Schnorr signatures: requires interaction
- possible using pairings: $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_t$ such that

$$e(aX, bY) = e(X, Y)^{ab}$$

- BLS signatures [BLS01, BGLS03]:
  - secret key: $x \leftarrow_{\$} \mathbb{Z}_p$ public key: $X = xG$
  - signature: $\sigma = xH(m)$ $(H : \{0,1\}^* \to \mathbb{G}_2)$
  - verification: $e(G, \sigma) \stackrel{?}{=} e(X, H(m))$
- aggregation can be done publicly after signatures have been computed (e.g. by miners)
- would allow to aggregate all signatures in the blockchain into a single one

(Non-interactive) aggregate signatures

- similar to multi-signatures but for different messages
- Schnorr signatures: requires interaction
- possible using pairings: $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_t$ such that

$$e(aX, bY) = e(X, Y)^{ab}$$

- BLS signatures [BLS01, BGLS03]:
    - secret key: $x \leftarrow_{\$} \mathbb{Z}_p$ public key: $X = xG$
    - signature: $\sigma = xH(m)$ $(H : \{0,1\}^* \to \mathbb{G}_2)$
    - verification: $e(G, \sigma) \stackrel{?}{=} e(X, H(m))$
- aggregation can be done publicly after signatures have been computed (e.g. by miners)
- would allow to aggregate all signatures in the blockchain into a single one

(Non-interactive) aggregate signatures

- similar to multi-signatures but for different messages
- Schnorr signatures: requires interaction
- possible using pairings: $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_t$ such that

$$e(aX, bY) = e(X, Y)^{ab}$$

- BLS signatures [BLS01, BGLS03]:
    - secret key: $x \leftarrow_\$ \mathbb{Z}_p$ public key: $X = xG$
    - signature: $\sigma = xH(m)$ $(H : \{0,1\}^* \rightarrow \mathbb{G}_2)$
    - verification: $e(G, \sigma) \stackrel{?}{=} e(X, H(m))$
- aggregation can be done publicly after signatures have been computed (e.g. by miners)
- would allow to aggregate all signatures in the blockchain into a single one

(Non-interactive) aggregate signatures

- similar to multi-signatures but for different messages
- Schnorr signatures: requires interaction
- possible using pairings: $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_t$ such that

$$e(aX, bY) = e(X, Y)^{ab}$$

- BLS signatures [BLS01, BGLS03]:
    - secret key: $x \leftarrow_{\$} \mathbb{Z}_p$ public key: $X = xG$
    - signature: $\sigma = xH(m)$ $(H : \{0,1\}^* \rightarrow \mathbb{G}_2)$
    - verification: $e(G, \sigma) \stackrel{?}{=} e(X, H(m))$
- aggregation can be done publicly after signatures have been computed (e.g. by miners)
- would allow to aggregate all signatures in the blockchain into a single one

The end...

Thanks for your attention!

Comments or questions?

References I

Dan Boneh, Manu Drijvers, and Gregory Neven. Compact multi-signatures for smaller blockchains. In *ASIACRYPT 2018, Part II*, pages 435–464.

Dan Boneh, Craig Gentry, Ben Lynn, and Hovav Shacham. Aggregate and verifiably encrypted signatures from bilinear maps. In *EUROCRYPT 2003*, pages 416–432.

Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the Weil pairing. In *ASIACRYPT 2001*, pages 514–532.

Gregory Maxwell, Andrew Poelstra, Yannick Seurin, and Pieter Wuille. Simple Schnorr multi-signatures with applications to Bitcoin. *Designs, Codes and Cryptography*, 2019.

David Pointcheval and Jacques Stern. Security arguments for digital signatures and blind signatures. *Journal of Cryptology*, 13(3):361–396, June 2000.

Claus-Peter Schnorr. Efficient identification and signatures for smart cards. In *CRYPTO'89*, pages 239–252.

References II

Claus-Peter Schnorr. Efficient signature generation by smart cards. *Journal of Cryptology*, 4(3):161–174, January 1991.