Efficiency and Privacy Improvements for Bitcoin with Schnorr Signatures

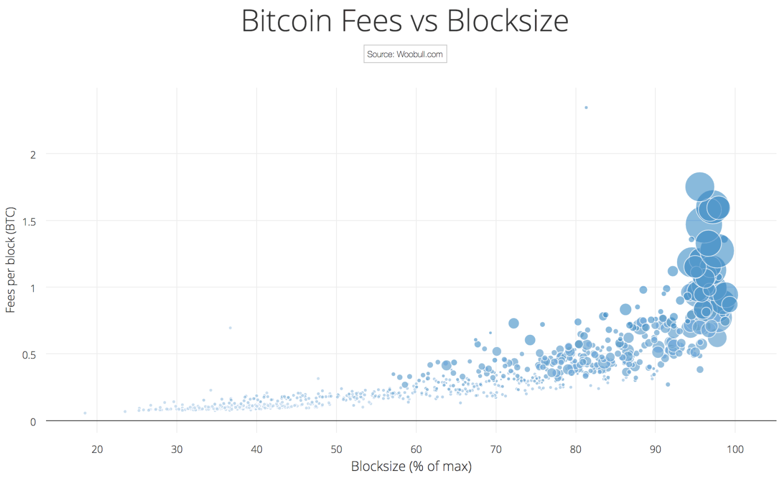
Yannick Seurin

(Based on joint work with
G. Maxwell, A. Poelstra, and P. Wuille)

Agence nationale de la sécurité des systèmes d'information

March 14, 2018 — Cryptofinance Seminar

Motivation: scalability problems



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

Signatures in Bitcoin

- to spend an output, users must provide a signature proving ownership
- spending a P2PKH output requires one signature
- spending a m-of-n multisig output (P2MS or P2SH) requires m signatures (and n public keys)
- signature data ⇒ transaction data ⇒ transaction fees (BTC/byte)
- typical size of an ECDSA signature over secp256k1 (two 32-bytes integers + 6 bytes DER encoding) = 72 bytes
- 300 000 000 transactions in the blockchain, $\sim$ 2 inputs/tx ⇒ at least 54 GB of signature data (28% blockchain size)
- could we use less/smaller signatures without harming security?

Signatures in Bitcoin

- to spend an output, users must provide a signature proving ownership
- spending a P2PKH output requires one signature
- spending a m-of-n multisig output (P2MS or P2SH) requires m signatures (and n public keys)
- signature data $\Rightarrow$ transaction data $\Rightarrow$ transaction fees (BTC/byte)
- typical size of an ECDSA signature over secp256k1 (two 32-bytes integers + 6 bytes DER encoding) = 72 bytes
- 300 000 000 transactions in the blockchain, $\sim$ 2 inputs/tx
  $\Rightarrow$ at least 54 GB of signature data (28% blockchain size)
- could we use less/smaller signatures without harming security?

Signatures in Bitcoin

- to spend an output, users must provide a signature proving ownership
- spending a P2PKH output requires one signature
- spending a m-of-n multisig output (P2MS or P2SH) requires m signatures (and n public keys)
- signature data ⇒ transaction data ⇒ transaction fees (BTC/byte)
- typical size of an ECDSA signature over secp256k1 (two 32-bytes integers + 6 bytes DER encoding) = 72 bytes
- 300 000 000 transactions in the blockchain, $\sim$ 2 inputs/tx ⇒ at least 54 GB of signature data (28% blockchain size)
- could we use less/smaller signatures without harming security?

Signatures in Bitcoin

- to spend an output, users must provide a signature proving ownership
- spending a P2PKH output requires one signature
- spending a m-of-n multisig output (P2MS or P2SH) requires m signatures (and n public keys)
- signature data $\Rightarrow$ transaction data $\Rightarrow$ transaction fees (BTC/byte)
- typical size of an ECDSA signature over secp256k1 (two 32-bytes integers + 6 bytes DER encoding) = 72 bytes
- 300 000 000 transactions in the blockchain, $\sim$ 2 inputs/tx
  $\Rightarrow$ at least 54 GB of signature data (28% blockchain size)
- could we use less/smaller signatures without harming security?

Signatures in Bitcoin

- to spend an output, users must provide a signature proving ownership
- spending a P2PKH output requires one signature
- spending a m-of-n multisig output (P2MS or P2SH) requires m signatures (and n public keys)
- signature data ⇒ transaction data ⇒ transaction fees (BTC/byte)
- typical size of an ECDSA signature over secp256k1 (two 32-bytes integers + 6 bytes DER encoding) = 72 bytes
- 300 000 000 transactions in the blockchain, ∼ 2 inputs/tx ⇒ at least 54 GB of signature data (28% blockchain size)
- could we use less/smaller signatures without harming security?

Signatures in Bitcoin

- to spend an output, users must provide a signature proving ownership
- spending a P2PKH output requires one signature
- spending a *m*-of-*n* multisig output (P2MS or P2SH) requires *m* signatures (and *n* public keys)
- signature data ⇒ transaction data ⇒ transaction fees (BTC/byte)
- typical size of an ECDSA signature over secp256k1 (two 32-bytes integers + 6 bytes DER encoding) = 72 bytes
- 300 000 000 transactions in the blockchain, ∼ 2 inputs/tx ⇒ at least 54 GB of signature data (28% blockchain size)
- could we use less/smaller signatures without harming security?

Signatures in Bitcoin

- to spend an output, users must provide a signature proving ownership
- spending a P2PKH output requires one signature
- spending a m-of-n multisig output (P2MS or P2SH) requires m signatures (and n public keys)
- signature data $\Rightarrow$ transaction data $\Rightarrow$ transaction fees (BTC/byte)
- typical size of an ECDSA signature over secp256k1 (two 32-bytes integers + 6 bytes DER encoding) = 72 bytes
- 300 000 000 transactions in the blockchain, $\sim$ 2 inputs/tx
  $\Rightarrow$ at least 54 GB of signature data (28% blockchain size)
- could we use less/smaller signatures without harming security?

MuSig: Schnorr-based multi-signatures

Simple Schnorr Multi-Signatures
with Applications to Bitcoin

Gregory Maxwell, Andrew Poelstra[1], Yannick Seurin[2], and Pieter Wuille[1]

[1] Blockstream
[2] ANSSI, Paris, France
greg@xiph.org,
{apoelstra, pwuille}@blockstream.com,
yannick.seurin@m4x.org

January 15, 2018

Abstract. We describe a new Schnorr-based multi-signature scheme (i.e., a protocol which allows a group of signers to produce a short, joint signature on a common message), provably secure in the plain public-key model (meaning that signers are only required to have a public key, but do not have to prove knowledge of the private key corresponding to their public key to some certification authority or to other signers before engaging the protocol), which improves over the state-of-art scheme of Bellare and Neven (ACM-CCS 2006) and its variants by Bagherzandi *et al.* (ACM-CCS 2008) and Ma *et al.* (Des. Codes Cryptogr., 2010) in two respects: (i) it is simple and efficient, having only two rounds of communication instead of three for the Bellare-Neven scheme and the same key and signature size as standard Schnorr signatures; (ii) it allows *key aggregation*, which informally means that the joint signature can

https://eprint.iacr.org/2018/068.pdf

Outline

Outline

Digital Signature Schemes

Signature and Key Aggregation

Other Applications

Conclusion

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

C.P. Schnorr

Signature scheme: definition

A signature scheme consists of three algorithms:

1. key generation algorithm $\texttt{Gen}$:
   - 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:

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

Signature scheme: definition

A signature scheme consists of three algorithms:

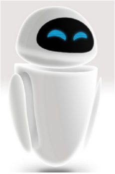
1. key generation algorithm $\texttt{Gen}$:
   - returns a public/secret key pair (pk, sk)
2. signature algorithm $\texttt{Sign}$:
   - takes as input a secret key sk and a message m
   - returns a signature σ
3. verification algorithm $\texttt{Ver}$:
   - takes as input a public key pk, a message m,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

Correctness property:

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

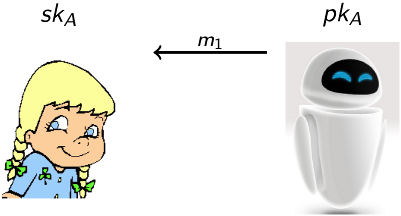
Signature scheme: security

sk_A pk_A



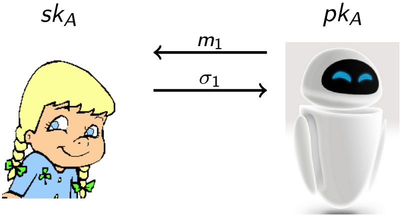
- "gold" security notion: Existential Unforgeability against Chosen Message Attacks (EUF-CMA)
- strong-EUF-CMA: $(m^*, \sigma^*) \neq (m_1, \sigma_1), \ldots, (m_q, \sigma_q)$
- strong-EUF-CMA $\Leftrightarrow$ EUF-CMA + non-malleability

Signature scheme: security



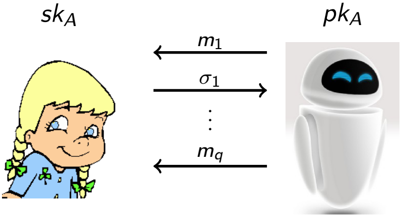
- "gold" security notion: Existential Unforgeability against Chosen Message Attacks (EUF-CMA)
- strong-EUF-CMA: $(m^*, \sigma^*) \neq (m_1, \sigma_1), \ldots, (m_q, \sigma_q)$
- strong-EUF-CMA $\Leftrightarrow$ EUF-CMA + non-malleability

Signature scheme: security



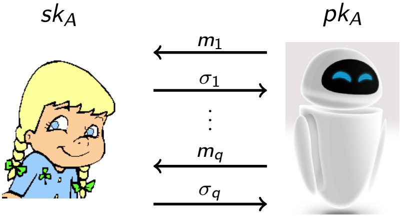
- "gold" security notion: Existential Unforgeability against Chosen Message Attacks (EUF-CMA)
- strong-EUF-CMA: $(m^*, \sigma^*) \neq (m_1, \sigma_1), \ldots, (m_q, \sigma_q)$
- strong-EUF-CMA $\Leftrightarrow$ EUF-CMA + non-malleability

Signature scheme: security



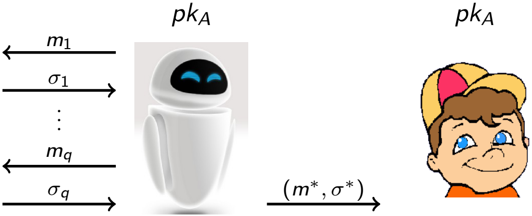
- "gold" security notion: Existential Unforgeability against Chosen Message Attacks (EUF-CMA)
- strong-EUF-CMA: $(m^*, \sigma^*) \neq (m_1, \sigma_1), \ldots, (m_q, \sigma_q)$
- strong-EUF-CMA $\Leftrightarrow$ EUF-CMA + non-malleability

Signature scheme: security



- "gold" security notion: Existential Unforgeability against Chosen Message Attacks (EUF-CMA)
- strong-EUF-CMA: $(m^*, \sigma^*) \neq (m_1, \sigma_1), \ldots, (m_q, \sigma_q)$
- strong-EUF-CMA $\Leftrightarrow$ EUF-CMA + non-malleability

Signature scheme: security



- “gold” security notion: Existential Unforgeability against Chosen Message Attacks (EUF-CMA)
- strong-EUF-CMA: $(m^*, \sigma^*) \neq (m_1, \sigma_1), \ldots, (m_q, \sigma_q)$
- strong-EUF-CMA $\Leftrightarrow$ EUF-CMA + non-malleability

Signature scheme: security

pk_A

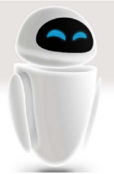
$\leftarrow m_1$

$\sigma_1 \rightarrow$

$\vdots$

$\leftarrow m_q$

$\sigma_q \rightarrow$



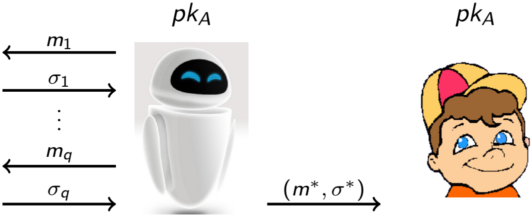
$(m^*, \sigma^*) \rightarrow$

pk_A

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

- "gold" security notion: Existential Unforgeability against Chosen Message Attacks (EUF-CMA)
- strong-EUF-CMA: $(m^*, \sigma^*) \neq (m_1, \sigma_1), \ldots, (m_q, \sigma_q)$
- strong-EUF-CMA $\Leftrightarrow$ EUF-CMA + non-malleability

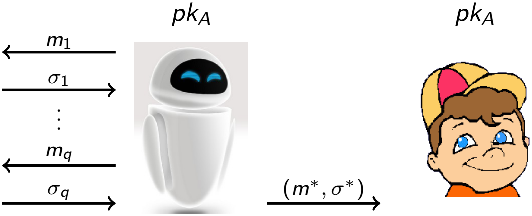
Signature scheme: security



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

- "gold" security notion: Existential Unforgeability against Chosen Message Attacks (EUF-CMA)
- strong-EUF-CMA: $(m^*, \sigma^*) \neq (m_1, \sigma_1), \ldots, (m_q, \sigma_q)$
- strong-EUF-CMA $\Leftrightarrow$ EUF-CMA + non-malleability

Signature scheme: security



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

- "gold" security notion: Existential Unforgeability against Chosen Message Attacks (EUF-CMA)
- strong-EUF-CMA: $(m^*, \sigma^*) \neq (m_1, \sigma_1), \ldots, (m_q, \sigma_q)$
- strong-EUF-CMA $\Leftrightarrow$ EUF-CMA + non-malleability

Provable security

- security proof = proving that breaking a cryptosystem is at least as hard as solving a hard problem P (factoring, discrete log, etc.)
- one assumes there exists an algorithm $\mathcal{A}$ breaking the cryptosystem
- one builds an algorithm solving P using $\mathcal{A}$ as an oracle
- also called reduction (solving P is reduced to breaking the cryptosystem)

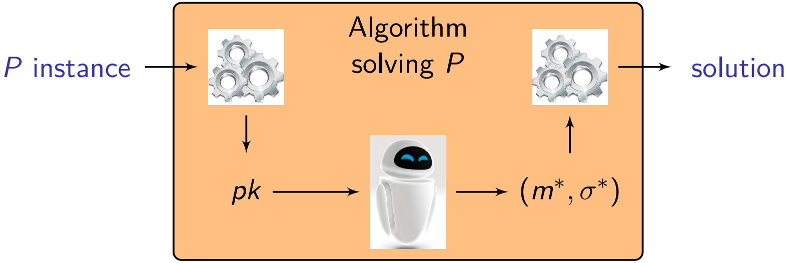
Provable security

- security proof = proving that breaking a cryptosystem is at least as hard as solving a hard problem P (factoring, discrete log, etc.)
- one assumes there exists an algorithm $\mathcal{A}$ breaking the cryptosystem
- one builds an algorithm solving P using $\mathcal{A}$ as an oracle
- also called reduction (solving P is reduced to breaking the cryptosystem)

$pk \longrightarrow$  $\longrightarrow (m^*, \sigma^*)$

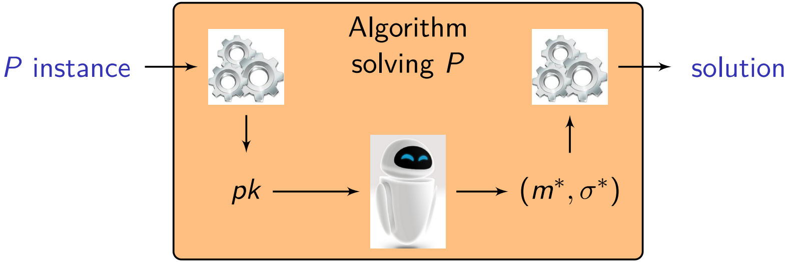
Provable security

- security proof = proving that breaking a cryptosystem is at least as hard as solving a hard problem P (factoring, discrete log, etc.)
- one assumes there exists an algorithm $\mathcal{A}$ breaking the cryptosystem
- one builds an algorithm solving P using $\mathcal{A}$ as an oracle
- also called reduction (solving P is reduced to breaking the cryptosystem)



Provable security

- security proof = proving that breaking a cryptosystem is at least as hard as solving a hard problem P (factoring, discrete log, etc.)
- one assumes there exists an algorithm $\mathcal{A}$ breaking the cryptosystem
- one builds an algorithm solving P using $\mathcal{A}$ as an oracle
- also called reduction (solving P is reduced to breaking the cryptosystem)



Mathematical background (1/2)

Abelian group

An abelian group is a set $\mathbb{G}$ with a binary operation $+ : \mathbb{G} \times \mathbb{G} \to \mathbb{G}$ such that the following holds:

- (associativity): $\forall A, B, C \in \mathbb{G}$, $(A + B) + C = A + (B + C)$
- (identity element): $\exists E \in \mathbb{G}$ such that $E + A = A + E = A$ for all $A \in \mathbb{G}$
- (inverse): $\forall A \in \mathbb{G}$, $\exists B \in \mathbb{G}$ such that $A + B = B + A = E$
- (commutativity): $\forall A, B \in \mathbb{G}$, $A + B = B + A$

Notation: for $n \in \mathbb{N}$, $nA = \underbrace{A + \cdots + A}_{n \text{ times}}$ (with $0A = E$)

Mathematical background (2/2)

Cyclic group and generator

Let $\mathbb{G}$ be an abelian group of order p. An element $G \in \mathbb{G}$ is called a *generator* if

$$\langle G \rangle \stackrel{\text{def}}{=} \{0G, 1G, 2G, \ldots\} = \mathbb{G}.$$

If G is a generator, then for any $X \in \mathbb{G}$, there exists a unique $x \in \{0, \ldots, p-1\}$ such that $X = xG$.

Discrete logarithm problem

Given $X \in \mathbb{G}$, find $x \in \{0, \ldots, p-1\}$ such that $X = xG$.

NB: with multiplicative notation, $xG \sim G^x$

Mathematical background (2/2)

Cyclic group and generator

Let $\mathbb{G}$ be an abelian group of order p. An element $G \in \mathbb{G}$ is called a *generator* if

$$\langle G \rangle \stackrel{\text{def}}{=} \{0G, 1G, 2G, \ldots\} = \mathbb{G}.$$

If G is a generator, then for any $X \in \mathbb{G}$, there exists a unique $x \in \{0, \ldots, p-1\}$ such that $X = xG$.

Discrete logarithm problem

Given $X \in \mathbb{G}$, find $x \in \{0, \ldots, p-1\}$ such that $X = xG$.

NB: with multiplicative notation, $xG \sim G^x$

Schnorr authentication protocol [Sch89, Sch91]

Public parameters: a cyclic group $\mathbb{G}$ of prime order p, a generator G of $\mathbb{G}$

$$\begin{cases} sk_{\text{Alice}} = x \leftarrow_{\$} \mathbb{Z}_p \\ pk_{\text{Alice}} = xG = X \end{cases}$$

$pk_{\text{Alice}} = X$

$r \leftarrow_{\$} \mathbb{Z}_p$, $R = rG$ $\xrightarrow{\quad R \quad}$

$\xleftarrow{\quad c \quad}$ $c \leftarrow_{\$} \mathbb{Z}_p$

$s = r + cx \bmod p$ $\xrightarrow{\quad s \quad}$ Check $sG \stackrel{?}{=} R + cX$

Schnorr's protocol is a "proof of knowledge"

Theorem

Schnorr's protocol is secure against impersonation under the discrete logarithm assumption.

Proof.

- assume there exists an attacker $\mathcal{A}$ which is able to authenticate with good probability
- we run $\mathcal{A}$ on public key X: it sends $R = rG$, we answer with c_1, and $\mathcal{A}$ returns the correct answer $s_1 = r + c_1 x \bmod p$
- we rewind $\mathcal{A}$ and run it again: it sends $R = rG$, we answer with $c_2 \neq c_1$, and $\mathcal{A}$ returns the correct answer $s_2 = r + c_2 x \bmod p$
- we compute $x = (s_1 - s_2)(c_1 - c_2)^{-1} \bmod p$ □

The Fiat-Shamir transform [FS86]

- it is easy to obtain a valid transcript (R, c, s) without knowledge of the secret key x by computing "backwards":
  - choose $s \leftarrow_{\$} \mathbb{Z}_p$
  - choose $c \leftarrow_{\$} \mathbb{Z}_p$
  - compute $R = sG - cX$
- what convinces Bob is that he knows that c was chosen after R was committed by Alice
- how could we make the protocol non-interactive?
- answer: replace the verifier (Bob) by a hash function H
- Alice computes the challenge by herself as $c = H(X, R)$
- assuming H "behaves randomly", this can be proved secure (random oracle model)

The Fiat-Shamir transform [FS86]

- it is easy to obtain a valid transcript (R, c, s) without knowledge of the secret key x by computing "backwards":
  - choose $s \leftarrow_{\$} \mathbb{Z}_p$
  - choose $c \leftarrow_{\$} \mathbb{Z}_p$
  - compute $R = sG - cX$
- what convinces Bob is that he knows that c was chosen after R was committed by Alice
- how could we make the protocol non-interactive?
- answer: replace the verifier (Bob) by a hash function H
- Alice computes the challenge by herself as $c = H(X, R)$
- assuming H "behaves randomly", this can be proved secure (random oracle model)

The Fiat-Shamir transform [FS86]

- it is easy to obtain a valid transcript (R, c, s) without knowledge of the secret key x by computing "backwards":
  - choose $s \leftarrow_{\$} \mathbb{Z}_p$
  - choose $c \leftarrow_{\$} \mathbb{Z}_p$
  - compute $R = sG - cX$
- what convinces Bob is that he knows that c was chosen after R was committed by Alice
- how could we make the protocol non-interactive?
- answer: replace the verifier (Bob) by a hash function H
- Alice computes the challenge by herself as $c = H(X, R)$
- assuming H "behaves randomly", this can be proved secure (random oracle model)

The Fiat-Shamir transform [FS86]

- it is easy to obtain a valid transcript (R, c, s) without knowledge of the secret key x by computing "backwards":
  - choose $s \leftarrow_{\$} \mathbb{Z}_p$
  - choose $c \leftarrow_{\$} \mathbb{Z}_p$
  - compute $R = sG - cX$
- what convinces Bob is that he knows that c was chosen after R was committed by Alice
- how could we make the protocol non-interactive?
- answer: replace the verifier (Bob) by a hash function H
- Alice computes the challenge by herself as $c = H(X, R)$
- assuming H "behaves randomly", this can be proved secure (random oracle model)

The Fiat-Shamir transform [FS86]

- it is easy to obtain a valid transcript (R, c, s) without knowledge of the secret key x by computing "backwards":
  - choose $s \leftarrow_{\$} \mathbb{Z}_p$
  - choose $c \leftarrow_{\$} \mathbb{Z}_p$
  - compute $R = sG - cX$
- what convinces Bob is that he knows that c was chosen after R was committed by Alice
- how could we make the protocol non-interactive?
- answer: replace the verifier (Bob) by a hash function H
- Alice computes the challenge by herself as $c = H(X, R)$
- assuming H "behaves randomly", this can be proved secure (random oracle model)

The Fiat-Shamir transform [FS86]

- it is easy to obtain a valid transcript (R, c, s) without knowledge of the secret key x by computing "backwards":
  - choose $s \leftarrow_{\$} \mathbb{Z}_p$
  - choose $c \leftarrow_{\$} \mathbb{Z}_p$
  - compute $R = sG - cX$
- what convinces Bob is that he knows that c was chosen after R was committed by Alice
- how could we make the protocol non-interactive?
- answer: replace the verifier (Bob) by a hash function H
- Alice computes the challenge by herself as $c = H(X, R)$
- assuming H "behaves randomly", this can be proved secure (random oracle model)

The Fiat-Shamir transform [FS86]

- it is easy to obtain a valid transcript (R, c, s) without knowledge of the secret key x by computing "backwards":
    - choose $s \leftarrow_{\$} \mathbb{Z}_p$
    - choose $c \leftarrow_{\$} \mathbb{Z}_p$
    - compute $R = sG - cX$
- what convinces Bob is that he knows that c was chosen after R was committed by Alice
- how could we make the protocol non-interactive?
- answer: replace the verifier (Bob) by a hash function H
- Alice computes the challenge by herself as $c = H(X, R)$
- assuming H "behaves randomly", this can be proved secure (random oracle model)

The Fiat-Shamir transform [FS86]

- it is easy to obtain a valid transcript (R, c, s) without knowledge of the secret key x by computing "backwards":
    - choose $s \leftarrow_{\$} \mathbb{Z}_p$
    - choose $c \leftarrow_{\$} \mathbb{Z}_p$
    - compute $R = sG - cX$
- what convinces Bob is that he knows that c was chosen after R was committed by Alice
- how could we make the protocol non-interactive?
- answer: replace the verifier (Bob) by a hash function H
- Alice computes the challenge by herself as $c = H(X, R)$
- assuming H "behaves randomly", this can be proved secure (random oracle model)

The Fiat-Shamir transform [FS86]

- it is easy to obtain a valid transcript (R, c, s) without knowledge of the secret key x by computing "backwards":
  - choose $s \leftarrow_{\$} \mathbb{Z}_p$
  - choose $c \leftarrow_{\$} \mathbb{Z}_p$
  - compute $R = sG - cX$
- what convinces Bob is that he knows that c was chosen after R was committed by Alice
- how could we make the protocol non-interactive?
- answer: replace the verifier (Bob) by a hash function H
- Alice computes the challenge by herself as $c = H(X, R)$
- assuming H "behaves randomly", this can be proved secure (random oracle model)

Schnorr signatures [Sch89, Sch91]

- public parameters:
  - a cyclic group $\mathbb{G}$ of prime order p and a generator G
  - a hash function H
- key generation:
  - secret key $x \leftarrow_\$ \mathbb{Z}_p$
  - public key $X = xG$
- signature: on input m and x,
  - draw $r \leftarrow_\$ \mathbb{Z}_p$ and compute $R = rG$
  - compute $c = H(X, R, m)$ and $s = r + cx \bmod p$
  - output $\sigma = (R, s)$
- verification: on input X, m and $\sigma = (R, s)$,
  - compute $c = H(X, R, m)$ and check $sG \stackrel{?}{=} R + cX$
- alternative:
  - signature $\sigma = (c, s)$
  - verification: compute $R = sG - cX$ and check $H(X, R, m) \stackrel{?}{=} c$

Schnorr signatures [Sch89, Sch91]

- public parameters:
  - a cyclic group $\mathbb{G}$ of prime order p and a generator G
  - a hash function H
- key generation:
  - secret key $x \leftarrow_\$ \mathbb{Z}_p$
  - public key $X = xG$
- signature: on input m and x,
  - draw $r \leftarrow_\$ \mathbb{Z}_p$ and compute $R = rG$
  - compute $c = H(X, R, m)$ and $s = r + cx \bmod p$
  - output $\sigma = (R, s)$
- verification: on input X, m and $\sigma = (R, s)$,
  - compute $c = H(X, R, m)$ and check $sG \stackrel{?}{=} R + cX$
- alternative:
  - signature $\sigma = (c, s)$
  - verification: compute $R = sG - cX$ and check $H(X, R, m) \stackrel{?}{=} c$

Schnorr signatures [Sch89, Sch91]

- public parameters:
  - a cyclic group $\mathbb{G}$ of prime order p and a generator G
  - a hash function H
- key generation:
  - secret key $x \leftarrow_{\$} \mathbb{Z}_p$
  - public key $X = xG$
- signature: on input m and x,
  - draw $r \leftarrow_{\$} \mathbb{Z}_p$ and compute $R = rG$
  - compute $c = H(X, R, m)$ and $s = r + cx \bmod p$
  - output $\sigma = (R, s)$
- verification: on input X, m and $\sigma = (R, s)$,
  - compute $c = H(X, R, m)$ and check $sG \stackrel{?}{=} R + cX$
- alternative:
  - signature $\sigma = (c, s)$
  - verification: compute $R = sG - cX$ and check $H(X, R, m) \stackrel{?}{=} c$

Schnorr signatures [Sch89, Sch91]

- public parameters:
  - a cyclic group $\mathbb{G}$ of prime order p and a generator G
  - a hash function H
- key generation:
  - secret key $x \leftarrow_\$ \mathbb{Z}_p$
  - public key $X = xG$
- signature: on input m and x,
  - draw $r \leftarrow_\$ \mathbb{Z}_p$ and compute $R = rG$
  - compute $c = H(X, R, m)$ and $s = r + cx \bmod p$
  - output $\sigma = (R, s)$
- verification: on input X, m and $\sigma = (R, s)$,
  - compute $c = H(X, R, m)$ and check $sG \stackrel{?}{=} R + cX$
- alternative:
  - signature $\sigma = (c, s)$
  - verification: compute $R = sG - cX$ and check $H(X, R, m) \stackrel{?}{=} c$

Schnorr signatures [Sch89, Sch91]

- public parameters:
  - a cyclic group $\mathbb{G}$ of prime order p and a generator G
  - a hash function H
- key generation:
  - secret key $x \leftarrow_{\$} \mathbb{Z}_p$
  - public key $X = xG$
- signature: on input m and x,
  - draw $r \leftarrow_{\$} \mathbb{Z}_p$ and compute $R = rG$
  - compute $c = H(X, R, m)$ and $s = r + cx \bmod p$
  - output $\sigma = (R, s)$
- verification: on input X, m and $\sigma = (R, s)$,
  - compute $c = H(X, R, m)$ and check $sG \stackrel{?}{=} R + cX$
- alternative:
  - signature $\sigma = (c, s)$
  - verification: compute $R = sG - cX$ and check $H(X, R, m) \stackrel{?}{=} c$

Public key-prefixing and BIP 32 (HD wallets)

- Schnorr signatures usually don't include the public key in the hash, i.e., the challenge is $c = H(R, m)$ rather than $c = H(X, R, m)$
- Schnorr signatures without key-prefixing are secure in the strong-EUF-CMA model
- BIP 32 (Hierarchical Deterministic wallets) allows to generate child key pairs from a master key pair $(x, X = xG)$ as

$$x_i = x + H'(i, X) \bmod p, \qquad X_i = X + H'(i, X)G$$

- without key-prefixing, any signature (R, s) valid under X can be turned into a valid signature for X_i: since $c = H(R, m)$,

$$sG = R + cX \Rightarrow (s + cH'(i, X))G = R + cX_i$$

- key-prefixing avoids this problem

Public key-prefixing and BIP 32 (HD wallets)

- Schnorr signatures usually don't include the public key in the hash, i.e., the challenge is $c = H(R, m)$ rather than $c = H(X, R, m)$
- Schnorr signatures without key-prefixing are secure in the strong-EUF-CMA model
- BIP 32 (Hierarchical Deterministic wallets) allows to generate child key pairs from a master key pair $(x, X = xG)$ as

$$x_i = x + H'(i, X) \bmod p, \qquad X_i = X + H'(i, X)G$$

- without key-prefixing, any signature (R, s) valid under X can be turned into a valid signature for X_i: since $c = H(R, m)$,

$$sG = R + cX \Rightarrow (s + cH'(i, X))G = R + cX_i$$

- key-prefixing avoids this problem

Public key-prefixing and BIP 32 (HD wallets)

- Schnorr signatures usually don't include the public key in the hash, i.e., the challenge is $c = H(R, m)$ rather than $c = H(X, R, m)$
- Schnorr signatures without key-prefixing are secure in the strong-EUF-CMA model
- BIP 32 (Hierarchical Deterministic wallets) allows to generate child key pairs from a master key pair $(x, X = xG)$ as

$$x_i = x + H'(i, X) \bmod p, \qquad X_i = X + H'(i, X)G$$

- without key-prefixing, any signature (R, s) valid under X can be turned into a valid signature for X_i: since $c = H(R, m)$,

$$sG = R + cX \Rightarrow (s + cH'(i, X))G = R + cX_i$$

- key-prefixing avoids this problem

Public key-prefixing and BIP 32 (HD wallets)

- Schnorr signatures usually don't include the public key in the hash, i.e., the challenge is $c = H(R, m)$ rather than $c = H(X, R, m)$
- Schnorr signatures without key-prefixing are secure in the strong-EUF-CMA model
- BIP 32 (Hierarchical Deterministic wallets) allows to generate child key pairs from a master key pair $(x, X = xG)$ as

$$x_i = x + H'(i, X) \bmod p, \qquad X_i = X + H'(i, X)G$$

- without key-prefixing, any signature (R, s) valid under X can be turned into a valid signature for X_i: since $c = H(R, m)$,

$$sG = R + cX \Rightarrow (s + cH'(i, X))G = R + cX_i$$

- key-prefixing avoids this problem

Public key-prefixing and BIP 32 (HD wallets)

- Schnorr signatures usually don't include the public key in the hash, i.e., the challenge is $c = H(R, m)$ rather than $c = H(X, R, m)$
- Schnorr signatures without key-prefixing are secure in the strong-EUF-CMA model
- BIP 32 (Hierarchical Deterministic wallets) allows to generate child key pairs from a master key pair $(x, X = xG)$ as

$$x_i = x + H'(i, X) \bmod p, \qquad X_i = X + H'(i, X)G$$

- without key-prefixing, any signature (R, s) valid under X can be turned into a valid signature for X_i: since $c = H(R, m)$,

$$sG = R + cX \Rightarrow (s + cH'(i, X))G = R + cX_i$$

- key-prefixing avoids this problem

"Generic" DSA signatures

- public parameters:
  - a cyclic group $\mathbb{G}$ of prime order p and a generator G
  - a hash function H
  - a "conversion" function $f : \mathbb{G} \to \mathbb{Z}_p$
- key generation:
  - secret key $x \leftarrow_\$ \mathbb{Z}_p$
  - public key $X = xG$
- signature: on input m and x,
  - draw $r \leftarrow_\$ \mathbb{Z}_p$ and compute $R = rG$
  - compute $c = f(R)$ and $s = r^{-1}(H(m) + cx) \bmod p$
  - output $\sigma = (c, s)$
- verification: on input X, m and $\sigma = (c, s)$,
  - compute $u = H(m)s^{-1} \bmod p$, $v = cs^{-1} \bmod p$, and $R = uG + vX$
  - check whether $f(R) \stackrel{?}{=} c$

"Generic" DSA signatures

- public parameters:
  - a cyclic group $\mathbb{G}$ of prime order p and a generator G
  - a hash function H
  - a "conversion" function $f : \mathbb{G} \to \mathbb{Z}_p$
- key generation:
  - secret key $x \leftarrow_{\$} \mathbb{Z}_p$
  - public key $X = xG$
- signature: on input m and x,
  - draw $r \leftarrow_{\$} \mathbb{Z}_p$ and compute $R = rG$
  - compute $c = f(R)$ and $s = r^{-1}(H(m) + cx) \bmod p$
  - output $\sigma = (c, s)$
- verification: on input X, m and $\sigma = (c, s)$,
  - compute $u = H(m)s^{-1} \bmod p$, $v = cs^{-1} \bmod p$, and $R = uG + vX$
  - check whether $f(R) \stackrel{?}{=} c$

"Generic" DSA signatures

- public parameters:
  - a cyclic group $\mathbb{G}$ of prime order p and a generator G
  - a hash function H
  - a "conversion" function $f : \mathbb{G} \to \mathbb{Z}_p$
- key generation:
  - secret key $x \leftarrow_{\$} \mathbb{Z}_p$
  - public key $X = xG$
- signature: on input m and x,
  - draw $r \leftarrow_{\$} \mathbb{Z}_p$ and compute $R = rG$
  - compute $c = f(R)$ and $s = r^{-1}(H(m) + cx) \bmod p$
  - output $\sigma = (c, s)$
- verification: on input X, m and $\sigma = (c, s)$,
  - compute $u = H(m)s^{-1} \bmod p$, $v = cs^{-1} \bmod p$, and $R = uG + vX$
  - check whether $f(R) \stackrel{?}{=} c$

"Generic" DSA signatures

- public parameters:
  - a cyclic group $\mathbb{G}$ of prime order p and a generator G
  - a hash function H
  - a "conversion" function $f : \mathbb{G} \to \mathbb{Z}_p$
- key generation:
  - secret key $x \leftarrow_{\$} \mathbb{Z}_p$
  - public key $X = xG$
- signature: on input m and x,
  - draw $r \leftarrow_{\$} \mathbb{Z}_p$ and compute $R = rG$
  - compute $c = f(R)$ and $s = r^{-1}(H(m) + cx) \bmod p$
  - output $\sigma = (c, s)$
- verification: on input X, m and $\sigma = (c, s)$,
  - compute $u = H(m)s^{-1} \bmod p$, $v = cs^{-1} \bmod p$, and $R = uG + vX$
  - check whether $f(R) \stackrel{?}{=} c$

(EC)DSA signatures

DSA and ECDSA are instantiations of the "generic" DSA scheme:

- for DSA:
  - $\mathbb{G}$ = cyclic subgroup of prime order p of $\mathbb{Z}_q^*$ for some large prime q ($|q| \geq 3072$ bits)
  - conversion function: $f(X) = X \bmod p$
- for ECDSA:
  - $\mathbb{G}$ = cyclic subgroup of prime order p of an elliptic curve group over some finite field ($\mathbb{F}_q$ for q prime or $q = 2^n$)
  - for q prime, group elements are pairs of integers $(x, y) \in \mathbb{F}_q^2$ satisfying the curve equation $E : y^2 = x^3 + ax + b$
  - conversion function: $f(X) = x \bmod p$ where $X = (x, y)$
  - Bitcoin uses curve **secp256k1** [SEC10] (not a NIST curve!) (Standards for Efficient Cryptography, Koblitz curve over prime field $\mathbb{F}_q$ where $q = 2^{256} - 2^{32} - 977$, $a = 0$, $b = 7$)
- Schnorr can be based on any group where DL is hard, in part. on any secure elliptic curve group (Ed25519 [BDL+11]?)

(EC)DSA signatures

DSA and ECDSA are instantiations of the "generic" DSA scheme:

- for DSA:
  - $\mathbb{G}$ = cyclic subgroup of prime order p of $\mathbb{Z}_q^*$ for some large prime q ($|q| \geq 3072$ bits)
  - conversion function: $f(X) = X \bmod p$
- for ECDSA:
  - $\mathbb{G}$ = cyclic subgroup of prime order p of an elliptic curve group over some finite field ($\mathbb{F}_q$ for q prime or $q = 2^n$)
  - for q prime, group elements are pairs of integers $(x, y) \in \mathbb{F}_q^2$ satisfying the curve equation $E : y^2 = x^3 + ax + b$
  - conversion function: $f(X) = x \bmod p$ where $X = (x, y)$
  - Bitcoin uses curve **secp256k1** [SEC10] (not a NIST curve!) (Standards for Efficient Cryptography, Koblitz curve over prime field $\mathbb{F}_q$ where $q = 2^{256} - 2^{32} - 977$, $a = 0$, $b = 7$)
- Schnorr can be based on any group where DL is hard, in part. on any secure elliptic curve group (Ed25519 [BDL+11]?)

(EC)DSA signatures

DSA and ECDSA are instantiations of the "generic" DSA scheme:

- for DSA:
  - $\mathbb{G}$ = cyclic subgroup of prime order p of $\mathbb{Z}_q^*$ for some large prime q ($|q| \geq 3072$ bits)
  - conversion function: $f(X) = X \bmod p$
- for ECDSA:
  - $\mathbb{G}$ = cyclic subgroup of prime order p of an elliptic curve group over some finite field ($\mathbb{F}_q$ for q prime or $q = 2^n$)
  - for q prime, group elements are pairs of integers $(x, y) \in \mathbb{F}_q^2$ satisfying the curve equation $E : y^2 = x^3 + ax + b$
  - conversion function: $f(X) = x \bmod p$ where $X = (x, y)$
  - Bitcoin uses curve **secp256k1** [SEC10] (not a NIST curve!) (Standards for Efficient Cryptography, Koblitz curve over prime field $\mathbb{F}_q$ where $q = 2^{256} - 2^{32} - 977$, $a = 0$, $b = 7$)
- Schnorr can be based on any group where DL is hard, in part. on any secure elliptic curve group (Ed25519 [BDL+11]?)

(EC)DSA signatures

DSA and ECDSA are instantiations of the "generic" DSA scheme:

- for DSA:
  - $\mathbb{G}$ = cyclic subgroup of prime order p of $\mathbb{Z}_q^*$ for some large prime q ($|q| \geq 3072$ bits)
  - conversion function: $f(X) = X \bmod p$
- for ECDSA:
  - $\mathbb{G}$ = cyclic subgroup of prime order p of an elliptic curve group over some finite field ($\mathbb{F}_q$ for q prime or $q = 2^n$)
  - for q prime, group elements are pairs of integers $(x, y) \in \mathbb{F}_q^2$ satisfying the curve equation $E : y^2 = x^3 + ax + b$
  - conversion function: $f(X) = x \bmod p$ where $X = (x, y)$
  - Bitcoin uses curve **secp256k1** [SEC10] (not a NIST curve!) (Standards for Efficient Cryptography, Koblitz curve over prime field $\mathbb{F}_q$ where $q = 2^{256} - 2^{32} - 977$, $a = 0$, $b = 7$)
- Schnorr can be based on any group where DL is hard, in part. on any secure elliptic curve group (Ed25519 [BDL+11]?)

ECDSA signature malleability

- ECDSA is not strongly EUF-CMA
- given a valid signature (c, s) for message m, it is possible to "maul" a different signature which is also valid, namely $(c, -s \bmod p)$
- verification equations:

(c, s)	$(c, -s)$
$u = H(m)s^{-1} \bmod p$	$u' = -H(m)s^{-1} \bmod p = -u$
$v = cs^{-1} \bmod p$	$v' = -cs^{-1} \bmod p = -v$
$R = uG + vX$	$R' = -uG - vX = -R$
$f(R) \stackrel{?}{=} c$	$f(-R) \stackrel{?}{=} c$

- verification succeeds in both cases because:
  - if $R = (x, y)$ then $-R = (x, -y \bmod q)$
  - f only depends on the first coordinate: $f(x, y) = x \bmod p$
- fixed by requiring a canonical "low-s" encoding (Bitcoin PR #6769)

ECDSA signature malleability

- ECDSA is not strongly EUF-CMA
- given a valid signature (c, s) for message m, it is possible to "maul" a different signature which is also valid, namely $(c, -s \bmod p)$
- verification equations:

(c, s)	$(c, -s)$
$u = H(m)s^{-1} \bmod p$	$u' = -H(m)s^{-1} \bmod p = -u$
$v = cs^{-1} \bmod p$	$v' = -cs^{-1} \bmod p = -v$
$R = uG + vX$	$R' = -uG - vX = -R$
$f(R) \stackrel{?}{=} c$	$f(-R) \stackrel{?}{=} c$

- verification succeeds in both cases because:
  - if $R = (x, y)$ then $-R = (x, -y \bmod q)$
  - f only depends on the first coordinate: $f(x, y) = x \bmod p$
- fixed by requiring a canonical "low-s" encoding (Bitcoin PR #6769)

ECDSA signature malleability

- ECDSA is not strongly EUF-CMA
- given a valid signature (c, s) for message m, it is possible to "maul" a different signature which is also valid, namely $(c, -s \bmod p)$
- verification equations:

(c, s)	$(c, -s)$
$u = H(m)s^{-1} \bmod p$	$u' = -H(m)s^{-1} \bmod p = -u$
$v = cs^{-1} \bmod p$	$v' = -cs^{-1} \bmod p = -v$
$R = uG + vX$	$R' = -uG - vX = -R$
$f(R) \stackrel{?}{=} c$	$f(-R) \stackrel{?}{=} c$

- verification succeeds in both cases because:
  - if $R = (x, y)$ then $-R = (x, -y \bmod q)$
  - f only depends on the first coordinate: $f(x, y) = x \bmod p$
- fixed by requiring a canonical "low-s" encoding (Bitcoin PR #6769)

ECDSA signature malleability

- ECDSA is not strongly EUF-CMA
- given a valid signature (c, s) for message m, it is possible to "maul" a different signature which is also valid, namely $(c, -s \bmod p)$
- verification equations:

(c, s)	$(c, -s)$
$u = H(m)s^{-1} \bmod p$	$u' = -H(m)s^{-1} \bmod p = -u$
$v = cs^{-1} \bmod p$	$v' = -cs^{-1} \bmod p = -v$
$R = uG + vX$	$R' = -uG - vX = -R$
$f(R) \stackrel{?}{=} c$	$f(-R) \stackrel{?}{=} c$

- verification succeeds in both cases because:
  - if $R = (x, y)$ then $-R = (x, -y \bmod q)$
  - f only depends on the first coordinate: $f(x, y) = x \bmod p$
- fixed by requiring a canonical "low-s" encoding (Bitcoin PR #6769)

ECDSA signature malleability

- ECDSA is not strongly EUF-CMA
- given a valid signature (c, s) for message m, it is possible to "maul" a different signature which is also valid, namely $(c, -s \bmod p)$
- verification equations:

(c, s)	$(c, -s)$
$u = H(m)s^{-1} \bmod p$	$u' = -H(m)s^{-1} \bmod p = -u$
$v = cs^{-1} \bmod p$	$v' = -cs^{-1} \bmod p = -v$
$R = uG + vX$	$R' = -uG - vX = -R$
$f(R) \stackrel{?}{=} c$	$f(-R) \stackrel{?}{=} c$

- verification succeeds in both cases because:
  - if $R = (x, y)$ then $-R = (x, -y \bmod q)$
  - f only depends on the first coordinate: $f(x, y) = x \bmod p$
- fixed by requiring a canonical "low-s" encoding (Bitcoin PR #6769)

ECDSA signature malleability

- ECDSA is not strongly EUF-CMA
- given a valid signature (c, s) for message m, it is possible to "maul" a different signature which is also valid, namely $(c, -s \bmod p)$
- verification equations:

(c, s)	$(c, -s)$
$u = H(m)s^{-1} \bmod p$	$u' = -H(m)s^{-1} \bmod p = -u$
$v = cs^{-1} \bmod p$	$v' = -cs^{-1} \bmod p = -v$
$R = uG + vX$	$R' = -uG - vX = -R$
$f(R) \stackrel{?}{=} c$	$f(-R) \stackrel{?}{=} c$

- verification succeeds in both cases because:
  - if $R = (x, y)$ then $-R = (x, -y \bmod q)$
  - f only depends on the first coordinate: $f(x, y) = x \bmod p$
- fixed by requiring a canonical "low-s" encoding (Bitcoin PR #6769)

ECDSA signature malleability

- ECDSA is not strongly EUF-CMA
- given a valid signature (c, s) for message m, it is possible to "maul" a different signature which is also valid, namely $(c, -s \bmod p)$
- verification equations:

(c, s)	$(c, -s)$
$u = H(m)s^{-1} \bmod p$	$u' = -H(m)s^{-1} \bmod p = -u$
$v = cs^{-1} \bmod p$	$v' = -cs^{-1} \bmod p = -v$
$R = uG + vX$	$R' = -uG - vX = -R$
$f(R) \stackrel{?}{=} c$	$f(-R) \stackrel{?}{=} c$

- verification succeeds in both cases because:
  - if $R = (x, y)$ then $-R = (x, -y \bmod q)$
  - f only depends on the first coordinate: $f(x, y) = x \bmod p$
- fixed by requiring a canonical "low-s" encoding (Bitcoin PR #6769)

Schnorr versus ECDSA

- Schnorr security:
  - Schnorr signatures have a security proof under the Discrete Logarithm assumption in the Random Oracle Model for H [PS96]
  - no known attacks against Schnorr based on H collisions
- ECDSA security:
  - security analysis of (EC)DSA is much more brittle [Bro05] (uses generic group model, proves non-malleability!)
  - the conversion function f in ECDSA is too "simple" to be realistically modeled as a random oracle
  - collisions on H directly give forgery attacks
- efficiency:
  - Schnorr signatures verification slightly more efficient
  - Schnorr allows efficient batch verification

Schnorr versus ECDSA

- Schnorr security:
  - Schnorr signatures have a security proof under the Discrete Logarithm assumption in the Random Oracle Model for H [PS96]
  - no known attacks against Schnorr based on H collisions
- ECDSA security:
  - security analysis of (EC)DSA is much more brittle [Bro05] (uses generic group model, proves non-malleability!)
  - the conversion function f in ECDSA is too "simple" to be realistically modeled as a random oracle
  - collisions on H directly give forgery attacks
- efficiency:
  - Schnorr signatures verification slightly more efficient
  - Schnorr allows efficient batch verification

Schnorr versus ECDSA

- Schnorr security:
  - Schnorr signatures have a security proof under the Discrete Logarithm assumption in the Random Oracle Model for H [PS96]
  - no known attacks against Schnorr based on H collisions
- ECDSA security:
  - security analysis of (EC)DSA is much more brittle [Bro05] (uses generic group model, proves non-malleability!)
  - the conversion function f in ECDSA is too "simple" to be realistically modeled as a random oracle
  - collisions on H directly give forgery attacks
- efficiency:
  - Schnorr signatures verification slightly more efficient
  - Schnorr allows efficient batch verification

Schnorr versus ECDSA

- Schnorr security:
  - Schnorr signatures have a security proof under the Discrete Logarithm assumption in the Random Oracle Model for H [PS96]
  - no known attacks against Schnorr based on H collisions
- ECDSA security:
  - security analysis of (EC)DSA is much more brittle [Bro05] (uses generic group model, proves non-malleability!)
  - the conversion function f in ECDSA is too "simple" to be realistically modeled as a random oracle
  - collisions on H directly give forgery attacks
- efficiency:
  - Schnorr signatures verification slightly more efficient
  - Schnorr allows efficient batch verification

Schnorr versus ECDSA

- Schnorr security:
  - Schnorr signatures have a security proof under the Discrete Logarithm assumption in the Random Oracle Model for H [PS96]
  - no known attacks against Schnorr based on H collisions
- ECDSA security:
  - security analysis of (EC)DSA is much more brittle [Bro05] (uses generic group model, proves non-malleability!)
  - the conversion function f in ECDSA is too "simple" to be realistically modeled as a random oracle
  - collisions on H directly give forgery attacks
- efficiency:
  - Schnorr signatures verification slightly more efficient
  - Schnorr allows efficient batch verification

Schnorr versus ECDSA

- Schnorr security:
  - Schnorr signatures have a security proof under the Discrete Logarithm assumption in the Random Oracle Model for H [PS96]
  - no known attacks against Schnorr based on H collisions
- ECDSA security:
  - security analysis of (EC)DSA is much more brittle [Bro05] (uses generic group model, proves non-malleability!)
  - the conversion function f in ECDSA is too "simple" to be realistically modeled as a random oracle
  - collisions on H directly give forgery attacks
- efficiency:
  - Schnorr signatures verification slightly more efficient
  - Schnorr allows efficient batch verification

Schnorr versus ECDSA

- Schnorr security:
  - Schnorr signatures have a security proof under the Discrete Logarithm assumption in the Random Oracle Model for H [PS96]
  - no known attacks against Schnorr based on H collisions
- ECDSA security:
  - security analysis of (EC)DSA is much more brittle [Bro05] (uses generic group model, proves non-malleability!)
  - the conversion function f in ECDSA is too "simple" to be realistically modeled as a random oracle
  - collisions on H directly give forgery attacks
- efficiency:
  - Schnorr signatures verification slightly more efficient
  - Schnorr allows efficient batch verification

Schnorr versus ECDSA

- Schnorr security:
  - Schnorr signatures have a security proof under the Discrete Logarithm assumption in the Random Oracle Model for H [PS96]
  - no known attacks against Schnorr based on H collisions
- ECDSA security:
  - security analysis of (EC)DSA is much more brittle [Bro05] (uses generic group model, proves non-malleability!)
  - the conversion function f in ECDSA is too "simple" to be realistically modeled as a random oracle
  - collisions on H directly give forgery attacks
- efficiency:
  - Schnorr signatures verification slightly more efficient
  - Schnorr allows efficient batch verification

Schnorr versus ECDSA

- Schnorr security:
  - Schnorr signatures have a security proof under the Discrete Logarithm assumption in the Random Oracle Model for H [PS96]
  - no known attacks against Schnorr based on H collisions
- ECDSA security:
  - security analysis of (EC)DSA is much more brittle [Bro05] (uses generic group model, proves non-malleability!)
  - the conversion function f in ECDSA is too "simple" to be realistically modeled as a random oracle
  - collisions on H directly give forgery attacks
- efficiency:
  - Schnorr signatures verification slightly more efficient
  - Schnorr allows efficient batch verification

Schnorr versus ECDSA

- Schnorr security:
  - Schnorr signatures have a security proof under the Discrete Logarithm assumption in the Random Oracle Model for H [PS96]
  - no known attacks against Schnorr based on H collisions
- ECDSA security:
  - security analysis of (EC)DSA is much more brittle [Bro05] (uses generic group model, proves non-malleability!)
  - the conversion function f in ECDSA is too “simple” to be realistically modeled as a random oracle
  - collisions on H directly give forgery attacks
- efficiency:
  - Schnorr signatures verification slightly more efficient
  - Schnorr allows efficient batch verification

Schnorr versus ECDSA: Summary

	Schnorr: $\sigma = (R, s)$	ECDSA: $\sigma = (c, s)$
`Ver`	$sG \stackrel{?}{=} R + H(R, m)X$	$f\left(H(m)s^{-1}G + cs^{-1}X\right) \stackrel{?}{=} c$
Fiat-Shamir	✓	×
sec. proof	✓	×
H	2nd preimage	collision
non-mall.	✓	×
batch ver.	✓	×

Reminder:

- computing two signatures with the same r leaks the private key!
- even minor weaknesses in the generation of r can leak the private key after a few hundreds of signatures [NS03]
- practical attacks (Sony PlayStation 3 hack, Android RNG)
- solution: derandomization (RFC 6979)

Schnorr versus ECDSA: Summary

	Schnorr: $\sigma = (R, s)$	ECDSA: $\sigma = (c, s)$
Ver	$sG \stackrel{?}{=} R + H(R, m)X$	$f\left(H(m)s^{-1}G + cs^{-1}X\right) \stackrel{?}{=} c$
Fiat-Shamir	✓	×
sec. proof	✓	×
H	2nd preimage	collision
non-mall.	✓	×
batch ver.	✓	×

Reminder:

- computing two signatures with the same r leaks the private key!
- even minor weaknesses in the generation of r can leak the private key after a few hundreds of signatures [NS03]
- practical attacks (Sony PlayStation 3 hack, Android RNG)
- solution: derandomization (RFC 6979)

Outline

Digital Signature Schemes

Signature and Key Aggregation

Other Applications

Conclusion

What is a multi-signature protocol?

- assume n signers with public keys $\{pk_1, \ldots, pk_n\}$ want to sign the same message (e.g., spending from an n-of-n multisig address)
- trivial solution: compute one signature for each pk_i and output $\Sigma = (\sigma_1, \ldots, \sigma_n)$
- problem: the length of Σ grows linearly with the number of signers. Can we do better? (Ideally, the size of the "multi-signature" should be independent from the number of signers)
- well-studied problem in cryptography originally tackled in [IN83]
- hard to achieve for ECDSA due to its complex algebraic structure (modular division)

What is a multi-signature protocol?

- assume n signers with public keys $\{pk_1, \ldots, pk_n\}$ want to sign the same message (e.g., spending from an n-of-n multisig address)
- trivial solution: compute one signature for each pk_i and output $\Sigma = (\sigma_1, \ldots, \sigma_n)$
- problem: the length of Σ grows linearly with the number of signers. Can we do better? (Ideally, the size of the "multi-signature" should be independent from the number of signers)
- well-studied problem in cryptography originally tackled in [IN83]
- hard to achieve for ECDSA due to its complex algebraic structure (modular division)

What is a multi-signature protocol?

- assume n signers with public keys $\{pk_1, \ldots, pk_n\}$ want to sign the same message (e.g., spending from an n-of-n multisig address)
- trivial solution: compute one signature for each pk_i and output $\Sigma = (\sigma_1, \ldots, \sigma_n)$
- problem: the length of Σ grows linearly with the number of signers. Can we do better? (Ideally, the size of the "multi-signature" should be independent from the number of signers)
- well-studied problem in cryptography originally tackled in [IN83]
- hard to achieve for ECDSA due to its complex algebraic structure (modular division)

What is a multi-signature protocol?

- assume n signers with public keys $\{pk_1, \ldots, pk_n\}$ want to sign the same message (e.g., spending from an n-of-n multisig address)
- trivial solution: compute one signature for each pk_i and output $\Sigma = (\sigma_1, \ldots, \sigma_n)$
- problem: the length of Σ grows linearly with the number of signers. Can we do better? (Ideally, the size of the "multi-signature" should be independent from the number of signers)
- well-studied problem in cryptography originally tackled in [IN83]
- hard to achieve for ECDSA due to its complex algebraic structure (modular division)

What is a multi-signature protocol?

- assume n signers with public keys $\{pk_1, \ldots, pk_n\}$ want to sign the same message (e.g., spending from an n-of-n multisig address)
- trivial solution: compute one signature for each pk_i and output $\Sigma = (\sigma_1, \ldots, \sigma_n)$
- problem: the length of Σ grows linearly with the number of signers. Can we do better? (Ideally, the size of the "multi-signature" should be independent from the number of signers)
- well-studied problem in cryptography originally tackled in [IN83]
- hard to achieve for ECDSA due to its complex algebraic structure (modular division)

"Naive" Schnorr multi-signatures

- Alice's public key $X_1 = x_1 G$, Bob's public key $X_2 = x_2 G$
- signature protocol:
  - Alice draws $R_1 = r_1 G$, Bob draws $R_2 = r_2 G$
  - they both compute $R = R_1 + R_2 = (r_1 + r_2)G$
  - they both compute $c = H(X_1, X_2, R, m)$
  - Alice computes $s_1 = r_1 + cx_1 \bmod p$
  - Bob computes $s_2 = r_2 + cx_2 \bmod p$
  - they both compute $s = s_1 + s_2 \bmod p = (r_1 + r_2) + c(x_1 + x_2) \bmod p$
  - the multi-signature is $\sigma = (R, s)$
- verification: a multi-signature $\sigma = (R, s)$ is valid if

$$sG = R + c \underbrace{(X_1 + X_2)}_{\text{aggregated key } \widetilde{X}} \quad \text{where } c = H(X_1, X_2, R, m)$$

- can be generalized to $n > 2$ signers

"Naive" Schnorr multi-signatures

- Alice's public key $X_1 = x_1 G$, Bob's public key $X_2 = x_2 G$
- signature protocol:
  - Alice draws $R_1 = r_1 G$, Bob draws $R_2 = r_2 G$
  - they both compute $R = R_1 + R_2 = (r_1 + r_2)G$
  - they both compute $c = H(X_1, X_2, R, m)$
  - Alice computes $s_1 = r_1 + cx_1 \bmod p$
  - Bob computes $s_2 = r_2 + cx_2 \bmod p$
  - they both compute $s = s_1 + s_2 \bmod p = (r_1 + r_2) + c(x_1 + x_2) \bmod p$
  - the multi-signature is $\sigma = (R, s)$
- verification: a multi-signature $\sigma = (R, s)$ is valid if

$$sG = R + c \underbrace{(X_1 + X_2)}_{\text{aggregated key } \widetilde{X}} \quad \text{where } c = H(X_1, X_2, R, m)$$

- can be generalized to $n > 2$ signers

"Naive" Schnorr multi-signatures

- Alice's public key $X_1 = x_1 G$, Bob's public key $X_2 = x_2 G$
- signature protocol:
  - Alice draws $R_1 = r_1 G$, Bob draws $R_2 = r_2 G$
  - they both compute $R = R_1 + R_2 = (r_1 + r_2)G$
  - they both compute $c = H(X_1, X_2, R, m)$
  - Alice computes $s_1 = r_1 + cx_1 \bmod p$
  - Bob computes $s_2 = r_2 + cx_2 \bmod p$
  - they both compute $s = s_1 + s_2 \bmod p = (r_1 + r_2) + c(x_1 + x_2) \bmod p$
  - the multi-signature is $\sigma = (R, s)$
- verification: a multi-signature $\sigma = (R, s)$ is valid if

$$sG = R + c \underbrace{(X_1 + X_2)}_{\text{aggregated key } \widetilde{X}} \quad \text{where } c = H(X_1, X_2, R, m)$$

- can be generalized to $n > 2$ signers

"Naive" Schnorr multi-signatures

- Alice's public key $X_1 = x_1 G$, Bob's public key $X_2 = x_2 G$
- signature protocol:
  - Alice draws $R_1 = r_1 G$, Bob draws $R_2 = r_2 G$
  - they both compute $R = R_1 + R_2 = (r_1 + r_2)G$
  - they both compute $c = H(X_1, X_2, R, m)$
  - Alice computes $s_1 = r_1 + cx_1 \bmod p$
  - Bob computes $s_2 = r_2 + cx_2 \bmod p$
  - they both compute $s = s_1 + s_2 \bmod p = (r_1 + r_2) + c(x_1 + x_2) \bmod p$
  - the multi-signature is $\sigma = (R, s)$
- verification: a multi-signature $\sigma = (R, s)$ is valid if

$$sG = R + c \underbrace{(X_1 + X_2)}_{\text{aggregated key } \widetilde{X}} \quad \text{where } c = H(X_1, X_2, R, m)$$

- can be generalized to $n > 2$ signers

"Naive" Schnorr multi-signatures

- Alice's public key $X_1 = x_1 G$, Bob's public key $X_2 = x_2 G$
- signature protocol:
  - Alice draws $R_1 = r_1 G$, Bob draws $R_2 = r_2 G$
  - they both compute $R = R_1 + R_2 = (r_1 + r_2)G$
  - they both compute $c = H(X_1, X_2, R, m)$
  - Alice computes $s_1 = r_1 + cx_1 \bmod p$
  - Bob computes $s_2 = r_2 + cx_2 \bmod p$
  - they both compute $s = s_1 + s_2 \bmod p = (r_1 + r_2) + c(x_1 + x_2) \bmod p$
  - the multi-signature is $\sigma = (R, s)$
- verification: a multi-signature $\sigma = (R, s)$ is valid if

$$sG = R + c \underbrace{(X_1 + X_2)}_{\text{aggregated key } \widetilde{X}} \quad \text{where } c = H(X_1, X_2, R, m)$$

- can be generalized to $n > 2$ signers

"Naive" Schnorr multi-signatures

- Alice's public key $X_1 = x_1 G$, Bob's public key $X_2 = x_2 G$
- signature protocol:
  - Alice draws $R_1 = r_1 G$, Bob draws $R_2 = r_2 G$
  - they both compute $R = R_1 + R_2 = (r_1 + r_2)G$
  - they both compute $c = H(X_1, X_2, R, m)$
  - Alice computes $s_1 = r_1 + cx_1 \bmod p$
  - Bob computes $s_2 = r_2 + cx_2 \bmod p$
  - they both compute $s = s_1 + s_2 \bmod p = (r_1 + r_2) + c(x_1 + x_2) \bmod p$
  - the multi-signature is $\sigma = (R, s)$
- verification: a multi-signature $\sigma = (R, s)$ is valid if

$$sG = R + c \underbrace{(X_1 + X_2)}_{\text{aggregated key } \widetilde{X}} \quad \text{where } c = H(X_1, X_2, R, m)$$

- can be generalized to $n > 2$ signers

"Naive" Schnorr multi-signatures

- Alice's public key $X_1 = x_1 G$, Bob's public key $X_2 = x_2 G$
- signature protocol:
  - Alice draws $R_1 = r_1 G$, Bob draws $R_2 = r_2 G$
  - they both compute $R = R_1 + R_2 = (r_1 + r_2)G$
  - they both compute $c = H(X_1, X_2, R, m)$
  - Alice computes $s_1 = r_1 + cx_1 \bmod p$
  - Bob computes $s_2 = r_2 + cx_2 \bmod p$
  - they both compute $s = s_1 + s_2 \bmod p = (r_1 + r_2) + c(x_1 + x_2) \bmod p$
  - the multi-signature is $\sigma = (R, s)$
- verification: a multi-signature $\sigma = (R, s)$ is valid if

$$sG = R + c \underbrace{(X_1 + X_2)}_{\text{aggregated key } \widetilde{X}} \quad \text{where } c = H(X_1, X_2, R, m)$$

- can be generalized to $n > 2$ signers

"Naive" Schnorr multi-signatures

- Alice's public key $X_1 = x_1 G$, Bob's public key $X_2 = x_2 G$
- signature protocol:
  - Alice draws $R_1 = r_1 G$, Bob draws $R_2 = r_2 G$
  - they both compute $R = R_1 + R_2 = (r_1 + r_2)G$
  - they both compute $c = H(X_1, X_2, R, m)$
  - Alice computes $s_1 = r_1 + cx_1 \bmod p$
  - Bob computes $s_2 = r_2 + cx_2 \bmod p$
  - they both compute $s = s_1 + s_2 \bmod p = (r_1 + r_2) + c(x_1 + x_2) \bmod p$
  - the multi-signature is $\sigma = (R, s)$
- verification: a multi-signature $\sigma = (R, s)$ is valid if

$$sG = R + c \underbrace{(X_1 + X_2)}_{\text{aggregated key } \widetilde{X}} \quad \text{where } c = H(X_1, X_2, R, m)$$

- can be generalized to $n > 2$ signers

"Naive" Schnorr multi-signatures

- Alice's public key $X_1 = x_1 G$, Bob's public key $X_2 = x_2 G$
- signature protocol:
  - Alice draws $R_1 = r_1 G$, Bob draws $R_2 = r_2 G$
  - they both compute $R = R_1 + R_2 = (r_1 + r_2)G$
  - they both compute $c = H(X_1, X_2, R, m)$
  - Alice computes $s_1 = r_1 + cx_1 \bmod p$
  - Bob computes $s_2 = r_2 + cx_2 \bmod p$
  - they both compute $s = s_1 + s_2 \bmod p = (r_1 + r_2) + c(x_1 + x_2) \bmod p$
  - the multi-signature is $\sigma = (R, s)$
- verification: a multi-signature $\sigma = (R, s)$ is valid if

$$sG = R + c \underbrace{(X_1 + X_2)}_{\text{aggregated key } \widetilde{X}} \quad \text{where } c = H(X_1, X_2, R, m)$$

- can be generalized to $n > 2$ signers

"Naive" Schnorr multi-signatures

- Alice's public key $X_1 = x_1 G$, Bob's public key $X_2 = x_2 G$
- signature protocol:
  - Alice draws $R_1 = r_1 G$, Bob draws $R_2 = r_2 G$
  - they both compute $R = R_1 + R_2 = (r_1 + r_2)G$
  - they both compute $c = H(X_1, X_2, R, m)$
  - Alice computes $s_1 = r_1 + cx_1 \bmod p$
  - Bob computes $s_2 = r_2 + cx_2 \bmod p$
  - they both compute $s = s_1 + s_2 \bmod p = (r_1 + r_2) + c(x_1 + x_2) \bmod p$
  - the multi-signature is $\sigma = (R, s)$
- verification: a multi-signature $\sigma = (R, s)$ is valid if

$$sG = R + c \underbrace{(X_1 + X_2)}_{\text{aggregated key } \widetilde{X}} \quad \text{where } c = H(X_1, X_2, R, m)$$

- can be generalized to $n > 2$ signers

"Naive" Schnorr multi-signatures

- Alice's public key $X_1 = x_1 G$, Bob's public key $X_2 = x_2 G$
- signature protocol:
  - Alice draws $R_1 = r_1 G$, Bob draws $R_2 = r_2 G$
  - they both compute $R = R_1 + R_2 = (r_1 + r_2)G$
  - they both compute $c = H(X_1, X_2, R, m)$
  - Alice computes $s_1 = r_1 + cx_1 \bmod p$
  - Bob computes $s_2 = r_2 + cx_2 \bmod p$
  - they both compute $s = s_1 + s_2 \bmod p = (r_1 + r_2) + c(x_1 + x_2) \bmod p$
  - the multi-signature is $\sigma = (R, s)$
- verification: a multi-signature $\sigma = (R, s)$ is valid if

$$sG = R + c \underbrace{(X_1 + X_2)}_{\text{aggregated key } \widetilde{X}} \quad \text{where } c = H(X_1, X_2, R, m)$$

- can be generalized to $n > 2$ signers

Wait! Rogue-key attacks

- assume that signers can claim whatever public key they want (plain public key model)
- Bob knows Alice's public key X_1
- he can "choose" public key $X_2 = x'G - X_1$
- Bob can forge a valid multi-signature (R, s) on his own with x':

$$sG = R + \underbrace{H(X_1, X_2, R, m)}_{c}\underbrace{(X_1 + X_2)}_{x'G} = (r + cx')G$$

- note that Bob does not know the private key for $X_2 = (x' - x_1)G$
- this can thwarted using a key setup procedure [MOR01] or by requiring signers to prove knowledge of their private key (with a zero-knowledge proof) [RY07]

Wait! Rogue-key attacks

- assume that signers can claim whatever public key they want (plain public key model)
- Bob knows Alice's public key X_1
- he can "choose" public key $X_2 = x'G - X_1$
- Bob can forge a valid multi-signature (R, s) on his own with x':

$$sG = R + \underbrace{H(X_1, X_2, R, m)}_{c}\underbrace{(X_1 + X_2)}_{x'G} = (r + cx')G$$

- note that Bob does not know the private key for $X_2 = (x' - x_1)G$
- this can thwarted using a key setup procedure [MOR01] or by requiring signers to prove knowledge of their private key (with a zero-knowledge proof) [RY07]

Wait! Rogue-key attacks

- assume that signers can claim whatever public key they want (plain public key model)
- Bob knows Alice's public key X_1
- he can "choose" public key $X_2 = x'G - X_1$
- Bob can forge a valid multi-signature (R, s) on his own with x':

$$sG = R + \underbrace{H(X_1, X_2, R, m)}_{c}\underbrace{(X_1 + X_2)}_{x'G} = (r + cx')G$$

- note that Bob does not know the private key for $X_2 = (x' - x_1)G$
- this can thwarted using a key setup procedure [MOR01] or by requiring signers to prove knowledge of their private key (with a zero-knowledge proof) [RY07]

Wait! Rogue-key attacks

- assume that signers can claim whatever public key they want (plain public key model)
- Bob knows Alice's public key X_1
- he can "choose" public key $X_2 = x'G - X_1$
- Bob can forge a valid multi-signature (R, s) on his own with x':

$$sG = R + \underbrace{H(X_1, X_2, R, m)}_{c} \underbrace{(X_1 + X_2)}_{x'G} = (r + cx')G$$

- note that Bob does not know the private key for $X_2 = (x' - x_1)G$
- this can thwarted using a key setup procedure [MOR01] or by requiring signers to prove knowledge of their private key (with a zero-knowledge proof) [RY07]

Wait! Rogue-key attacks

- assume that signers can claim whatever public key they want (plain public key model)
- Bob knows Alice's public key X_1
- he can "choose" public key $X_2 = x'G - X_1$
- Bob can forge a valid multi-signature (R, s) on his own with x':

$$sG = R + \underbrace{H(X_1, X_2, R, m)}_{c} \underbrace{(X_1 + X_2)}_{x'G} = (r + cx')G$$

- note that Bob does not know the private key for $X_2 = (x' - x_1)G$
- this can thwarted using a key setup procedure [MOR01] or by requiring signers to prove knowledge of their private key (with a zero-knowledge proof) [RY07]

Wait! Rogue-key attacks

- assume that signers can claim whatever public key they want (plain public key model)
- Bob knows Alice's public key X_1
- he can "choose" public key $X_2 = x'G - X_1$
- Bob can forge a valid multi-signature (R, s) on his own with x':

$$sG = R + \underbrace{H(X_1, X_2, R, m)}_{c} \underbrace{(X_1 + X_2)}_{x'G} = (r + cx')G$$

- note that Bob does not know the private key for $X_2 = (x' - x_1)G$
- this can thwarted using a key setup procedure [MOR01] or by requiring signers to prove knowledge of their private key (with a zero-knowledge proof) [RY07]

Bellare-Neven multi-signature scheme [BN06]

- list of signers pubic keys $L = \{X_1 = x_1 G, \ldots, X_n = x_n G\}$
- signature protocol:
    - each signer draws $R_i = r_i G$
    - all signers compute $R = \sum_{i=1}^{n} R_i = \left(\sum_{i=1}^{n} r_i\right) G$
    - each signer computes a distinct challenge $c_i = H(L, X_i, R, m)$ and a partial signature $s_i = r_i + c_i x_i \bmod p$
    - partial signatures are added: $s = \sum_{i=1}^{n} s_i \bmod p$
    - the multi-signature is $\sigma = (R, s)$
- verification: a multi-signature $\sigma = (R, s)$ is valid if

$$sG = R + \sum_{i=1}^{n} c_i X_i$$

- thwarts rogue-key attacks but key aggregation is not possible...

Bellare-Neven multi-signature scheme [BN06]

- list of signers pubic keys $L = \{X_1 = x_1 G, \ldots, X_n = x_n G\}$
- signature protocol:
  - each signer draws $R_i = r_i G$
  - all signers compute $R = \sum_{i=1}^{n} R_i = \left(\sum_{i=1}^{n} r_i\right) G$
  - each signer computes a distinct challenge $c_i = H(L, X_i, R, m)$ and a partial signature $s_i = r_i + c_i x_i \bmod p$
  - partial signatures are added: $s = \sum_{i=1}^{n} s_i \bmod p$
  - the multi-signature is $\sigma = (R, s)$
- verification: a multi-signature $\sigma = (R, s)$ is valid if

$$sG = R + \sum_{i=1}^{n} c_i X_i$$

- thwarts rogue-key attacks but key aggregation is not possible...

Bellare-Neven multi-signature scheme [BN06]

- list of signers pubic keys $L = \{X_1 = x_1 G, \ldots, X_n = x_n G\}$
- signature protocol:
  - each signer draws $R_i = r_i G$
  - all signers compute $R = \sum_{i=1}^{n} R_i = \left(\sum_{i=1}^{n} r_i\right) G$
  - each signer computes a distinct challenge $c_i = H(L, X_i, R, m)$ and a partial signature $s_i = r_i + c_i x_i \bmod p$
  - partial signatures are added: $s = \sum_{i=1}^{n} s_i \bmod p$
  - the multi-signature is $\sigma = (R, s)$
- verification: a multi-signature $\sigma = (R, s)$ is valid if

$$sG = R + \sum_{i=1}^{n} c_i X_i$$

- thwarts rogue-key attacks but key aggregation is not possible...

Bellare-Neven multi-signature scheme [BN06]

- list of signers pubic keys $L = \{X_1 = x_1 G, \dots, X_n = x_n G\}$
- signature protocol:
  - each signer draws $R_i = r_i G$
  - all signers compute $R = \sum_{i=1}^n R_i = \left(\sum_{i=1}^n r_i\right) G$
  - each signer computes a distinct challenge $c_i = H(L, X_i, R, m)$ and a partial signature $s_i = r_i + c_i x_i \bmod p$
  - partial signatures are added: $s = \sum_{i=1}^n s_i \bmod p$
  - the multi-signature is $\sigma = (R, s)$
- verification: a multi-signature $\sigma = (R, s)$ is valid if

$$sG = R + \sum_{i=1}^{n} c_i X_i$$

- thwarts rogue-key attacks but key aggregation is not possible...

Bellare-Neven multi-signature scheme [BN06]

- list of signers pubic keys $L = \{X_1 = x_1 G, \ldots, X_n = x_n G\}$
- signature protocol:
  - each signer draws $R_i = r_i G$
  - all signers compute $R = \sum_{i=1}^{n} R_i = \left(\sum_{i=1}^{n} r_i\right) G$
  - each signer computes a distinct challenge $c_i = H(L, X_i, R, m)$ and a partial signature $s_i = r_i + c_i x_i \bmod p$
  - partial signatures are added: $s = \sum_{i=1}^{n} s_i \bmod p$
  - the multi-signature is $\sigma = (R, s)$
- verification: a multi-signature $\sigma = (R, s)$ is valid if

$$sG = R + \sum_{i=1}^{n} c_i X_i$$

- thwarts rogue-key attacks but key aggregation is not possible...

Bellare-Neven multi-signature scheme [BN06]

- list of signers pubic keys $L = \{X_1 = x_1 G, \ldots, X_n = x_n G\}$
- signature protocol:
  - each signer draws $R_i = r_i G$
  - all signers compute $R = \sum_{i=1}^{n} R_i = \left(\sum_{i=1}^{n} r_i\right) G$
  - each signer computes a distinct challenge $c_i = H(L, X_i, R, m)$ and a partial signature $s_i = r_i + c_i x_i \bmod p$
  - partial signatures are added: $s = \sum_{i=1}^{n} s_i \bmod p$
  - the multi-signature is $\sigma = (R, s)$
- verification: a multi-signature $\sigma = (R, s)$ is valid if

$$sG = R + \sum_{i=1}^{n} c_i X_i$$

- thwarts rogue-key attacks but key aggregation is not possible...

Bellare-Neven multi-signature scheme [BN06]

- list of signers pubic keys $L = \{X_1 = x_1 G, \ldots, X_n = x_n G\}$
- signature protocol:
  - each signer draws $R_i = r_i G$
  - all signers compute $R = \sum_{i=1}^n R_i = \left(\sum_{i=1}^n r_i\right) G$
  - each signer computes a distinct challenge $c_i = H(L, X_i, R, m)$ and a partial signature $s_i = r_i + c_i x_i \bmod p$
  - partial signatures are added: $s = \sum_{i=1}^n s_i \bmod p$
  - the multi-signature is $\sigma = (R, s)$
- verification: a multi-signature $\sigma = (R, s)$ is valid if

$$sG = R + \sum_{i=1}^{n} c_i X_i$$

- thwarts rogue-key attacks but key aggregation is not possible...

Bellare-Neven multi-signature scheme [BN06]

- list of signers pubic keys $L = \{X_1 = x_1 G, \dots, X_n = x_n G\}$
- signature protocol:
  - each signer draws $R_i = r_i G$
  - all signers compute $R = \sum_{i=1}^n R_i = \left(\sum_{i=1}^n r_i\right) G$
  - each signer computes a distinct challenge $c_i = H(L, X_i, R, m)$ and a partial signature $s_i = r_i + c_i x_i \bmod p$
  - partial signatures are added: $s = \sum_{i=1}^n s_i \bmod p$
  - the multi-signature is $\sigma = (R, s)$
- verification: a multi-signature $\sigma = (R, s)$ is valid if

$$sG = R + \sum_{i=1}^{n} c_i X_i$$

- thwarts rogue-key attacks but key aggregation is not possible...

Bellare-Neven multi-signature scheme [BN06]

- list of signers pubic keys $L = \{X_1 = x_1 G, \ldots, X_n = x_n G\}$
- signature protocol:
  - each signer draws $R_i = r_i G$
  - all signers compute $R = \sum_{i=1}^{n} R_i = \left(\sum_{i=1}^{n} r_i\right) G$
  - each signer computes a distinct challenge $c_i = H(L, X_i, R, m)$ and a partial signature $s_i = r_i + c_i x_i \bmod p$
  - partial signatures are added: $s = \sum_{i=1}^{n} s_i \bmod p$
  - the multi-signature is $\sigma = (R, s)$
- verification: a multi-signature $\sigma = (R, s)$ is valid if

$$sG = R + \sum_{i=1}^{n} c_i X_i$$

- thwarts rogue-key attacks but key aggregation is not possible...

MuSig: key aggregation in the plain public key model

- variant of BN where the challenge for the i-th signer is

$$c_i = \underbrace{H_0(L, X_i)}_{a_i} \underbrace{H_1(\widetilde{X}, R, m)}_{c} \quad \text{where} \quad \widetilde{X} = \sum_{i=1}^{n} H_0(L, X_i) X_i$$

- partial signature $s_i = r_i + c a_i x_i \bmod p$, $s = \sum_{i=1}^{n} s_i \bmod p$
- $\widetilde{X}$ is called the aggregated key
- verification identical to "normal" signature with public key $\widetilde{X}$:

$$sG = R + \sum_{i=1}^{n} c_i X_i = R + \underbrace{H_1(\widetilde{X}, R, m)}_{c} \underbrace{\sum_{i=1}^{n} H_0(L, X_i) X_i}_{\widetilde{X}}$$

- variant with $\widetilde{X} = \sum_{i=1}^{n} H_0(X_i) X_i$ is insecure (Wagner's algorithm)

MuSig: key aggregation in the plain public key model

- variant of BN where the challenge for the i-th signer is

$$c_i = \underbrace{H_0(L, X_i)}_{a_i} \underbrace{H_1(\widetilde{X}, R, m)}_{c} \quad \text{where} \quad \widetilde{X} = \sum_{i=1}^{n} H_0(L, X_i) X_i$$

- partial signature $s_i = r_i + c a_i x_i \bmod p$, $s = \sum_{i=1}^{n} s_i \bmod p$
- $\widetilde{X}$ is called the aggregated key
- verification identical to "normal" signature with public key $\widetilde{X}$:

$$sG = R + \sum_{i=1}^{n} c_i X_i = R + \underbrace{H_1(\widetilde{X}, R, m)}_{c} \underbrace{\sum_{i=1}^{n} H_0(L, X_i) X_i}_{\widetilde{X}}$$

- variant with $\widetilde{X} = \sum_{i=1}^{n} H_0(X_i) X_i$ is insecure (Wagner's algorithm)

MuSig: key aggregation in the plain public key model

- variant of BN where the challenge for the i-th signer is

$$c_i = \underbrace{H_0(L, X_i)}_{a_i}\underbrace{H_1(\widetilde{X}, R, m)}_{c} \quad \text{where} \quad \widetilde{X} = \sum_{i=1}^{n} H_0(L, X_i) X_i$$

- partial signature $s_i = r_i + c a_i x_i \bmod p$, $s = \sum_{i=1}^{n} s_i \bmod p$
- $\widetilde{X}$ is called the aggregated key
- verification identical to "normal" signature with public key $\widetilde{X}$:

$$sG = R + \sum_{i=1}^{n} c_i X_i = R + \underbrace{H_1(\widetilde{X}, R, m)}_{c}\underbrace{\sum_{i=1}^{n} H_0(L, X_i) X_i}_{\widetilde{X}}$$

- variant with $\widetilde{X} = \sum_{i=1}^{n} H_0(X_i) X_i$ is insecure (Wagner's algorithm)

MuSig: key aggregation in the plain public key model

- variant of BN where the challenge for the i-th signer is

$$c_i = \underbrace{H_0(L, X_i)}_{a_i} \underbrace{H_1(\widetilde{X}, R, m)}_{c} \quad \text{where} \quad \widetilde{X} = \sum_{i=1}^{n} H_0(L, X_i) X_i$$

- partial signature $s_i = r_i + c a_i x_i \bmod p$, $s = \sum_{i=1}^{n} s_i \bmod p$
- $\widetilde{X}$ is called the aggregated key
- verification identical to "normal" signature with public key $\widetilde{X}$:

$$sG = R + \sum_{i=1}^{n} c_i X_i = R + \underbrace{H_1(\widetilde{X}, R, m)}_{c} \underbrace{\sum_{i=1}^{n} H_0(L, X_i) X_i}_{\widetilde{X}}$$

- variant with $\widetilde{X} = \sum_{i=1}^{n} H_0(X_i) X_i$ is insecure (Wagner's algorithm)

MuSig: key aggregation in the plain public key model

- variant of BN where the challenge for the i-th signer is

$$c_i = \underbrace{H_0(L, X_i)}_{a_i} \underbrace{H_1(\widetilde{X}, R, m)}_{c} \quad \text{where} \quad \widetilde{X} = \sum_{i=1}^{n} H_0(L, X_i) X_i$$

- partial signature $s_i = r_i + c a_i x_i \bmod p$, $s = \sum_{i=1}^{n} s_i \bmod p$
- $\widetilde{X}$ is called the aggregated key
- verification identical to "normal" signature with public key $\widetilde{X}$:

$$sG = R + \sum_{i=1}^{n} c_i X_i = R + \underbrace{H_1(\widetilde{X}, R, m)}_{c} \underbrace{\sum_{i=1}^{n} H_0(L, X_i) X_i}_{\widetilde{X}}$$

- variant with $\widetilde{X} = \sum_{i=1}^{n} H_0(X_i) X_i$ is insecure (Wagner's algorithm)

Application 1: replacing `OP_CHECKMULTISIG`

- using MuSig, n-of-n multisig outputs can be replaced by standard P2PKH output for the aggregated key $\widetilde{X}$
- this improves privacy
  - individual public keys are never revealed
  - the resulting output is indistinguishable from a standard P2PKH output
- for "threshold" m-of-n multisigs with $m < n$:
  - build a Merkle tree where leaves are all $\binom{n}{m}$ possible aggregated keys and only put the root in the `ScriptPubKey`
  - to spend, give a Merkle proof of membership of some $\widetilde{X}$ and a signature valid for $\widetilde{X}$

Application 2: cross-input signature aggregation

- transaction with multiple inputs: each key signs a different message
- ⇒ Interactive Aggregate Signature (IAS) scheme
- BN proposed to use a multi-signature scheme with message $M = m_1 \| m_2 \| \ldots \| m_n$ (generic conversion MS → IAS)
- insecure in the plain public key model (credit: R. O'Connor):
  - Alice has two outputs O_1 and O_2 (same pub. key $X_a = x_a G$)
  - let m_i be the message for spending O_i
  - Alice wants to spend O_1 (only) in a CoinJoin with Bob
  - Bob claims he has the same key X_a, and chooses as message m_2
  - both Alice and Bob run the protocol on input X_a and $M = m_1 \| m_2$
  - Bob can simply copy Alice's messages (although he does not know private key x_a)
  - both O_1 and O_2 are spent

Application 2: cross-input signature aggregation

- transaction with multiple inputs: each key signs a different message
- ⇒ Interactive Aggregate Signature (IAS) scheme
- BN proposed to use a multi-signature scheme with message $M = m_1 \| m_2 \| \dots \| m_n$ (generic conversion MS → IAS)
- insecure in the plain public key model (credit: R. O'Connor):
  - Alice has two outputs O_1 and O_2 (same pub. key $X_a = x_a G$)
  - let m_i be the message for spending O_i
  - Alice wants to spend O_1 (only) in a CoinJoin with Bob
  - Bob claims he has the same key X_a, and chooses as message m_2
  - both Alice and Bob run the protocol on input X_a and $M = m_1 \| m_2$
  - Bob can simply copy Alice's messages (although he does not know private key x_a)
  - both O_1 and O_2 are spent

Application 2: cross-input signature aggregation

- transaction with multiple inputs: each key signs a different message
- ⇒ Interactive Aggregate Signature (IAS) scheme
- BN proposed to use a multi-signature scheme with message $M = m_1 \| m_2 \| \ldots \| m_n$ (generic conversion MS → IAS)
- insecure in the plain public key model (credit: R. O'Connor):
  - Alice has two outputs O_1 and O_2 (same pub. key $X_a = x_a G$)
  - let m_i be the message for spending O_i
  - Alice wants to spend O_1 (only) in a CoinJoin with Bob
  - Bob claims he has the same key X_a, and chooses as message m_2
  - both Alice and Bob run the protocol on input X_a and $M = m_1 \| m_2$
  - Bob can simply copy Alice's messages (although he does not know private key x_a)
  - both O_1 and O_2 are spent

Application 2: cross-input signature aggregation

- transaction with multiple inputs: each key signs a different message
- $\Rightarrow$ Interactive Aggregate Signature (IAS) scheme
- BN proposed to use a multi-signature scheme with message $M = m_1 \| m_2 \| \ldots \| m_n$ (generic conversion MS $\rightarrow$ IAS)
- insecure in the plain public key model (credit: R. O'Connor):
  - Alice has two outputs O_1 and O_2 (same pub. key $X_a = x_a G$)
  - let m_i be the message for spending O_i
  - Alice wants to spend O_1 (only) in a CoinJoin with Bob
  - Bob claims he has the same key X_a, and chooses as message m_2
  - both Alice and Bob run the protocol on input X_a and $M = m_1 \| m_2$
  - Bob can simply copy Alice's messages (although he does not know private key x_a)
  - both O_1 and O_2 are spent

Application 2: cross-input signature aggregation

- transaction with multiple inputs: each key signs a different message
- ⇒ Interactive Aggregate Signature (IAS) scheme
- BN proposed to use a multi-signature scheme with message $M = m_1 \| m_2 \| \ldots \| m_n$ (generic conversion MS → IAS)
- insecure in the plain public key model (credit: R. O'Connor):
  - Alice has two outputs O_1 and O_2 (same pub. key $X_a = x_a G$)
  - let m_i be the message for spending O_i
  - Alice wants to spend O_1 (only) in a CoinJoin with Bob
  - Bob claims he has the same key X_a, and chooses as message m_2
  - both Alice and Bob run the protocol on input X_a and $M = m_1 \| m_2$
  - Bob can simply copy Alice's messages (although he does not know private key x_a)
  - both O_1 and O_2 are spent

Application 2: cross-input signature aggregation

- transaction with multiple inputs: each key signs a different message
- ⇒ Interactive Aggregate Signature (IAS) scheme
- BN proposed to use a multi-signature scheme with message $M = m_1 \| m_2 \| \ldots \| m_n$ (generic conversion MS → IAS)
- insecure in the plain public key model (credit: R. O'Connor):
  - Alice has two outputs O_1 and O_2 (same pub. key $X_a = x_a G$)
  - let m_i be the message for spending O_i
  - Alice wants to spend O_1 (only) in a CoinJoin with Bob
  - Bob claims he has the same key X_a, and chooses as message m_2
  - both Alice and Bob run the protocol on input X_a and $M = m_1 \| m_2$
  - Bob can simply copy Alice's messages (although he does not know private key x_a)
  - both O_1 and O_2 are spent

Application 2: cross-input signature aggregation

- transaction with multiple inputs: each key signs a different message
- $\Rightarrow$ Interactive Aggregate Signature (IAS) scheme
- BN proposed to use a multi-signature scheme with message $M = m_1 \| m_2 \| \dots \| m_n$ (generic conversion MS $\rightarrow$ IAS)
- insecure in the plain public key model (credit: R. O'Connor):
  - Alice has two outputs O_1 and O_2 (same pub. key $X_a = x_a G$)
  - let m_i be the message for spending O_i
  - Alice wants to spend O_1 (only) in a CoinJoin with Bob
  - Bob claims he has the same key X_a, and chooses as message m_2
  - both Alice and Bob run the protocol on input X_a and $M = m_1 \| m_2$
  - Bob can simply copy Alice's messages (although he does not know private key x_a)
  - both O_1 and O_2 are spent

Application 2: cross-input signature aggregation

- transaction with multiple inputs: each key signs a different message
- $\Rightarrow$ Interactive Aggregate Signature (IAS) scheme
- BN proposed to use a multi-signature scheme with message $M = m_1 \| m_2 \| \ldots \| m_n$ (generic conversion MS $\rightarrow$ IAS)
- insecure in the plain public key model (credit: R. O'Connor):
    - Alice has two outputs O_1 and O_2 (same pub. key $X_a = x_a G$)
    - let m_i be the message for spending O_i
    - Alice wants to spend O_1 (only) in a CoinJoin with Bob
    - Bob claims he has the same key X_a, and chooses as message m_2
    - both Alice and Bob run the protocol on input X_a and $M = m_1 \| m_2$
    - Bob can simply copy Alice's messages (although he does not know private key x_a)
    - both O_1 and O_2 are spent

Application 2: cross-input signature aggregation

- transaction with multiple inputs: each key signs a different message
- ⇒ Interactive Aggregate Signature (IAS) scheme
- BN proposed to use a multi-signature scheme with message $M = m_1 \| m_2 \| \dots \| m_n$ (generic conversion MS → IAS)
- insecure in the plain public key model (credit: R. O'Connor):
  - Alice has two outputs O_1 and O_2 (same pub. key $X_a = x_a G$)
  - let m_i be the message for spending O_i
  - Alice wants to spend O_1 (only) in a CoinJoin with Bob
  - Bob claims he has the same key X_a, and chooses as message m_2
  - both Alice and Bob run the protocol on input X_a and $M = m_1 \| m_2$
  - Bob can simply copy Alice's messages (although he does not know private key x_a)
  - both O_1 and O_2 are spent

Application 2: cross-input signature aggregation

- transaction with multiple inputs: each key signs a different message
- $\Rightarrow$ Interactive Aggregate Signature (IAS) scheme
- BN proposed to use a multi-signature scheme with message $M = m_1 \| m_2 \| \ldots \| m_n$ (generic conversion MS $\rightarrow$ IAS)
- insecure in the plain public key model (credit: R. O'Connor):
    - Alice has two outputs O_1 and O_2 (same pub. key $X_a = x_a G$)
    - let m_i be the message for spending O_i
    - Alice wants to spend O_1 (only) in a CoinJoin with Bob
    - Bob claims he has the same key X_a, and chooses as message m_2
    - both Alice and Bob run the protocol on input X_a and $M = m_1 \| m_2$
    - Bob can simply copy Alice's messages (although he does not know private key x_a)
    - both O_1 and O_2 are spent

Application 2: cross-input signature aggregation

- transaction with multiple inputs: each key signs a different message
- $\Rightarrow$ Interactive Aggregate Signature (IAS) scheme
- BN proposed to use a multi-signature scheme with message $M = m_1 \| m_2 \| \ldots \| m_n$ (generic conversion MS $\to$ IAS)
- insecure in the plain public key model (credit: R. O'Connor):
  - Alice has two outputs O_1 and O_2 (same pub. key $X_a = x_a G$)
  - let m_i be the message for spending O_i
  - Alice wants to spend O_1 (only) in a CoinJoin with Bob
  - Bob claims he has the same key X_a, and chooses as message m_2
  - both Alice and Bob run the protocol on input X_a and $M = m_1 \| m_2$
  - Bob can simply copy Alice's messages (although he does not know private key x_a)
  - both O_1 and O_2 are spent

Application 2: cross-input signature aggregation

- a secure IAS scheme requires "breaking the symmetry" for signers with the same public key
- solution: modify BN to compute the challenge for i-th signer as

$$c_i = H(\{(X_1, m_1), \ldots, (X_n, m_n)\}, R, i)$$

- the previous attack does not work since Alice computes

$$c_1 = H(\{(X_a, m_1), (X_a, m_2)\}, R, 1)$$

whereas Bob must use

$$c_2 = H(\{(X_a, m_1), (X_a, m_2)\}, R, 2)$$

Application 2: cross-input signature aggregation

- a secure IAS scheme requires "breaking the symmetry" for signers with the same public key
- solution: modify BN to compute the challenge for i-th signer as

$$c_i = H(\{(X_1, m_1), \ldots, (X_n, m_n)\}, R, i)$$

- the previous attack does not work since Alice computes

$$c_1 = H(\{(X_a, m_1), (X_a, m_2)\}, R, 1)$$

whereas Bob must use

$$c_2 = H(\{(X_a, m_1), (X_a, m_2)\}, R, 2)$$

Application 2: cross-input signature aggregation

- a secure IAS scheme requires "breaking the symmetry" for signers with the same public key
- solution: modify BN to compute the challenge for i-th signer as

$$c_i = H(\{(X_1, m_1), \ldots, (X_n, m_n)\}, R, i)$$

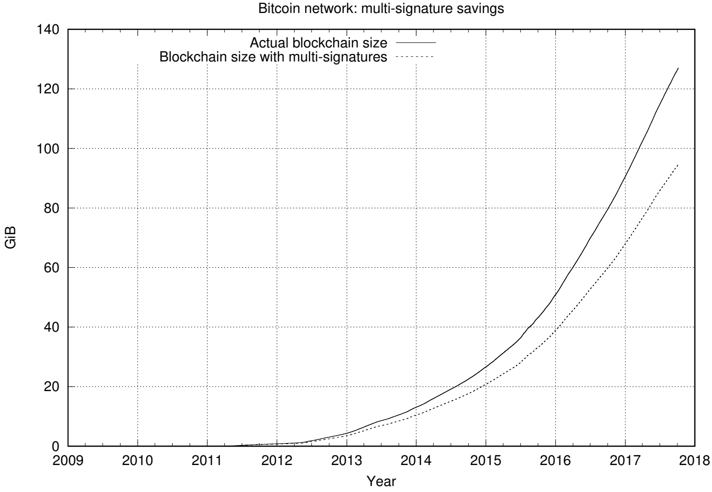
- the previous attack does not work since Alice computes

$$c_1 = H(\{(X_a, m_1), (X_a, m_2)\}, R, 1)$$

whereas Bob must use

$$c_2 = H(\{(X_a, m_1), (X_a, m_2)\}, R, 2)$$

Benefits: Space savings



Benefits: UTXO set consolidation

- actors handling a large number of transactions can end up with a large number of "dust" UTXOs (e.g. exchanges)


LaurentMT @LaurentMT · 21 déc. 2017
For example, this entity (oxt.me/entity/tiid/48…) is a wallet controlled by Coinbase. To date, it owns around 203 BTC split in 1,464,545 utxos ! With BTC at $15.8k, it means $3.2M with an average utxo value of 2.2$. #DustInTheChain

- they become impossible to spend when fees are too high
- cross-input signature aggregation allows to merge them into a single UTXO with a single signature rather than one signature per input ⇒ lower transaction fees

Benefits: UTXO set consolidation

- actors handling a large number of transactions can end up with a large number of "dust" UTXOs (e.g. exchanges)

LaurentMT @LaurentMT · 21 déc. 2017
For example, this entity (oxt.me/entity/tiid/48...) is a wallet controlled by Coinbase. To date, it owns around 203 BTC split in 1,464,545 utxos ! With BTC at $15.8k, it means $3.2M with an average utxo value of 2.2$. #DustInTheChain

- they become impossible to spend when fees are too high
- cross-input signature aggregation allows to merge them into a single UTXO with a single signature rather than one signature per input $\Rightarrow$ lower transaction fees

Benefits: UTXO set consolidation

- actors handling a large number of transactions can end up with a large number of "dust" UTXOs (e.g. exchanges)

LaurentMT @LaurentMT · 21 déc. 2017
For example, this entity (oxt.me/entity/tiid/48…) is a wallet controlled by Coinbase. To date, it owns around 203 BTC split in 1,464,545 utxos ! With BTC at $15.8k, it means $3.2M with an average utxo value of 2.2$. #DustInTheChain

- they become impossible to spend when fees are too high
- cross-input signature aggregation allows to merge them into a single UTXO with a single signature rather than one signature per input $\Rightarrow$ lower transaction fees

Outline

Digital Signature Schemes

Signature and Key Aggregation

Other Applications

Conclusion

Taproot (G. Maxwell)

- conditions for spending an output often of the form

$$\underbrace{(n \text{ parties agree to sign})}_{n\text{-of-}n \text{ multisig}} \text{ OR } \underbrace{(\text{some more complex conditions})}_{\text{script } S}$$

- this can be achieved indistinguishably from a standard P2PKH output
- let $\widetilde{X}$ be the MuSig aggregated key for the n parties
- output uses public key $Y = \widetilde{X} + H(\widetilde{X}, S)G$
- two ways to spend the output:
  - the n parties agree to sign with Y (one of them simply adds a corrective term $cH(\widetilde{X}, S)$ to its partial signature)
  - $\widetilde{X}$ and S are revealed and a ScriptSig S' is provided; the network checks $\widetilde{X} + H(\widetilde{X}, S)G \stackrel{?}{=} Y$ and that $S \| S'$ returns True

Taproot (G. Maxwell)

- conditions for spending an output often of the form

$$\underbrace{(n \text{ parties agree to sign})}_{n\text{-of-}n \text{ multisig}} \text{ OR } \underbrace{(\text{some more complex conditions})}_{\text{script } S}$$

- this can be achieved indistinguishably from a standard P2PKH output
- let $\widetilde{X}$ be the MuSig aggregated key for the n parties
- output uses public key $Y = \widetilde{X} + H(\widetilde{X}, S)G$
- two ways to spend the output:
  - the n parties agree to sign with Y (one of them simply adds a corrective term $cH(\widetilde{X}, S)$ to its partial signature)
  - $\widetilde{X}$ and S are revealed and a `ScriptSig` S' is provided; the network checks $\widetilde{X} + H(\widetilde{X}, S)G \stackrel{?}{=} Y$ and that $S \| S'$ returns True

Taproot (G. Maxwell)

- conditions for spending an output often of the form

$$\underbrace{(n \text{ parties agree to sign})}_{n\text{-of-}n \text{ multisig}} \text{ OR } \underbrace{(\text{some more complex conditions})}_{\text{script } S}$$

- this can be achieved indistinguishably from a standard P2PKH output
- let $\widetilde{X}$ be the MuSig aggregated key for the n parties
- output uses public key $Y = \widetilde{X} + H(\widetilde{X}, S)G$
- two ways to spend the output:
  - the n parties agree to sign with Y (one of them simply adds a corrective term $cH(\widetilde{X}, S)$ to its partial signature)
  - $\widetilde{X}$ and S are revealed and a ScriptSig S' is provided; the network checks $\widetilde{X} + H(\widetilde{X}, S)G \stackrel{?}{=} Y$ and that $S \| S'$ returns True

Taproot (G. Maxwell)

- conditions for spending an output often of the form

$$\underbrace{(n \text{ parties agree to sign})}_{n\text{-of-}n \text{ multisig}} \text{ OR } \underbrace{(\text{some more complex conditions})}_{\text{script } S}$$

- this can be achieved indistinguishably from a standard P2PKH output
- let $\widetilde{X}$ be the MuSig aggregated key for the n parties
- output uses public key $Y = \widetilde{X} + H(\widetilde{X}, S)G$
- two ways to spend the output:
  - the n parties agree to sign with Y (one of them simply adds a corrective term $cH(\widetilde{X}, S)$ to its partial signature)
  - $\widetilde{X}$ and S are revealed and a ScriptSig S' is provided; the network checks $\widetilde{X} + H(\widetilde{X}, S)G \stackrel{?}{=} Y$ and that $S \| S'$ returns True

Taproot (G. Maxwell)

- conditions for spending an output often of the form

$$\underbrace{(n \text{ parties agree to sign})}_{n\text{-of-}n \text{ multisig}} \text{ OR } \underbrace{(\text{some more complex conditions})}_{\text{script } S}$$

- this can be achieved indistinguishably from a standard P2PKH output
- let $\widetilde{X}$ be the MuSig aggregated key for the n parties
- output uses public key $Y = \widetilde{X} + H(\widetilde{X}, S)G$
- two ways to spend the output:
  - the n parties agree to sign with Y (one of them simply adds a corrective term $cH(\widetilde{X}, S)$ to its partial signature)
  - $\widetilde{X}$ and S are revealed and a `ScriptSig` S' is provided; the network checks $\widetilde{X} + H(\widetilde{X}, S)G \stackrel{?}{=} Y$ and that $S \| S'$ returns True

Taproot (G. Maxwell)

- conditions for spending an output often of the form

$$\underbrace{(n \text{ parties agree to sign})}_{n\text{-of-}n \text{ multisig}} \text{ OR } \underbrace{(\text{some more complex conditions})}_{\text{script } S}$$

- this can be achieved indistinguishably from a standard P2PKH output
- let $\widetilde{X}$ be the MuSig aggregated key for the n parties
- output uses public key $Y = \widetilde{X} + H(\widetilde{X}, S)G$
- two ways to spend the output:
  - the n parties agree to sign with Y (one of them simply adds a corrective term $cH(\widetilde{X}, S)$ to its partial signature)
  - $\widetilde{X}$ and S are revealed and a ScriptSig S' is provided; the network checks $\widetilde{X} + H(\widetilde{X}, S)G \stackrel{?}{=} Y$ and that $S \| S'$ returns True

Taproot (G. Maxwell)

- conditions for spending an output often of the form

$$\underbrace{(n \text{ parties agree to sign})}_{n\text{-of-}n \text{ multisig}} \text{ OR } \underbrace{(\text{some more complex conditions})}_{\text{script } S}$$

- this can be achieved indistinguishably from a standard P2PKH output
- let $\widetilde{X}$ be the MuSig aggregated key for the n parties
- output uses public key $Y = \widetilde{X} + H(\widetilde{X}, S)G$
- two ways to spend the output:
  - the n parties agree to sign with Y (one of them simply adds a corrective term $cH(\widetilde{X}, S)$ to its partial signature)
  - $\widetilde{X}$ and S are revealed and a `ScriptSig` S' is provided; the network checks $\widetilde{X} + H(\widetilde{X}, S)G \stackrel{?}{=} Y$ and that $S\|S'$ returns True

Scriptless scripts (A. Poelstra)

- goal: enforce smart contracts without publishing the contract in the blockchain
- relies on "adaptor" signatures:
  - Alice has key pair $(x, X = xG)$
  - Alice draws two ephemeral keys $R = rG$, $T = tG$
  - she computes $s = r + t + H(X, R + T, m)x$ and sends (R, T, s') to Bob where $s' = s - t$
  - Bob can check $s'G \stackrel{?}{=} R + H(X, R + T, m)X$ but can't compute a valid signature for m
  - now revealing signature $s \Leftrightarrow$ revealing t
- t can be some secret value necessary for an auxiliary protocol (correctness can be proved in zero-knowledge from T)
- using a 2-of-2 multisig and an adaptor signature, one can obtain a cross-chain atomic swap protocol indistinguishable from standard spendings on each chain

Scriptless scripts (A. Poelstra)

- goal: enforce smart contracts without publishing the contract in the blockchain
- relies on "adaptor" signatures:
    - Alice has key pair $(x, X = xG)$
    - Alice draws two ephemeral keys $R = rG$, $T = tG$
    - she computes $s = r + t + H(X, R + T, m)x$ and sends (R, T, s') to Bob where $s' = s - t$
    - Bob can check $s'G \stackrel{?}{=} R + H(X, R + T, m)X$ but can't compute a valid signature for m
    - now revealing signature $s \Leftrightarrow$ revealing t
- t can be some secret value necessary for an auxiliary protocol (correctness can be proved in zero-knowledge from T)
- using a 2-of-2 multisig and an adaptor signature, one can obtain a cross-chain atomic swap protocol indistinguishable from standard spendings on each chain

Scriptless scripts (A. Poelstra)

- goal: enforce smart contracts without publishing the contract in the blockchain
- relies on "adaptor" signatures:
  - Alice has key pair $(x, X = xG)$
  - Alice draws two ephemeral keys $R = rG$, $T = tG$
  - she computes $s = r + t + H(X, R + T, m)x$ and sends (R, T, s') to Bob where $s' = s - t$
  - Bob can check $s'G \stackrel{?}{=} R + H(X, R + T, m)X$ but can't compute a valid signature for m
  - now revealing signature $s \Leftrightarrow$ revealing t
- t can be some secret value necessary for an auxiliary protocol (correctness can be proved in zero-knowledge from T)
- using a 2-of-2 multisig and an adaptor signature, one can obtain a cross-chain atomic swap protocol indistinguishable from standard spendings on each chain

Scriptless scripts (A. Poelstra)

- goal: enforce smart contracts without publishing the contract in the blockchain
- relies on "adaptor" signatures:
  - Alice has key pair $(x, X = xG)$
  - Alice draws two ephemeral keys $R = rG$, $T = tG$
  - she computes $s = r + t + H(X, R + T, m)x$ and sends (R, T, s') to Bob where $s' = s - t$
  - Bob can check $s'G \stackrel{?}{=} R + H(X, R + T, m)X$ but can't compute a valid signature for m
  - now revealing signature $s \Leftrightarrow$ revealing t
- t can be some secret value necessary for an auxiliary protocol (correctness can be proved in zero-knowledge from T)
- using a 2-of-2 multisig and an adaptor signature, one can obtain a cross-chain atomic swap protocol indistinguishable from standard spendings on each chain

Scriptless scripts (A. Poelstra)

- goal: enforce smart contracts without publishing the contract in the blockchain
- relies on "adaptor" signatures:
  - Alice has key pair $(x, X = xG)$
  - Alice draws two ephemeral keys $R = rG,\ T = tG$
  - she computes $s = r + t + H(X, R + T, m)x$ and sends (R, T, s') to Bob where $s' = s - t$
  - Bob can check $s'G \stackrel{?}{=} R + H(X, R + T, m)X$ but can't compute a valid signature for m
  - now revealing signature $s \Leftrightarrow$ revealing t
- t can be some secret value necessary for an auxiliary protocol (correctness can be proved in zero-knowledge from T)
- using a 2-of-2 multisig and an adaptor signature, one can obtain a cross-chain atomic swap protocol indistinguishable from standard spendings on each chain

Scriptless scripts (A. Poelstra)

- goal: enforce smart contracts without publishing the contract in the blockchain
- relies on "adaptor" signatures:
  - Alice has key pair $(x, X = xG)$
  - Alice draws two ephemeral keys $R = rG$, $T = tG$
  - she computes $s = r + t + H(X, R + T, m)x$ and sends (R, T, s') to Bob where $s' = s - t$
  - Bob can check $s'G \stackrel{?}{=} R + H(X, R + T, m)X$ but can't compute a valid signature for m
  - now revealing signature $s \Leftrightarrow$ revealing t
- t can be some secret value necessary for an auxiliary protocol (correctness can be proved in zero-knowledge from T)
- using a 2-of-2 multisig and an adaptor signature, one can obtain a cross-chain atomic swap protocol indistinguishable from standard spendings on each chain

Scriptless scripts (A. Poelstra)

- goal: enforce smart contracts without publishing the contract in the blockchain
- relies on "adaptor" signatures:
    - Alice has key pair $(x, X = xG)$
    - Alice draws two ephemeral keys $R = rG$, $T = tG$
    - she computes $s = r + t + H(X, R + T, m)x$ and sends (R, T, s') to Bob where $s' = s - t$
    - Bob can check $s'G \stackrel{?}{=} R + H(X, R + T, m)X$ but can't compute a valid signature for m
    - now revealing signature $s \Leftrightarrow$ revealing t
- t can be some secret value necessary for an auxiliary protocol (correctness can be proved in zero-knowledge from T)
- using a 2-of-2 multisig and an adaptor signature, one can obtain a cross-chain atomic swap protocol indistinguishable from standard spendings on each chain

Scriptless scripts (A. Poelstra)

- goal: enforce smart contracts without publishing the contract in the blockchain
- relies on "adaptor" signatures:
  - Alice has key pair $(x, X = xG)$
  - Alice draws two ephemeral keys $R = rG$, $T = tG$
  - she computes $s = r + t + H(X, R + T, m)x$ and sends (R, T, s') to Bob where $s' = s - t$
  - Bob can check $s'G \stackrel{?}{=} R + H(X, R + T, m)X$ but can't compute a valid signature for m
  - now revealing signature $s \Leftrightarrow$ revealing t
- t can be some secret value necessary for an auxiliary protocol (correctness can be proved in zero-knowledge from T)
- using a 2-of-2 multisig and an adaptor signature, one can obtain a cross-chain atomic swap protocol indistinguishable from standard spendings on each chain

Scriptless scripts (A. Poelstra)

- goal: enforce smart contracts without publishing the contract in the blockchain
- relies on "adaptor" signatures:
  - Alice has key pair $(x, X = xG)$
  - Alice draws two ephemeral keys $R = rG$, $T = tG$
  - she computes $s = r + t + H(X, R + T, m)x$ and sends (R, T, s') to Bob where $s' = s - t$
  - Bob can check $s'G \stackrel{?}{=} R + H(X, R + T, m)X$ but can't compute a valid signature for m
  - now revealing signature $s \Leftrightarrow$ revealing t
- t can be some secret value necessary for an auxiliary protocol (correctness can be proved in zero-knowledge from T)
- using a 2-of-2 multisig and an adaptor signature, one can obtain a cross-chain atomic swap protocol indistinguishable from standard spendings on each chain

Discreet Log Contracts (T. Dryja)

- allows to enforce contracts based on external events
  - oracle Olivia has public key: pair $(X = xG, R = rG)$
  - Olivia's signature on m is simply $s_m = r + H(R, m)x$
  - for any message m, anybody can compute $S_m = s_m G = R + H(R, m)X$
- to establish a contract, Alice and Bob send funds to a shared multisig address ($\sim$ payment channels in Lightning Network)
- for each possible outcome m_i of the external event, Alice and Bob have public keys $X_{a,m_i} = X_a + S_{m_i}$, resp. $X_{b,m_i} = X_b + S_{m_i}$ allowing to spend from the funding channel
- when the external event happens, Olivia signs the observed outcome m_{obs}
- Alice and Bob can compute resp. $X_{a,m_{\text{obs}}}$ and $X_{b,m_{\text{obs}}}$ and execute the contract

Discreet Log Contracts (T. Dryja)

- allows to enforce contracts based on external events
  - oracle Olivia has public key: pair $(X = xG, R = rG)$
  - Olivia's signature on m is simply $s_m = r + H(R, m)x$
  - for any message m, anybody can compute $S_m = s_m G = R + H(R, m)X$
- to establish a contract, Alice and Bob send funds to a shared multisig address ($\sim$ payment channels in Lightning Network)
- for each possible outcome m_i of the external event, Alice and Bob have public keys $X_{a,m_i} = X_a + S_{m_i}$, resp. $X_{b,m_i} = X_b + S_{m_i}$ allowing to spend from the funding channel
- when the external event happens, Olivia signs the observed outcome m_{obs}
- Alice and Bob can compute resp. $X_{a,m_{\text{obs}}}$ and $X_{b,m_{\text{obs}}}$ and execute the contract

Discreet Log Contracts (T. Dryja)

- allows to enforce contracts based on external events
  - oracle Olivia has public key: pair $(X = xG, R = rG)$
  - Olivia's signature on m is simply $s_m = r + H(R, m)x$
  - for any message m, anybody can compute $S_m = s_m G = R + H(R, m)X$
- to establish a contract, Alice and Bob send funds to a shared multisig address ($\sim$ payment channels in Lightning Network)
- for each possible outcome m_i of the external event, Alice and Bob have public keys $X_{a,m_i} = X_a + S_{m_i}$, resp. $X_{b,m_i} = X_b + S_{m_i}$ allowing to spend from the funding channel
- when the external event happens, Olivia signs the observed outcome m_{obs}
- Alice and Bob can compute resp. $X_{a,m_{\text{obs}}}$ and $X_{b,m_{\text{obs}}}$ and execute the contract

Discreet Log Contracts (T. Dryja)

- allows to enforce contracts based on external events
  - oracle Olivia has public key: pair $(X = xG, R = rG)$
  - Olivia's signature on m is simply $s_m = r + H(R, m)x$
  - for any message m, anybody can compute $S_m = s_m G = R + H(R, m)X$
- to establish a contract, Alice and Bob send funds to a shared multisig address ($\sim$ payment channels in Lightning Network)
- for each possible outcome m_i of the external event, Alice and Bob have public keys $X_{a,m_i} = X_a + S_{m_i}$, resp. $X_{b,m_i} = X_b + S_{m_i}$ allowing to spend from the funding channel
- when the external event happens, Olivia signs the observed outcome m_{obs}
- Alice and Bob can compute resp. $X_{a,m_{\text{obs}}}$ and $X_{b,m_{\text{obs}}}$ and execute the contract

Discreet Log Contracts (T. Dryja)

- allows to enforce contracts based on external events
  - oracle Olivia has public key: pair $(X = xG, R = rG)$
  - Olivia's signature on m is simply $s_m = r + H(R, m)x$
  - for any message m, anybody can compute $S_m = s_m G = R + H(R, m)X$
- to establish a contract, Alice and Bob send funds to a shared multisig address ($\sim$ payment channels in Lightning Network)
- for each possible outcome m_i of the external event, Alice and Bob have public keys $X_{a,m_i} = X_a + S_{m_i}$, resp. $X_{b,m_i} = X_b + S_{m_i}$ allowing to spend from the funding channel
- when the external event happens, Olivia signs the observed outcome m_{obs}
- Alice and Bob can compute resp. $X_{a,m_{\text{obs}}}$ and $X_{b,m_{\text{obs}}}$ and execute the contract

Discreet Log Contracts (T. Dryja)

- allows to enforce contracts based on external events
  - oracle Olivia has public key: pair $(X = xG, R = rG)$
  - Olivia's signature on m is simply $s_m = r + H(R, m)x$
  - for any message m, anybody can compute $S_m = s_m G = R + H(R, m)X$
- to establish a contract, Alice and Bob send funds to a shared multisig address ($\sim$ payment channels in Lightning Network)
- for each possible outcome m_i of the external event, Alice and Bob have public keys $X_{a,m_i} = X_a + S_{m_i}$, resp. $X_{b,m_i} = X_b + S_{m_i}$ allowing to spend from the funding channel
- when the external event happens, Olivia signs the observed outcome m_{obs}
- Alice and Bob can compute resp. $X_{a,m_{\text{obs}}}$ and $X_{b,m_{\text{obs}}}$ and execute the contract

Discreet Log Contracts (T. Dryja)

- allows to enforce contracts based on external events
  - oracle Olivia has public key: pair $(X = xG, R = rG)$
  - Olivia's signature on m is simply $s_m = r + H(R, m)x$
  - for any message m, anybody can compute $S_m = s_m G = R + H(R, m)X$
- to establish a contract, Alice and Bob send funds to a shared multisig address ($\sim$ payment channels in Lightning Network)
- for each possible outcome m_i of the external event, Alice and Bob have public keys $X_{a,m_i} = X_a + S_{m_i}$, resp. $X_{b,m_i} = X_b + S_{m_i}$ allowing to spend from the funding channel
- when the external event happens, Olivia signs the observed outcome m_{obs}
- Alice and Bob can compute resp. $X_{a,m_{\text{obs}}}$ and $X_{b,m_{\text{obs}}}$ and execute the contract

Discreet Log Contracts (T. Dryja)

- allows to enforce contracts based on external events
  - oracle Olivia has public key: pair $(X = xG, R = rG)$
  - Olivia's signature on m is simply $s_m = r + H(R, m)x$
  - for any message m, anybody can compute $S_m = s_m G = R + H(R, m)X$
- to establish a contract, Alice and Bob send funds to a shared multisig address ($\sim$ payment channels in Lightning Network)
- for each possible outcome m_i of the external event, Alice and Bob have public keys $X_{a,m_i} = X_a + S_{m_i}$, resp. $X_{b,m_i} = X_b + S_{m_i}$ allowing to spend from the funding channel
- when the external event happens, Olivia signs the observed outcome m_{obs}
- Alice and Bob can compute resp. $X_{a,m_{\text{obs}}}$ and $X_{b,m_{\text{obs}}}$ and execute the contract

Outline

Conclusion

- Schnorr signatures can:
  - reduce the size of transaction and speed up verification (lower cost, less network overhead, ...)
  - improve privacy (private multisigs, incentive to use CoinJoin, ...)
  - enable fun new applications (Sciptless scripts, Discreet Log Contracts, ...)
- can be activated as a soft fork (thanks to Segwit script versioning)
- careful: any change to cryptographic algorithms requires A LOT of review

Conclusion

- Schnorr signatures can:
  - reduce the size of transaction and speed up verification (lower cost, less network overhead, ...)
  - improve privacy (private multisigs, incentive to use CoinJoin, ...)
  - enable fun new applications (Sciptless scripts, Discreet Log Contracts, ...)
- can be activated as a soft fork (thanks to Segwit script versioning)
- careful: any change to cryptographic algorithms requires A LOT of review

Conclusion

- Schnorr signatures can:
  - reduce the size of transaction and speed up verification (lower cost, less network overhead, . . .)
  - improve privacy (private multisigs, incentive to use CoinJoin, . . .)
  - enable fun new applications (Sciptless scripts, Discreet Log Contracts, . . .)
- can be activated as a soft fork (thanks to Segwit script versioning)
- careful: any change to cryptographic algorithms requires A LOT of review

Conclusion

- Schnorr signatures can:
  - reduce the size of transaction and speed up verification (lower cost, less network overhead, . . .)
  - improve privacy (private multisigs, incentive to use CoinJoin, . . .)
  - enable fun new applications (Sciptless scripts, Discreet Log Contracts, . . .)
- can be activated as a soft fork (thanks to Segwit script versioning)
- careful: any change to cryptographic algorithms requires A LOT of review

Conclusion

- Schnorr signatures can:
  - reduce the size of transaction and speed up verification (lower cost, less network overhead, . . .)
  - improve privacy (private multisigs, incentive to use CoinJoin, . . .)
  - enable fun new applications (Sciptless scripts, Discreet Log Contracts, . . .)
- can be activated as a soft fork (thanks to Segwit script versioning)
- careful: any change to cryptographic algorithms requires A LOT of review

Conclusion

- Schnorr signatures can:
  - reduce the size of transaction and speed up verification (lower cost, less network overhead, . . .)
  - improve privacy (private multisigs, incentive to use CoinJoin, . . .)
  - enable fun new applications (Sciptless scripts, Discreet Log Contracts, . . .)
- can be activated as a soft fork (thanks to Segwit script versioning)
- careful: any change to cryptographic algorithms requires A LOT of review

The end...

Thanks for your attention!

Comments or questions?

References I

- Daniel J. Bernstein, Niels Duif, Tanja Lange, Peter Schwabe, and Bo-Yin Yang. High-Speed High-Security Signatures. In *Cryptographic Hardware and Embedded Systems - CHES 2011*, volume 6917 of *LNCS*, pages 124–142. Springer, 2011.

- Mihir Bellare and Gregory Neven. Multi-Signatures in the Plain Public-Key Model and a General Forking Lemma. In *ACM Conference on Computer and Communications Security - CCS 2006*, pages 390–399. ACM, 2006.

- Daniel R. L. Brown. Generic Groups, Collision Resistance, and ECDSA. *Des. Codes Cryptography*, 35(1):119–152, 2005.

- Amos Fiat and Adi Shamir. How to Prove Yourself: Practical Solutions to Identification and Signature Problems. In *Advances in Cryptology - CRYPTO '86*, volume 263 of *LNCS*, pages 186–194. Springer, 1986.

- K. Itakura and K. Nakamura. A public-key cryptosystem suitable for digital multisignatures. *NEC Research and Development*, 71:1–8, 1983.

References II

- Silvio Micali, Kazuo Ohta, and Leonid Reyzin. Accountable-Subgroup Multisignatures. In *ACM Conference on Computer and Communications Security - CCS 2001*, pages 245–254. ACM, 2001.
- Phong Q. Nguyen and Igor Shparlinski. The Insecurity of the Elliptic Curve Digital Signature Algorithm with Partially Known Nonces. *Des. Codes Cryptography*, 30(2):201–217, 2003.
- David Pointcheval and Jacques Stern. Security Proofs for Signature Schemes. In *Advances in Cryptology - EUROCRYPT '96*, volume 1070 of *LNCS*, pages 387–398. Springer, 1996.
- Thomas Ristenpart and Scott Yilek. The Power of Proofs-of-Possession: Securing Multiparty Signatures against Rogue-Key Attacks. In *Advances in Cryptology - EUROCRYPT 2007*, volume 4515 of *LNCS*, pages 228–245. Springer, 2007.
- Claus-Peter Schnorr. Efficient Identification and Signatures for Smart Cards. In *Advances in Cryptology - CRYPTO '89*, volume 435 of *LNCS*, pages 239–252. Springer, 1989.

References III

Claus-Peter Schnorr. Efficient Signature Generation by Smart Cards. *J. Cryptology*, 4(3):161–174, 1991.

Certicom Research. *SEC 2: Recommended Elliptic Curve Domain Parameters, v2.0*, 2010. Available at `http://www.secg.org/sec2-v2.pdf`.