# More Schnorr Tricks for Bitcoin

Yannick Seurin

Agence nationale de la sécurité des systèmes d'information

November 22, 2018 — “BlockSem” Seminar

# Motivation: improving efficiency and privacy

- Bitcoin script allows to specify (pretty sophisticated) conditions for spending a transaction output
- allows very nice applications, but:
    - scripts are recorded forever in the blockchain
      → goes against space efficiency and privacy
    - scripts must be validated by all nodes
      → goes against computational efficiency
    - coins have a distinguished "history"
      → goes against fungibility (all coins should be "equivalent")
- we will see how Schnorr signatures can help make things better

# Motivation: improving efficiency and privacy

- Bitcoin script allows to specify (pretty sophisticated) conditions for spending a transaction output
- allows very nice applications, but:
  - scripts are recorded forever in the blockchain
    → goes against space efficiency and privacy
  - scripts must be validated by all nodes
    → goes against computational efficiency
  - coins have a distinguished "history"
    → goes against fungibility (all coins should be "equivalent")
- we will see how Schnorr signatures can help make things better

# Motivation: improving efficiency and privacy

- Bitcoin script allows to specify (pretty sophisticated) conditions for spending a transaction output
- allows very nice applications, but:
  - scripts are recorded forever in the blockchain
    → goes against space efficiency and privacy
  - scripts must be validated by all nodes
    → goes against computational efficiency
  - coins have a distinguished "history"
    → goes against fungibility (all coins should be "equivalent")
- we will see how Schnorr signatures can help make things better

# Motivation: improving efficiency and privacy

- Bitcoin script allows to specify (pretty sophisticated) conditions for spending a transaction output
- allows very nice applications, but:
  - scripts are recorded forever in the blockchain
    → goes against space efficiency and privacy
  - scripts must be validated by all nodes
    → goes against computational efficiency
  - coins have a distinguished "history"
    → goes against fungibility (all coins should be "equivalent")
- we will see how Schnorr signatures can help make things better

# Motivation: improving efficiency and privacy

- Bitcoin script allows to specify (pretty sophisticated) conditions for spending a transaction output
- allows very nice applications, but:
  - scripts are recorded forever in the blockchain
    → goes against space efficiency and privacy
  - scripts must be validated by all nodes
    → goes against computational efficiency
  - coins have a distinguished "history"
    → goes against fungibility (all coins should be "equivalent")
- we will see how Schnorr signatures can help make things better

# Motivation: improving efficiency and privacy

- Bitcoin script allows to specify (pretty sophisticated) conditions for spending a transaction output
- allows very nice applications, but:
  - scripts are recorded forever in the blockchain
    → goes against space efficiency and privacy
  - scripts must be validated by all nodes
    → goes against computational efficiency
  - coins have a distinguished "history"
    → goes against fungibility (all coins should be "equivalent")
- we will see how Schnorr signatures can help make things better

# Outline

# Outline

Bitcoin Script

Refresher: Schnorr Signatures and MuSig

Taproot

Scriptless Scripts

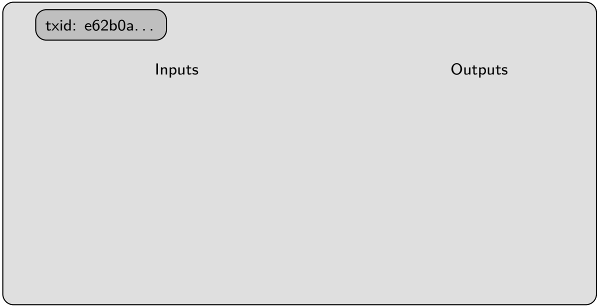
Discreet Log Contracts

Conclusion

# Bitcoin transactions: UTXO model

A Bitcoin transaction spends inputs and creates outputs:

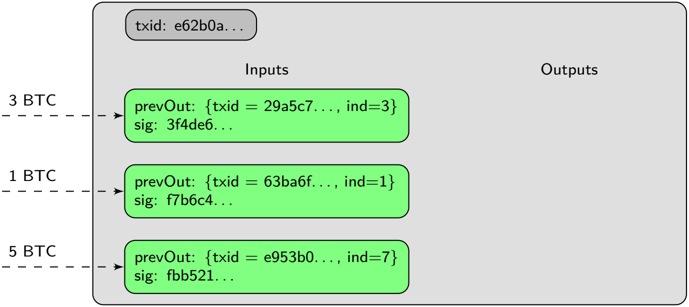
- an input consists of a reference to an output of a previous transaction and a signature authorizing spending of this output
- an output consists of an amount and a public key



# Bitcoin transactions: UTXO model

A Bitcoin transaction spends inputs and creates outputs:

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



txid: e62b0a...

Inputs

Outputs

prevOut: {txid = 29a5c7..., ind=3}
sig: 3f4de6...

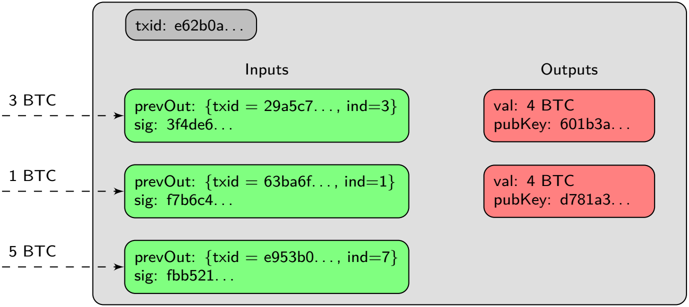
prevOut: {txid = 63ba6f..., ind=1}
sig: f7b6c4...

prevOut: {txid = e953b0..., ind=7}
sig: fbb521...

# Bitcoin transactions: UTXO model

A Bitcoin transaction spends inputs and creates outputs:

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



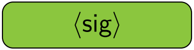
# Programmable money: Bitcoin script

- output public keys and input signatures are actually scripts
- output: scriptPubKey, input: scriptSig
- concatenated script scriptSig $\|$ scriptPubKey must execute correctly
- stack-based language designed for Bitcoin, inspired by Forth
- 256 instructions (15 disabled, 75 reserved):
  - basic arithmetic, logic (if/then), data handling
  - cryptographic operations (hash and signature verification)
- no loops, Turing-incomplete
- limits on time/memory required for execution (no halting problem)

# Example: *Pay-to-Public-Key-Hash* (P2PKH)

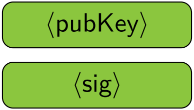
$$\underbrace{\langle\text{sig}\rangle \quad \langle\text{pubKey}\rangle}_{\text{scriptSig}} \qquad \underbrace{\text{OP\_DUP} \quad \text{OP\_HASH160} \quad \langle\text{pubKeyHash}\rangle \quad \text{OP\_EQUALVERIFY} \quad \text{OP\_CHECKSIG}}_{\text{scriptPubKey}}$$

# Example: *Pay-to-Public-Key-Hash* (P2PKH)



$$\underbrace{\color{red}{\langle\text{sig}\rangle}\quad\langle\text{pubKey}\rangle}_{\text{scriptSig}}\qquad\underbrace{\text{OP\_DUP}\quad\text{OP\_HASH160}\quad\langle\text{pubKeyHash}\rangle\quad\text{OP\_EQUALVERIFY}\quad\text{OP\_CHECKSIG}}_{\text{scriptPubKey}}$$

# Example: *Pay-to-Public-Key-Hash* (P2PKH)

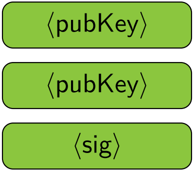


⟨pubKey⟩

⟨sig⟩

$$\underbrace{\langle \text{sig} \rangle \quad \langle \text{pubKey} \rangle}_{\text{scriptSig}} \qquad \underbrace{\text{OP\_DUP} \quad \text{OP\_HASH160} \quad \langle \text{pubKeyHash} \rangle \quad \text{OP\_EQUALVERIFY} \quad \text{OP\_CHECKSIG}}_{\text{scriptPubKey}}$$

# Example: *Pay-to-Public-Key-Hash* (P2PKH)



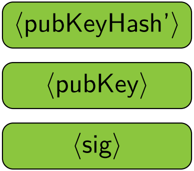
$\langle$pubKey$\rangle$

$\langle$pubKey$\rangle$

$\langle$sig$\rangle$

$$\underbrace{\langle\text{sig}\rangle \quad \langle\text{pubKey}\rangle}_{\text{scriptSig}} \qquad \underbrace{\text{OP\_DUP} \quad \text{OP\_HASH160} \quad \langle\text{pubKeyHash}\rangle \quad \text{OP\_EQUALVERIFY} \quad \text{OP\_CHECKSIG}}_{\text{scriptPubKey}}$$

# Example: *Pay-to-Public-Key-Hash* (P2PKH)



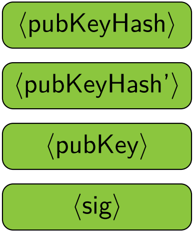
⟨pubKeyHash'⟩

⟨pubKey⟩

⟨sig⟩

$\underbrace{\langle\text{sig}\rangle \quad \langle\text{pubKey}\rangle}_{\text{scriptSig}} \quad \underbrace{\text{OP\_DUP} \quad \text{OP\_HASH160} \quad \langle\text{pubKeyHash}\rangle \quad \text{OP\_EQUALVERIFY} \quad \text{OP\_CHECKSIG}}_{\text{scriptPubKey}}$

# Example: *Pay-to-Public-Key-Hash* (P2PKH)



⟨pubKeyHash⟩

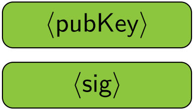
⟨pubKeyHash’⟩

⟨pubKey⟩

⟨sig⟩

$$\underbrace{\langle\text{sig}\rangle \quad \langle\text{pubKey}\rangle}_{\text{scriptSig}} \qquad \underbrace{\text{OP\_DUP} \quad \text{OP\_HASH160} \quad \langle\text{pubKeyHash}\rangle \quad \text{OP\_EQUALVERIFY} \quad \text{OP\_CHECKSIG}}_{\text{scriptPubKey}}$$

# Example: *Pay-to-Public-Key-Hash* (P2PKH)



$$\underbrace{\langle\text{sig}\rangle\quad\langle\text{pubKey}\rangle}_{\text{scriptSig}}\qquad\underbrace{\text{OP\_DUP}\quad\text{OP\_HASH160}\quad\langle\text{pubKeyHash}\rangle\quad\text{OP\_EQUALVERIFY}\quad\text{OP\_CHECKSIG}}_{\text{scriptPubKey}}$$

# Example: *Pay-to-Public-Key-Hash* (P2PKH)

✓

$$\underbrace{\langle \text{sig} \rangle \quad \langle \text{pubKey} \rangle}_{\text{scriptSig}} \qquad \underbrace{\text{OP\_DUP} \quad \text{OP\_HASH160} \quad \langle \text{pubKeyHash} \rangle \quad \text{OP\_EQUALVERIFY} \quad \text{OP\_CHECKSIG}}_{\text{scriptPubKey}}$$

# Example: *Pay-to-Public-Key-Hash* (P2PKH)

$$\underbrace{\langle\text{sig}\rangle\quad\langle\text{pubKey}\rangle}_{\text{scriptSig}}\qquad\underbrace{\text{OP\_DUP}\quad\text{OP\_HASH160}\quad\langle\text{pubKeyHash}\rangle\quad\text{OP\_EQUALVERIFY}\quad\text{OP\_CHECKSIG}}_{\text{scriptPubKey}}$$

- Bitcoin “address” = RIPEMD-160(SHA-256(public key)) encoded in Base58Check format (starts with a ’1’)

# Other useful instructions

- *m*-of-*n* MULTISIG:
  - scriptPubKey contains *n* public keys
  - scriptSig must provide $m \leq n$ valid signatures for *m* out of *n* of these public keys
  - many applications (multi-authentication wallet, escrow, etc.)
- OP_RETURN:
  - makes output unspendable
  - used to put arbitrary data in the blockchain
- Lock-time:
  - output unspendable until some time in the future
  - absolute (CLTV) or relative (CSV)
  - application: payment channels, Lightning Network

# Other useful instructions

- $m$-of-$n$ MULTISIG:
  - scriptPubKey contains $n$ public keys
  - scriptSig must provide $m \leq n$ valid signatures for $m$ out of $n$ of these public keys
  - many applications (multi-authentication wallet, escrow, etc.)
- OP_RETURN:
  - makes output unspendable
  - used to put arbitrary data in the blockchain
- Lock-time:
  - output unspendable until some time in the future
  - absolute (CLTV) or relative (CSV)
  - application: payment channels, Lightning Network

# Other useful instructions

- *m*-of-*n* MULTISIG:
    - scriptPubKey contains $n$ public keys
    - scriptSig must provide $m \leq n$ valid signatures for $m$ out of $n$ of these public keys
    - many applications (multi-authentication wallet, escrow, etc.)
- OP_RETURN:
    - makes output unspendable
    - used to put arbitrary data in the blockchain
- Lock-time:
    - output unspendable until some time in the future
    - absolute (CLTV) or relative (CSV)
    - application: payment channels, Lightning Network

# Other useful instructions

- *m*-of-*n* MULTISIG:
  - scriptPubKey contains $n$ public keys
  - scriptSig must provide $m \leq n$ valid signatures for $m$ out of $n$ of these public keys
  - many applications (multi-authentication wallet, escrow, etc.)
- OP_RETURN:
  - makes output unspendable
  - used to put arbitrary data in the blockchain
- Lock-time:
  - output unspendable until some time in the future
  - absolute (CLTV) or relative (CSV)
  - application: payment channels, Lightning Network

# Other useful instructions

- $m$-of-$n$ MULTISIG:
  - scriptPubKey contains $n$ public keys
  - scriptSig must provide $m \leq n$ valid signatures for $m$ out of $n$ of these public keys
  - many applications (multi-authentication wallet, escrow, etc.)
- OP_RETURN:
  - makes output unspendable
  - used to put arbitrary data in the blockchain
- Lock-time:
  - output unspendable until some time in the future
  - absolute (CLTV) or relative (CSV)
  - application: payment channels, Lightning Network

# Other useful instructions

- *m*-of-*n* MULTISIG:
  - scriptPubKey contains *n* public keys
  - scriptSig must provide $m \leq n$ valid signatures for *m* out of *n* of these public keys
  - many applications (multi-authentication wallet, escrow, etc.)
- OP_RETURN:
  - makes output unspendable
  - used to put arbitrary data in the blockchain
- Lock-time:
  - output unspendable until some time in the future
  - absolute (CLTV) or relative (CSV)
  - application: payment channels, Lightning Network

# Other useful instructions

- *m*-of-*n* MULTISIG:
  - scriptPubKey contains *n* public keys
  - scriptSig must provide $m \leq n$ valid signatures for *m* out of *n* of these public keys
  - many applications (multi-authentication wallet, escrow, etc.)
- OP_RETURN:
  - makes output unspendable
  - used to put arbitrary data in the blockchain
- Lock-time:
  - output unspendable until some time in the future
  - absolute (CLTV) or relative (CSV)
  - application: payment channels, Lightning Network

# Other useful instructions

- $m$-of-$n$ MULTISIG:
  - scriptPubKey contains $n$ public keys
  - scriptSig must provide $m \leq n$ valid signatures for $m$ out of $n$ of these public keys
  - many applications (multi-authentication wallet, escrow, etc.)
- OP_RETURN:
  - makes output unspendable
  - used to put arbitrary data in the blockchain
- Lock-time:
  - output unspendable until some time in the future
  - absolute (CLTV) or relative (CSV)
  - application: payment channels, Lightning Network

# Other useful instructions

- $m$-of-$n$ MULTISIG:
  - scriptPubKey contains $n$ public keys
  - scriptSig must provide $m \leq n$ valid signatures for $m$ out of $n$ of these public keys
  - many applications (multi-authentication wallet, escrow, etc.)
- OP_RETURN:
  - makes output unspendable
  - used to put arbitrary data in the blockchain
- Lock-time:
  - output unspendable until some time in the future
  - absolute (CLTV) or relative (CSV)
  - application: payment channels, Lightning Network

# Other useful instructions

- $m$-of-$n$ MULTISIG:
  - scriptPubKey contains $n$ public keys
  - scriptSig must provide $m \leq n$ valid signatures for $m$ out of $n$ of these public keys
  - many applications (multi-authentication wallet, escrow, etc.)
- OP_RETURN:
  - makes output unspendable
  - used to put arbitrary data in the blockchain
- Lock-time:
  - output unspendable until some time in the future
  - absolute (CLTV) or relative (CSV)
  - application: payment channels, Lightning Network

# Other useful instructions

- $m$-of-$n$ MULTISIG:
    - scriptPubKey contains $n$ public keys
    - scriptSig must provide $m \leq n$ valid signatures for $m$ out of $n$ of these public keys
    - many applications (multi-authentication wallet, escrow, etc.)
- OP_RETURN:
    - makes output unspendable
    - used to put arbitrary data in the blockchain
- Lock-time:
    - output unspendable until some time in the future
    - absolute (CLTV) or relative (CSV)
    - application: payment channels, Lightning Network

# Hash Time-Lock Contract (HTLC)

- Hash Time-Locked Contracts HTLC$(h, X_1, \tau, X_2)$:

OP_IF OP_SHA256 $\langle h \rangle$ OP_EQUALVERIFY $\langle X_1 \rangle$ OP_CHECKSIG
OP_ELSE $\langle \tau \rangle$ OP_CLTV OP_DROP $\langle X_2 \rangle$ OP_CHECKSIG OP_ENDIF

- in words, such a output can be spent either
  - with $y$ such that SHA256$(y) = h$ and a signature under $X_1$
  - OR after time $\tau$ with a signature under $X_2$
- used in the Lightning Network for payment channels and routing

# Hash Time-Lock Contract (HTLC)

- Hash Time-Locked Contracts HTLC$(h, X_1, \tau, X_2)$:

OP_IF OP_SHA256 $\langle h \rangle$ OP_EQUALVERIFY $\langle X_1 \rangle$ OP_CHECKSIG
OP_ELSE $\langle \tau \rangle$ OP_CLTV OP_DROP $\langle X_2 \rangle$ OP_CHECKSIG OP_ENDIF

- in words, such a output can be spent either
  - with $y$ such that SHA256$(y) = h$ and a signature under $X_1$
  - OR after time $\tau$ with a signature under $X_2$
- used in the Lightning Network for payment channels and routing

# Hash Time-Lock Contract (HTLC)

- Hash Time-Locked Contracts HTLC$(h, X_1, \tau, X_2)$:

```
OP_IF OP_SHA256 ⟨h⟩ OP_EQUALVERIFY ⟨X_1⟩ OP_CHECKSIG
OP_ELSE ⟨τ⟩ OP_CLTV OP_DROP ⟨X_2⟩ OP_CHECKSIG OP_ENDIF
```

- in words, such a output can be spent either
  - with $y$ such that SHA256$(y) = h$ and a signature under $X_1$
  - OR after time $\tau$ with a signature under $X_2$
- used in the Lightning Network for payment channels and routing

# Atomic (cross-chain) swaps [Nol13]

- allows trading without a trusted party
- suppose Alice wants to trade 1 bitcoin for 100 litecoins with Bob
- Alice (public key $X_A$) and Bob (public key $X_B$) proceed as follows:
  - Bob chooses random $y$ and sends $h = \text{SHA256}(y)$ to Alice
  - Bob sends 100 litecoins to $\text{HTLC}(X_A, h, X_B, \tau_B)$
  - Alice sends 1 bitcoin to $\text{HTLC}(X_B, h, X_A, \tau_A)$
  - Bob claims Alice's bitcoin, revealing $y$
  - Alice can claim Bob's 100 litecoins using $y$
- if anything goes wrong, parties can get funds back after $\tau_A/\tau_B$
- $\tau_B$ must be significantly later than $\tau_A$ (otherwise Bob could claim both HTLC outputs between $\tau_B$ and $\tau_A$)
- problem: not private at all, the payments can be linked with $y$

# Atomic (cross-chain) swaps [Nol13]

- allows trading without a trusted party
- suppose Alice wants to trade 1 bitcoin for 100 litecoins with Bob
- Alice (public key $X_A$) and Bob (public key $X_B$) proceed as follows:
  - Bob chooses random $y$ and sends $h = \text{SHA256}(y)$ to Alice
  - Bob sends 100 litecoins to $\text{HTLC}(X_A, h, X_B, \tau_B)$
  - Alice sends 1 bitcoin to $\text{HTLC}(X_B, h, X_A, \tau_A)$
  - Bob claims Alice's bitcoin, revealing $y$
  - Alice can claim Bob's 100 litecoins using $y$
- if anything goes wrong, parties can get funds back after $\tau_A/\tau_B$
- $\tau_B$ must be significantly later than $\tau_A$ (otherwise Bob could claim both HTLC outputs between $\tau_B$ and $\tau_A$)
- problem: not private at all, the payments can be linked with $y$

# Atomic (cross-chain) swaps [Nol13]

- allows trading without a trusted party
- suppose Alice wants to trade 1 bitcoin for 100 litecoins with Bob
- Alice (public key $X_A$) and Bob (public key $X_B$) proceed as follows:
  - Bob chooses random $y$ and sends $h = \mathrm{SHA256}(y)$ to Alice
  - Bob sends 100 litecoins to $\mathrm{HTLC}(X_A, h, X_B, \tau_B)$
  - Alice sends 1 bitcoin to $\mathrm{HTLC}(X_B, h, X_A, \tau_A)$
  - Bob claims Alice's bitcoin, revealing $y$
  - Alice can claim Bob's 100 litecoins using $y$
- if anything goes wrong, parties can get funds back after $\tau_A/\tau_B$
- $\tau_B$ must be significantly later than $\tau_A$ (otherwise Bob could claim both HTLC outputs between $\tau_B$ and $\tau_A$)
- problem: not private at all, the payments can be linked with $y$

# Atomic (cross-chain) swaps [Nol13]

- allows trading without a trusted party
- suppose Alice wants to trade 1 bitcoin for 100 litecoins with Bob
- Alice (public key $X_A$) and Bob (public key $X_B$) proceed as follows:
  - Bob chooses random $y$ and sends $h = \mathrm{SHA256}(y)$ to Alice
  - Bob sends 100 litecoins to $\mathrm{HTLC}(X_A, h, X_B, \tau_B)$
  - Alice sends 1 bitcoin to $\mathrm{HTLC}(X_B, h, X_A, \tau_A)$
  - Bob claims Alice's bitcoin, revealing $y$
  - Alice can claim Bob's 100 litecoins using $y$
- if anything goes wrong, parties can get funds back after $\tau_A/\tau_B$
- $\tau_B$ must be significantly later than $\tau_A$ (otherwise Bob could claim both HTLC outputs between $\tau_B$ and $\tau_A$)
- problem: not private at all, the payments can be linked with $y$

# Atomic (cross-chain) swaps [Nol13]

- allows trading without a trusted party
- suppose Alice wants to trade 1 bitcoin for 100 litecoins with Bob
- Alice (public key $X_A$) and Bob (public key $X_B$) proceed as follows:
  - Bob chooses random $y$ and sends $h = \text{SHA256}(y)$ to Alice
  - Bob sends 100 litecoins to $\text{HTLC}(X_A, h, X_B, \tau_B)$
  - Alice sends 1 bitcoin to $\text{HTLC}(X_B, h, X_A, \tau_A)$
  - Bob claims Alice's bitcoin, revealing $y$
  - Alice can claim Bob's 100 litecoins using $y$
- if anything goes wrong, parties can get funds back after $\tau_A/\tau_B$
- $\tau_B$ must be significantly later than $\tau_A$ (otherwise Bob could claim both HTLC outputs between $\tau_B$ and $\tau_A$)
- problem: not private at all, the payments can be linked with $y$

# Atomic (cross-chain) swaps [Nol13]

- allows trading without a trusted party
- suppose Alice wants to trade 1 bitcoin for 100 litecoins with Bob
- Alice (public key $X_A$) and Bob (public key $X_B$) proceed as follows:
    - Bob chooses random $y$ and sends $h = \mathsf{SHA256}(y)$ to Alice
    - Bob sends 100 litecoins to $\mathsf{HTLC}(X_A, h, X_B, \tau_B)$
    - Alice sends 1 bitcoin to $\mathsf{HTLC}(X_B, h, X_A, \tau_A)$
    - Bob claims Alice's bitcoin, revealing $y$
    - Alice can claim Bob's 100 litecoins using $y$
- if anything goes wrong, parties can get funds back after $\tau_A/\tau_B$
- $\tau_B$ must be significantly later than $\tau_A$ (otherwise Bob could claim both HTLC outputs between $\tau_B$ and $\tau_A$)
- problem: not private at all, the payments can be linked with $y$

# Atomic (cross-chain) swaps [Nol13]

- allows trading without a trusted party
- suppose Alice wants to trade 1 bitcoin for 100 litecoins with Bob
- Alice (public key $X_A$) and Bob (public key $X_B$) proceed as follows:
  - Bob chooses random $y$ and sends $h = \mathrm{SHA256}(y)$ to Alice
  - Bob sends 100 litecoins to $\mathrm{HTLC}(X_A, h, X_B, \tau_B)$
  - Alice sends 1 bitcoin to $\mathrm{HTLC}(X_B, h, X_A, \tau_A)$
  - Bob claims Alice's bitcoin, revealing $y$
  - Alice can claim Bob's 100 litecoins using $y$
- if anything goes wrong, parties can get funds back after $\tau_A/\tau_B$
- $\tau_B$ must be significantly later than $\tau_A$ (otherwise Bob could claim both HTLC outputs between $\tau_B$ and $\tau_A$)
- problem: not private at all, the payments can be linked with $y$

# Atomic (cross-chain) swaps [Nol13]

- allows trading without a trusted party
- suppose Alice wants to trade 1 bitcoin for 100 litecoins with Bob
- Alice (public key $X_A$) and Bob (public key $X_B$) proceed as follows:
  - Bob chooses random $y$ and sends $h = \text{SHA256}(y)$ to Alice
  - Bob sends 100 litecoins to $\text{HTLC}(X_A, h, X_B, \tau_B)$
  - Alice sends 1 bitcoin to $\text{HTLC}(X_B, h, X_A, \tau_A)$
  - Bob claims Alice's bitcoin, revealing $y$
  - Alice can claim Bob's 100 litecoins using $y$
- if anything goes wrong, parties can get funds back after $\tau_A/\tau_B$
- $\tau_B$ must be significantly later than $\tau_A$ (otherwise Bob could claim both HTLC outputs between $\tau_B$ and $\tau_A$)
- problem: not private at all, the payments can be linked with $y$

# Atomic (cross-chain) swaps [Nol13]

- allows trading without a trusted party
- suppose Alice wants to trade 1 bitcoin for 100 litecoins with Bob
- Alice (public key $X_A$) and Bob (public key $X_B$) proceed as follows:
    - Bob chooses random $y$ and sends $h = \mathsf{SHA256}(y)$ to Alice
    - Bob sends 100 litecoins to $\mathsf{HTLC}(X_A, h, X_B, \tau_B)$
    - Alice sends 1 bitcoin to $\mathsf{HTLC}(X_B, h, X_A, \tau_A)$
    - Bob claims Alice's bitcoin, revealing $y$
    - Alice can claim Bob's 100 litecoins using $y$
- if anything goes wrong, parties can get funds back after $\tau_A/\tau_B$
- $\tau_B$ must be significantly later than $\tau_A$ (otherwise Bob could claim both HTLC outputs between $\tau_B$ and $\tau_A$)
- problem: not private at all, the payments can be linked with $y$

# Atomic (cross-chain) swaps [Nol13]

- allows trading without a trusted party
- suppose Alice wants to trade 1 bitcoin for 100 litecoins with Bob
- Alice (public key $X_A$) and Bob (public key $X_B$) proceed as follows:
  - Bob chooses random $y$ and sends $h = \mathsf{SHA256}(y)$ to Alice
  - Bob sends 100 litecoins to $\mathsf{HTLC}(X_A, h, X_B, \tau_B)$
  - Alice sends 1 bitcoin to $\mathsf{HTLC}(X_B, h, X_A, \tau_A)$
  - Bob claims Alice's bitcoin, revealing $y$
  - Alice can claim Bob's 100 litecoins using $y$
- if anything goes wrong, parties can get funds back after $\tau_A/\tau_B$
- $\tau_B$ must be significantly later than $\tau_A$ (otherwise Bob could claim both HTLC outputs between $\tau_B$ and $\tau_A$)
- problem: not private at all, the payments can be linked with $y$

# Atomic (cross-chain) swaps [Nol13]

- allows trading without a trusted party
- suppose Alice wants to trade 1 bitcoin for 100 litecoins with Bob
- Alice (public key $X_A$) and Bob (public key $X_B$) proceed as follows:
  - Bob chooses random $y$ and sends $h = \mathsf{SHA256}(y)$ to Alice
  - Bob sends 100 litecoins to $\mathsf{HTLC}(X_A, h, X_B, \tau_B)$
  - Alice sends 1 bitcoin to $\mathsf{HTLC}(X_B, h, X_A, \tau_A)$
  - Bob claims Alice's bitcoin, revealing $y$
  - Alice can claim Bob's 100 litecoins using $y$
- if anything goes wrong, parties can get funds back after $\tau_A/\tau_B$
- $\tau_B$ must be significantly later than $\tau_A$ (otherwise Bob could claim both HTLC outputs between $\tau_B$ and $\tau_A$)
- problem: not private at all, the payments can be linked with $y$

# Automated bounties

- What does the following scriptPubKey?

```
OP_2DUP  OP_EQUAL  OP_NOT  OP_VERIFY  OP_SHA1  OP_SWAP  OP_SHA1  OP_EQUAL
```

# Automated bounties

- What does the following scriptPubKey?

```
OP_2DUP OP_EQUAL OP_NOT OP_VERIFY OP_SHA1 OP_SWAP OP_SHA1 OP_EQUAL
```

- scriptSig $= \langle m_1 \rangle \ \langle m_2 \rangle$ returns True if

$$m_1 \neq m_2 \text{ and } \mathsf{SHA1}(m_1) = \mathsf{SHA1}(m_2)$$

# Automated bounties

- What does the following scriptPubKey?

```
OP_2DUP   OP_EQUAL   OP_NOT   OP_VERIFY   OP_SHA1   OP_SWAP   OP_SHA1   OP_EQUAL
```

- scriptSig $= \langle m_1 \rangle \ \langle m_2 \rangle$ returns True if

$$m_1 \neq m_2 \text{ and } \mathrm{SHA1}(m_1) = \mathrm{SHA1}(m_2)$$

- bounty created in Sept. 2013 by P. Todd

(`https://bitcointalk.org/index.php?topic=293382.0`)

8d31992805518fd62daa3bdd2a5c4fd2cd3054c9b3dca1d78055e9528cff6adc (Fee: 0.01153717 BTC - 41.06 sat/WU - 164.25 sat/B - Size: 7024 bytes) 2017-02-23 13:03:33

37k7toV1Nv4DfmQbmZ8KuZDQCYK9x5KpzP (0.01 BTC - Output)
37k7toV1Nv4DfmQbmZ8KuZDQCYK9x5KpzP (0.1 BTC - Output)
37k7toV1Nv4DfmQbmZ8KuZDQCYK9x5KpzP (0.5 BTC - Output)
37k7toV1Nv4DfmQbmZ8KuZDQCYK9x5KpzP (0.01 BTC - Output)
37k7toV1Nv4DfmQbmZ8KuZDQCYK9x5KpzP (1 BTC - Output)
37k7toV1Nv4DfmQbmZ8KuZDQCYK9x5KpzP (0.5 BTC - Output)
37k7toV1Nv4DfmQbmZ8KuZDQCYK9x5KpzP (0.05153717 BTC - Output)
37k7toV1Nv4DfmQbmZ8KuZDQCYK9x5KpzP (0.3 BTC - Output)
37k7toV1Nv4DfmQbmZ8KuZDQCYK9x5KpzP (0.01 BTC - Output)
37k7toV1Nv4DfmQbmZ8KuZDQCYK9x5KpzP (0.01 BTC - Output)

➡ 1EohDhHJT9byKsYhxp5zX6PNkuGhxoEu9r - (Spent) 2.48 BTC

-2.49153717 BTC

# Outline

# Signature scheme: definition

A signature scheme consists of three algorithms:

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

Correctness property:

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

# Mathematical background

## 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

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

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

## 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 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 $sG \overset{?}{=} R + cX$
- alternative:
  - signature $\sigma = (c, s)$
  - verification: compute $R = sG - cX$ and check $H(X, R, m) \overset{?}{=} c$

# 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 $sG \overset{?}{=} R + cX$
- alternative:
    - signature $\sigma = (c, s)$
    - verification: compute $R = sG - cX$ and check $H(X, R, m) \overset{?}{=} c$

# 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 $sG \overset{?}{=} R + cX$
- alternative:
  - signature $\sigma = (c, s)$
  - verification: compute $R = sG - cX$ and check $H(X, R, m) \overset{?}{=} c$

# 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 $sG \overset{?}{=} R + cX$
- alternative:
  - signature $\sigma = (c, s)$
  - verification: compute $R = sG - cX$ and check $H(X, R, m) \overset{?}{=} c$

# MuSig: Multi-signatures supporting key aggregation

- assume $n$ signers with public keys $\{X_1 = x_1 G, \ldots, X_n = x_n G\}$ want to sign the same message $m$
- they compute an aggregate key

$$\widetilde{X} := \sum_{i=1}^{n} \mu_i X_i \quad \text{with} \quad \mu_i = H(\{X_1, \ldots, X_n\}, X_i)$$

- signature protocol:
  - signers draw nonces $R_i = r_i G$ and send commitments $h_i = H'(R_i)$
  - signers exchange nonces $R_i$
  - signers compute $R = \sum_{i=1}^{n} R_i$ and $c = H(\widetilde{X}, R, m)$
  - signers compute and exchange partial signatures $s_i = r_i + c\mu_i x_i$
  - signers compute $s = \sum_{i=1}^{n} s_i \bmod p$
  - the multi-signature is $\sigma = (R, s)$

# MuSig: Multi-signatures supporting key aggregation

- assume $n$ signers with public keys $\{X_1 = x_1 G, \ldots, X_n = x_n G\}$ want to sign the same message $m$
- they compute an aggregate key

$$\widetilde{X} := \sum_{i=1}^{n} \mu_i X_i \quad \text{with} \quad \mu_i = H(\{X_1, \ldots, X_n\}, X_i)$$

- signature protocol:
  - signers draw nonces $R_i = r_i G$ and send commitments $h_i = H'(R_i)$
  - signers exchange nonces $R_i$
  - signers compute $R = \sum_{i=1}^{n} R_i$ and $c = H(\widetilde{X}, R, m)$
  - signers compute and exchange partial signatures $s_i = r_i + c\mu_i x_i$
  - signers compute $s = \sum_{i=1}^{n} s_i \bmod p$
  - the multi-signature is $\sigma = (R, s)$

# MuSig: Multi-signatures supporting key aggregation

- assume $n$ signers with public keys $\{X_1 = x_1 G, \ldots, X_n = x_n G\}$ want to sign the same message $m$
- they compute an aggregate key

$$\widetilde{X} := \sum_{i=1}^{n} \mu_i X_i \quad \text{with} \quad \mu_i = H(\{X_1, \ldots, X_n\}, X_i)$$

- signature protocol:
  - signers draw nonces $R_i = r_i G$ and send commitments $h_i = H'(R_i)$
  - signers exchange nonces $R_i$
  - signers compute $R = \sum_{i=1}^{n} R_i$ and $c = H(\widetilde{X}, R, m)$
  - signers compute and exchange partial signatures $s_i = r_i + c\mu_i x_i$
  - signers compute $s = \sum_{i=1}^{n} s_i \bmod p$
  - the multi-signature is $\sigma = (R, s)$

# MuSig: Multi-signatures supporting key aggregation

- assume $n$ signers with public keys $\{X_1 = x_1 G, \dots, X_n = x_n G\}$ want to sign the same message $m$
- they compute an aggregate key

$$\widetilde{X} := \sum_{i=1}^{n} \mu_i X_i \quad \text{with} \quad \mu_i = H(\{X_1, \dots, X_n\}, X_i)$$

- signature protocol:
    - signers draw nonces $R_i = r_i G$ and send commitments $h_i = H'(R_i)$
    - signers exchange nonces $R_i$
    - signers compute $R = \sum_{i=1}^{n} R_i$ and $c = H(\widetilde{X}, R, m)$
    - signers compute and exchange partial signatures $s_i = r_i + c\mu_i x_i$
    - signers compute $s = \sum_{i=1}^{n} s_i \bmod p$
    - the multi-signature is $\sigma = (R, s)$

# MuSig: Multi-signatures supporting key aggregation

- assume $n$ signers with public keys $\{X_1 = x_1 G, \dots, X_n = x_n G\}$ want to sign the same message $m$
- they compute an aggregate key

$$\widetilde{X} := \sum_{i=1}^{n} \mu_i X_i \quad \text{with} \quad \mu_i = H(\{X_1, \dots, X_n\}, X_i)$$

- signature protocol:
  - signers draw nonces $R_i = r_i G$ and send commitments $h_i = H'(R_i)$
  - signers exchange nonces $R_i$
  - signers compute $R = \sum_{i=1}^{n} R_i$ and $c = H(\widetilde{X}, R, m)$
  - signers compute and exchange partial signatures $s_i = r_i + c\mu_i x_i$
  - signers compute $s = \sum_{i=1}^{n} s_i \bmod p$
  - the multi-signature is $\sigma = (R, s)$

# MuSig: Multi-signatures supporting key aggregation

- assume $n$ signers with public keys $\{X_1 = x_1 G, \ldots, X_n = x_n G\}$ want to sign the same message $m$
- they compute an aggregate key

$$\widetilde{X} := \sum_{i=1}^{n} \mu_i X_i \quad \text{with} \quad \mu_i = H(\{X_1, \ldots, X_n\}, X_i)$$

- signature protocol:
  - signers draw nonces $R_i = r_i G$ and send commitments $h_i = H'(R_i)$
  - signers exchange nonces $R_i$
  - signers compute $R = \sum_{i=1}^{n} R_i$ and $c = H(\widetilde{X}, R, m)$
  - signers compute and exchange partial signatures $s_i = r_i + c\mu_i x_i$
  - signers compute $s = \sum_{i=1}^{n} s_i \bmod p$
  - the multi-signature is $\sigma = (R, s)$

# MuSig: Multi-signatures supporting key aggregation

- assume $n$ signers with public keys $\{X_1 = x_1 G, \ldots, X_n = x_n G\}$ want to sign the same message $m$
- they compute an aggregate key

$$\widetilde{X} := \sum_{i=1}^{n} \mu_i X_i \quad \text{with} \quad \mu_i = H(\{X_1, \ldots, X_n\}, X_i)$$

- signature protocol:
  - signers draw nonces $R_i = r_i G$ and send commitments $h_i = H'(R_i)$
  - signers exchange nonces $R_i$
  - signers compute $R = \sum_{i=1}^{n} R_i$ and $c = H(\widetilde{X}, R, m)$
  - signers compute and exchange partial signatures $s_i = r_i + c\mu_i x_i$
  - signers compute $s = \sum_{i=1}^{n} s_i \bmod p$
  - the multi-signature is $\sigma = (R, s)$

# MuSig: Multi-signatures supporting key aggregation

- assume $n$ signers with public keys $\{X_1 = x_1 G, \ldots, X_n = x_n G\}$ want to sign the same message $m$
- they compute an aggregate key

$$\widetilde{X} := \sum_{i=1}^{n} \mu_i X_i \quad \text{with} \quad \mu_i = H(\{X_1, \ldots, X_n\}, X_i)$$

- signature protocol:
  - signers draw nonces $R_i = r_i G$ and send commitments $h_i = H'(R_i)$
  - signers exchange nonces $R_i$
  - signers compute $R = \sum_{i=1}^{n} R_i$ and $c = H(\widetilde{X}, R, m)$
  - signers compute and exchange partial signatures $s_i = r_i + c\mu_i x_i$
  - signers compute $s = \sum_{i=1}^{n} s_i \bmod p$
  - the multi-signature is $\sigma = (R, s)$

# MuSig: Multi-signatures supporting key aggregation

- assume $n$ signers with public keys $\{X_1 = x_1 G, \ldots, X_n = x_n G\}$ want to sign the same message $m$
- they compute an aggregate key

$$\widetilde{X} := \sum_{i=1}^{n} \mu_i X_i \quad \text{with} \quad \mu_i = H(\{X_1, \ldots, X_n\}, X_i)$$

- signature protocol:
  - signers draw nonces $R_i = r_i G$ and send commitments $h_i = H'(R_i)$
  - signers exchange nonces $R_i$
  - signers compute $R = \sum_{i=1}^{n} R_i$ and $c = H(\widetilde{X}, R, m)$
  - signers compute and exchange partial signatures $s_i = r_i + c\mu_i x_i$
  - signers compute $s = \sum_{i=1}^{n} s_i \bmod p$
  - the multi-signature is $\sigma = (R, s)$

# MuSig: Multi-signatures supporting key aggregation

- verification: $(R, s)$ is a valid signature for $m$ under $\widetilde{X}$ if

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

- correctness proof:

$$sG = \sum_{i=1}^{n} s_i G = \underbrace{\sum r_i G}_{R} + H(\widetilde{X}, R, m) \underbrace{\sum \mu_i x_i G}_{\widetilde{X}}$$

- same as standard Schnorr signature for public key $\widetilde{X}$!
- secure in the plain public key model:
  - no assumption on how participants choose their public keys
  - multipliers $\mu_i = H(\{X_1, \ldots, X_n\}, X_i)$ prevent rogue key attacks

# MuSig: Multi-signatures supporting key aggregation

- verification: $(R, s)$ is a valid signature for $m$ under $\widetilde{X}$ if

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

- correctness proof:

$$sG = \sum_{i=1}^{n} s_i G = \underbrace{\sum r_i G}_{R} + H(\widetilde{X}, R, m) \underbrace{\sum \mu_i x_i G}_{\widetilde{X}}$$

- same as standard Schnorr signature for public key $\widetilde{X}$!
- secure in the plain public key model:
  - no assumption on how participants choose their public keys
  - multipliers $\mu_i = H(\{X_1, \ldots, X_n\}, X_i)$ prevent rogue key attacks

# MuSig: Multi-signatures supporting key aggregation

- verification: $(R, s)$ is a valid signature for $m$ under $\widetilde{X}$ if

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

- correctness proof:

$$sG = \sum_{i=1}^{n} s_i G = \underbrace{\sum r_i G}_{R} + H(\widetilde{X}, R, m) \underbrace{\sum \mu_i x_i G}_{\widetilde{X}}$$

- same as standard Schnorr signature for public key $\widetilde{X}$!
- secure in the plain public key model:
  - no assumption on how participants choose their public keys
  - multipliers $\mu_i = H(\{X_1, \ldots, X_n\}, X_i)$ prevent rogue key attacks

# MuSig: Multi-signatures supporting key aggregation

- verification: $(R, s)$ is a valid signature for $m$ under $\widetilde{X}$ if

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

- correctness proof:

$$sG = \sum_{i=1}^{n} s_i G = \underbrace{\sum r_i G}_{R} + H(\widetilde{X}, R, m) \underbrace{\sum \mu_i x_i G}_{\widetilde{X}}$$

- same as standard Schnorr signature for public key $\widetilde{X}$!
- secure in the plain public key model:
  - no assumption on how participants choose their public keys
  - multipliers $\mu_i = H(\{X_1, \ldots, X_n\}, X_i)$ prevent rogue key attacks

# MuSig: Multi-signatures supporting key aggregation

- verification: $(R, s)$ is a valid signature for $m$ under $\widetilde{X}$ if

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

- correctness proof:

$$sG = \sum_{i=1}^{n} s_i G = \underbrace{\sum r_i G}_{R} + H(\widetilde{X}, R, m) \underbrace{\sum \mu_i x_i G}_{\widetilde{X}}$$

- same as standard Schnorr signature for public key $\widetilde{X}$!
- secure in the plain public key model:
  - no assumption on how participants choose their public keys
  - multipliers $\mu_i = H(\{X_1, \ldots, X_n\}, X_i)$ prevent rogue key attacks

# MuSig: Multi-signatures supporting key aggregation

- verification: $(R, s)$ is a valid signature for $m$ under $\widetilde{X}$ if

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

- correctness proof:

$$sG = \sum_{i=1}^{n} s_i G = \underbrace{\sum r_i G}_{R} + H(\widetilde{X}, R, m) \underbrace{\sum \mu_i x_i G}_{\widetilde{X}}$$

- same as standard Schnorr signature for public key $\widetilde{X}$!
- secure in the plain public key model:
  - no assumption on how participants choose their public keys
  - multipliers $\mu_i = H(\{X_1, \ldots, X_n\}, X_i)$ prevent rogue key attacks

# Application: replacing `OP_CHECKMULTISIG`

- using MuSig, an *n*-of-*n* multisig output for public keys $\{X_1, \ldots, X_n\}$ can be replaced by a standard P2PKH output for the aggregate key $\widetilde{X}$
- this improves both efficiency and privacy
  - one public key and one signature to store and verify (versus $n$ pk and $n$ sigs)
  - individual public keys are never revealed
  - the multisig output is indistinguishable from a standard P2PKH output

# Application: replacing `OP_CHECKMULTISIG`

- using MuSig, an $n$-of-$n$ multisig output for public keys $\{X_1, \ldots, X_n\}$ can be replaced by a standard P2PKH output for the aggregate key $\widetilde{X}$
- this improves both efficiency and privacy
  - one public key and one signature to store and verify (versus $n$ pk and $n$ sigs)
  - individual public keys are never revealed
  - the multisig output is indistinguishable from a standard P2PKH output

# Application: replacing `OP_CHECKMULTISIG`

- using MuSig, an $n$-of-$n$ multisig output for public keys $\{X_1, \ldots, X_n\}$ can be replaced by a standard P2PKH output for the aggregate key $\widetilde{X}$
- this improves both efficiency and privacy
  - one public key and one signature to store and verify (versus $n$ pk and $n$ sigs)
  - individual public keys are never revealed
  - the multisig output is indistinguishable from a standard P2PKH output

# Application: replacing `OP_CHECKMULTISIG`

- using MuSig, an *n*-of-*n* multisig output for public keys $\{X_1, \ldots, X_n\}$ can be replaced by a standard P2PKH output for the aggregate key $\widetilde{X}$
- this improves both efficiency and privacy
  - one public key and one signature to store and verify (versus $n$ pk and $n$ sigs)
  - individual public keys are never revealed
  - the multisig output is indistinguishable from a standard P2PKH output

# Application: replacing `OP_CHECKMULTISIG`

- using MuSig, an $n$-of-$n$ multisig output for public keys $\{X_1, \ldots, X_n\}$ can be replaced by a standard P2PKH output for the aggregate key $\widetilde{X}$
- this improves both efficiency and privacy
  - one public key and one signature to store and verify (versus $n$ pk and $n$ sigs)
  - individual public keys are never revealed
  - the multisig output is indistinguishable from a standard P2PKH output

# Outline

Bitcoin Script

Refresher: Schnorr Signatures and MuSig

Taproot

Scriptless Scripts

Discreet Log Contracts

Conclusion

# P2SH (Pay-to-Script-Hash)

- new type of transaction activated in 2012 (BIP 16)
- output only contains a hash of the actual scriptPubKey (*redeem script*) acting as a (binding) commitment
- spending the output requires the redeem script and a valid signature script
- advantages:
  - the sender does not need to know the redeem script when creating the transaction (only the hash)
  - all P2SH addresses "look the same"
  - redeem scripts not contained in the UTXO set anymore (only revealed when spending an output)
- P2SH addresses start with a '3'

# P2SH (Pay-to-Script-Hash)

- new type of transaction activated in 2012 (BIP 16)
- output only contains a hash of the actual scriptPubKey (*redeem script*) acting as a (binding) commitment
- spending the output requires the redeem script and a valid signature script
- advantages:
    - the sender does not need to know the redeem script when creating the transaction (only the hash)
    - all P2SH addresses "look the same"
    - redeem scripts not contained in the UTXO set anymore (only revealed when spending an output)
- P2SH addresses start with a '3'

# P2SH (Pay-to-Script-Hash)

- new type of transaction activated in 2012 (BIP 16)
- output only contains a hash of the actual scriptPubKey (*redeem script*) acting as a (binding) commitment
- spending the output requires the redeem script and a valid signature script
- advantages:
    - the sender does not need to know the redeem script when creating the transaction (only the hash)
    - all P2SH addresses "look the same"
    - redeem scripts not contained in the UTXO set anymore (only revealed when spending an output)
- P2SH addresses start with a '3'

# P2SH (Pay-to-Script-Hash)

- new type of transaction activated in 2012 (BIP 16)
- output only contains a hash of the actual scriptPubKey (*redeem script*) acting as a (binding) commitment
- spending the output requires the redeem script and a valid signature script
- advantages:
    - the sender does not need to know the redeem script when creating the transaction (only the hash)
    - all P2SH addresses "look the same"
    - redeem scripts not contained in the UTXO set anymore (only revealed when spending an output)
- P2SH addresses start with a '3'

# P2SH (Pay-to-Script-Hash)

- new type of transaction activated in 2012 (BIP 16)
- output only contains a hash of the actual scriptPubKey (*redeem script*) acting as a (binding) commitment
- spending the output requires the redeem script and a valid signature script
- advantages:
    - the sender does not need to know the redeem script when creating the transaction (only the hash)
    - all P2SH addresses "look the same"
    - redeem scripts not contained in the UTXO set anymore (only revealed when spending an output)
- P2SH addresses start with a '3'

# P2SH (Pay-to-Script-Hash)

- new type of transaction activated in 2012 (BIP 16)
- output only contains a hash of the actual scriptPubKey (*redeem script*) acting as a (binding) commitment
- spending the output requires the redeem script and a valid signature script
- advantages:
  - the sender does not need to know the redeem script when creating the transaction (only the hash)
  - all P2SH addresses "look the same"
  - redeem scripts not contained in the UTXO set anymore (only revealed when spending an output)
- P2SH addresses start with a '3'

# P2SH (Pay-to-Script-Hash)

- new type of transaction activated in 2012 (BIP 16)
- output only contains a hash of the actual scriptPubKey (*redeem script*) acting as a (binding) commitment
- spending the output requires the redeem script and a valid signature script
- advantages:
  - the sender does not need to know the redeem script when creating the transaction (only the hash)
  - all P2SH addresses "look the same"
  - redeem scripts not contained in the UTXO set anymore (only revealed when spending an output)
- P2SH addresses start with a '3'

# P2SH (Pay-to-Script-Hash)

- new type of transaction activated in 2012 (BIP 16)
- output only contains a hash of the actual scriptPubKey (*redeem script*) acting as a (binding) commitment
- spending the output requires the redeem script and a valid signature script
- advantages:
  - the sender does not need to know the redeem script when creating the transaction (only the hash)
  - all P2SH addresses "look the same"
  - redeem scripts not contained in the UTXO set anymore (only revealed when spending an output)
- P2SH addresses start with a '3'

# MAST (Merkelized Abstract Syntax Trees) [RNS14]

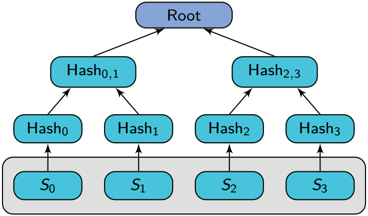
- credited to R. O'Connor and P. Wuille, not deployed yet
- scripts are usually an OR of several conditions
- put all disjunctions in a Merkel tree
- output contains the Merkle root
- to spend a MAST output, the input must contain one of the disjunctions $S_i$, a Merkle proof, and a valid scriptSig for $S_i$

# MAST (Merkelized Abstract Syntax Trees) [RNS14]

- credited to R. O'Connor and P. Wuille, not deployed yet
- scripts are usually an OR of several conditions
- put all disjunctions in a Merkel tree
- output contains the Merkle root
- to spend a MAST output, the input must contain one of the disjunctions $S_i$, a Merkle proof, and a valid scriptSig for $S_i$

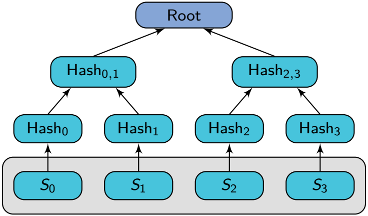
# MAST (Merkelized Abstract Syntax Trees) [RNS14]

- credited to R. O'Connor and P. Wuille, not deployed yet
- scripts are usually an OR of several conditions
- put all disjunctions in a Merkel tree
- output contains the Merkle root
- to spend a MAST output, the input must contain one of the disjunctions $S_i$, a Merkle proof, and a valid scriptSig for $S_i$



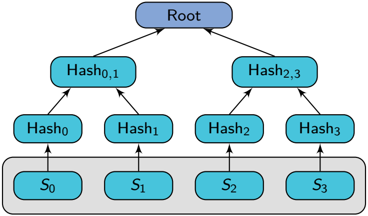
# MAST (Merkelized Abstract Syntax Trees) [RNS14]

- credited to R. O'Connor and P. Wuille, not deployed yet
- scripts are usually an OR of several conditions
- put all disjunctions in a Merkel tree
- output contains the Merkle root
- to spend a MAST output, the input must contain one of the disjunctions $S_i$, a Merkle proof, and a valid scriptSig for $S_i$



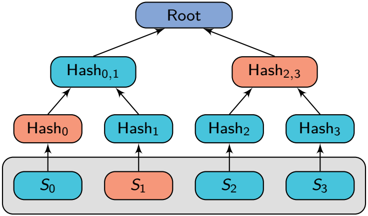
# MAST (Merkelized Abstract Syntax Trees) [RNS14]

- credited to R. O'Connor and P. Wuille, not deployed yet
- scripts are usually an OR of several conditions
- put all disjunctions in a Merkel tree
- output contains the Merkle root
- to spend a MAST output, the input must contain one of the disjunctions $S_i$, a Merkle proof, and a valid scriptSig for $S_i$



# MAST (Merkelized Abstract Syntax Trees) [RNS14]

- credited to R. O'Connor and P. Wuille, not deployed yet
- scripts are usually an OR of several conditions
- put all disjunctions in a Merkel tree
- output contains the Merkle root
- to spend a MAST output, the input must contain one of the disjunctions $S_i$, a Merkle proof, and a valid scriptSig for $S_i$



# Taproot: description

- propose by G. Maxwell [Max18]
- in practice, redeem scripts often have a unanimity clause:

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

- can be achieved indistinguishably from a standard P2PKH output
- let $\widetilde{X}$ be the MuSig aggregate 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 $s_i$)
    $\Rightarrow$ looks like a normal P2PKH spending, $S$ remains forever private
  - $\widetilde{X}$ and $S$ are revealed and a scriptSig $S'$ is provided; valid if $\widetilde{X} + H(\widetilde{X}, S)G = Y$ and $S' \| S$ returns True

# Taproot: description

- propose by G. Maxwell [Max18]
- in practice, redeem scripts often have a unanimity clause:

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

- can be achieved indistinguishably from a standard P2PKH output
- let $\widetilde{X}$ be the MuSig aggregate 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 $s_i$)
      $\Rightarrow$ looks like a normal P2PKH spending, $S$ remains forever private
    - $\widetilde{X}$ and $S$ are revealed and a scriptSig $S'$ is provided; valid if $\widetilde{X} + H(\widetilde{X}, S)G = Y$ and $S' \| S$ returns True

# Taproot: description

- propose by G. Maxwell [Max18]
- in practice, redeem scripts often have a unanimity clause:

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

- can be achieved indistinguishably from a standard P2PKH output
- let $\widetilde{X}$ be the MuSig aggregate 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 $s_i$)
    $\Rightarrow$ looks like a normal P2PKH spending, $S$ remains forever private
  - $\widetilde{X}$ and $S$ are revealed and a scriptSig $S'$ is provided; valid if $\widetilde{X} + H(\widetilde{X}, S)G = Y$ and $S' \| S$ returns True

# Taproot: description

- propose by G. Maxwell [Max18]
- in practice, redeem scripts often have a unanimity clause:

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

- can be achieved indistinguishably from a standard P2PKH output
- let $\widetilde{X}$ be the MuSig aggregate 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 $s_i$)
    $\Rightarrow$ looks like a normal P2PKH spending, $S$ remains forever private
  - $\widetilde{X}$ and $S$ are revealed and a scriptSig $S'$ is provided; valid if $\widetilde{X} + H(\widetilde{X}, S)G = Y$ and $S' \| S$ returns True

# Taproot: description

- propose by G. Maxwell [Max18]
- in practice, redeem scripts often have a unanimity clause:

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

- can be achieved indistinguishably from a standard P2PKH output
- let $\widetilde{X}$ be the MuSig aggregate 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 $s_i$)
    $\Rightarrow$ looks like a normal P2PKH spending, $S$ remains forever private
  - $\widetilde{X}$ and $S$ are revealed and a scriptSig $S'$ is provided; valid if $\widetilde{X} + H(\widetilde{X}, S)G = Y$ and $S' \| S$ returns True

# Taproot: description

- propose by G. Maxwell [Max18]
- in practice, redeem scripts often have a unanimity clause:

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

- can be achieved indistinguishably from a standard P2PKH output
- let $\widetilde{X}$ be the MuSig aggregate 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 $s_i$)
    $\Rightarrow$ looks like a normal P2PKH spending, $S$ remains forever private
  - $\widetilde{X}$ and $S$ are revealed and a scriptSig $S'$ is provided; valid if $\widetilde{X} + H(\widetilde{X}, S)G = Y$ and $S' \| S$ returns True

# Taproot: description

- propose by G. Maxwell [Max18]
- in practice, redeem scripts often have a unanimity clause:

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

- can be achieved indistinguishably from a standard P2PKH output
- let $\widetilde{X}$ be the MuSig aggregate 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 $s_i$)
    $\Rightarrow$ looks like a normal P2PKH spending, $S$ remains forever private
  - $\widetilde{X}$ and $S$ are revealed and a scriptSig $S'$ is provided; valid if $\widetilde{X} + H(\widetilde{X}, S)G = Y$ and $S' \| S$ returns True

# Taproot: description

- propose by G. Maxwell [Max18]
- in practice, redeem scripts often have a unanimity clause:

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

- can be achieved indistinguishably from a standard P2PKH output
- let $\widetilde{X}$ be the MuSig aggregate 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 $s_i$)
    $\Rightarrow$ looks like a normal P2PKH spending, $S$ remains forever private
  - $\widetilde{X}$ and $S$ are revealed and a scriptSig $S'$ is provided; valid if $\widetilde{X} + H(\widetilde{X}, S)G = Y$ and $S' \| S$ returns True

# Taproot: security

- a taproot public key $Y = \widetilde{X} + H(\widetilde{X}, S)G$ acts as a (hiding and binding) commitment on $S$:
  - hiding: $Y$ does not reveal anything about $S$
  - binding: computationally hard to find $(\widetilde{X}', S') \neq (\widetilde{X}, S)$ such that $Y = \widetilde{X}' + H(\widetilde{X}', S')G$ (provably so in the random oracle model)
- unforgeability can be proved in the ROM by extending the proof for Schnorr signatures

# Taproot: security

- a taproot public key $Y = \widetilde{X} + H(\widetilde{X}, S)G$ acts as a (hiding and binding) commitment on $S$:
  - hiding: $Y$ does not reveal anything about $S$
  - binding: computationally hard to find $(\widetilde{X}', S') \neq (\widetilde{X}, S)$ such that $Y = \widetilde{X}' + H(\widetilde{X}', S')G$ (provably so in the random oracle model)
- unforgeability can be proved in the ROM by extending the proof for Schnorr signatures

# Taproot: security

- a taproot public key $Y = \widetilde{X} + H(\widetilde{X}, S)G$ acts as a (hiding and binding) commitment on $S$:
  - hiding: $Y$ does not reveal anything about $S$
  - binding: computationally hard to find $(\widetilde{X}', S') \neq (\widetilde{X}, S)$ such that $Y = \widetilde{X}' + H(\widetilde{X}', S')G$ (provably so in the random oracle model)
- unforgeability can be proved in the ROM by extending the proof for Schnorr signatures

# Taproot: security

- a taproot public key $Y = \widetilde{X} + H(\widetilde{X}, S)G$ acts as a (hiding and binding) commitment on $S$:
    - hiding: $Y$ does not reveal anything about $S$
    - binding: computationally hard to find $(\widetilde{X}', S') \neq (\widetilde{X}, S)$ such that $Y = \widetilde{X}' + H(\widetilde{X}', S')G$ (provably so in the random oracle model)
- unforgeability can be proved in the ROM by extending the proof for Schnorr signatures

# Outline

Bitcoin Script

Refresher: Schnorr Signatures and MuSig

Taproot

Scriptless Scripts

Discreet Log Contracts

Conclusion

# Scriptless Scripts

- proposed by A. Poelstra, originally motivated by Mimblewimble
- goal: enforce smart contracts without publishing the contract in the blockchain, using only standard (P2PKH) transactions
- MuSig is a kind of basic scriptless script (makes $n$-of-$n$ multisig indistinguishable from a standard P2PKH)
- relies on a tool called adaptor signatures

# Scriptless Scripts

- proposed by A. Poelstra, originally motivated by Mimblewimble
- goal: enforce smart contracts without publishing the contract in the blockchain, using only standard (P2PKH) transactions
- MuSig is a kind of basic scriptless script (makes *n*-of-*n* multisig indistinguishable from a standard P2PKH)
- relies on a tool called adaptor signatures

# Scriptless Scripts

- proposed by A. Poelstra, originally motivated by Mimblewimble
- goal: enforce smart contracts without publishing the contract in the blockchain, using only standard (P2PKH) transactions
- MuSig is a kind of basic scriptless script (makes $n$-of-$n$ multisig indistinguishable from a standard P2PKH)
- relies on a tool called adaptor signatures

# Scriptless Scripts

- proposed by A. Poelstra, originally motivated by Mimblewimble
- goal: enforce smart contracts without publishing the contract in the blockchain, using only standard (P2PKH) transactions
- MuSig is a kind of basic scriptless script (makes $n$-of-$n$ multisig indistinguishable from a standard P2PKH)
- relies on a tool called adaptor signatures

# Adaptor signatures

- Schnorr signature $(R = rG, s)$ on $m$ under key $(x, X = xG)$:

$$\begin{aligned} &\text{secret eq.} \qquad s = r + H(X,R,m)x \\ &\text{public eq.} \qquad sG = R + H(X,R,m)X \end{aligned}$$

- assume the signer chooses $(t, T = tG)$ and offsets the signature:

$$s - t = r - t + H(X,R,m)x \tag{1}$$
$$(s-t)G = R - T + H(X,R,m)X \tag{2}$$

- signer reveals adaptor signature $(R, T, \bar{s} = s - t)$:
  → not a valid signature, but (1) can be verified using (2)
- then 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 $T$)

# Adaptor signatures

- Schnorr signature $(R = rG, s)$ on $m$ under key $(x, X = xG)$:

$$\begin{array}{ll} \text{secret eq.} & s = r + H(X, R, m)x \\ \text{public eq.} & sG = R + H(X, R, m)X \end{array}$$

- assume the signer chooses $(t, T = tG)$ and offsets the signature:

$$s - t = r - t + H(X, R, m)x \tag{1}$$

$$(s - t)G = R - T + H(X, R, m)X \tag{2}$$

- signer reveals adaptor signature $(R, T, \bar{s} = s - t)$:
  → not a valid signature, but (1) can be verified using (2)
- then 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 $T$)

# Adaptor signatures

- Schnorr signature $(R = rG, s)$ on $m$ under key $(x, X = xG)$:

$$\begin{array}{ll} \text{secret eq.} & s = r + H(X, R, m)x \\ \text{public eq.} & sG = R + H(X, R, m)X \end{array}$$

- assume the signer chooses $(t, T = tG)$ and offsets the signature:

$$s - t = r - t + H(X, R, m)x \tag{1}$$

$$(s - t)G = R - T + H(X, R, m)X \tag{2}$$

- signer reveals adaptor signature $(R, T, \bar{s} = s - t)$:
  → not a valid signature, but (1) can be verified using (2)
- then 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 $T$)

# Adaptor signatures

- Schnorr signature $(R = rG, s)$ on $m$ under key $(x, X = xG)$:

$$\begin{array}{ll} \text{secret eq.} & s = r + H(X, R, m)x \\ \text{public eq.} & sG = R + H(X, R, m)X \end{array}$$

- assume the signer chooses $(t, T = tG)$ and offsets the signature:

$$s - t = r - t + H(X, R, m)x \tag{1}$$
$$(s - t)G = R - T + H(X, R, m)X \tag{2}$$

- signer reveals adaptor signature $(R, T, \bar{s} = s - t)$:
  → not a valid signature, but (1) can be verified using (2)
- then 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 $T$)

# Adaptor signatures

- Schnorr signature $(R = rG, s)$ on $m$ under key $(x, X = xG)$:

$$\begin{array}{ll} \text{secret eq.} & s = r + H(X, R, m)x \\ \text{public eq.} & sG = R + H(X, R, m)X \end{array}$$

- assume the signer chooses $(t, T = tG)$ and offsets the signature:

$$s - t = r - t + H(X, R, m)x \tag{1}$$

$$(s - t)G = R - T + H(X, R, m)X \tag{2}$$

- signer reveals adaptor signature $(R, T, \bar{s} = s - t)$:
  → not a valid signature, but (1) can be verified using (2)
- then 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 $T$)

# Application: private atomic swaps [Gib17]

- suppose Alice wants to trade 1 bitcoin for 100 litecoins with Bob
- Alice sends 1 bitcoin to a 2-of-2 MuSig public key

$$\widetilde{X} = \mu_A X_A + \mu_B X_B \quad \text{with} \quad \mu_i = H(\{X_A, X_B\}, X_i), i \in \{A, B\}$$

- Bob sends 100 litecoins to a 2-of-2 MuSig public key

$$\widetilde{X}' = \mu'_A X'_A + \mu'_B X'_B \quad \text{with} \quad \mu'_i = H(\{X'_A, X'_B\}, X'_i), i \in \{A, B\}$$

- Alice and Bob must now compute two signatures:
  - $(R = (r_A + r_B)G, s)$ sending the bitcoin to Bob with

$$s = \underbrace{r_A + H(\widetilde{X}, R, m)\mu_A x_A}_{s_A} + \underbrace{r_B + H(\widetilde{X}, R, m)\mu_B x_B}_{s_B}$$

  - $(R' = (r'_A + r'_B)G, s')$ sending the 100 litecoins to Alice with

$$s' = \underbrace{r'_A + H(\widetilde{X}', R', m')\mu'_A x'_A}_{s'_A} + \underbrace{r'_B + H(\widetilde{X}', R', m')\mu'_B x'_B}_{s'_B}$$

# Application: private atomic swaps [Gib17]

- suppose Alice wants to trade 1 bitcoin for 100 litecoins with Bob
- Alice sends 1 bitcoin to a 2-of-2 MuSig public key

$$\widetilde{X} = \mu_A X_A + \mu_B X_B \quad \text{with} \quad \mu_i = H(\{X_A, X_B\}, X_i), i \in \{A, B\}$$

- Bob sends 100 litecoins to a 2-of-2 MuSig public key

$$\widetilde{X}' = \mu'_A X'_A + \mu'_B X'_B \quad \text{with} \quad \mu'_i = H(\{X'_A, X'_B\}, X'_i), i \in \{A, B\}$$

- Alice and Bob must now compute two signatures:
  - $(R = (r_A + r_B)G, s)$ sending the bitcoin to Bob with

$$s = \underbrace{r_A + H(\widetilde{X}, R, m)\mu_A x_A}_{s_A} + \underbrace{r_B + H(\widetilde{X}, R, m)\mu_B x_B}_{s_B}$$

  - $(R' = (r'_A + r'_B)G, s')$ sending the 100 litecoins to Alice with

$$s' = \underbrace{r'_A + H(\widetilde{X}', R', m')\mu'_A x'_A}_{s'_A} + \underbrace{r'_B + H(\widetilde{X}', R', m')\mu'_B x'_B}_{s'_B}$$

# Application: private atomic swaps [Gib17]

- suppose Alice wants to trade 1 bitcoin for 100 litecoins with Bob
- Alice sends 1 bitcoin to a 2-of-2 MuSig public key

$$\widetilde{X} = \mu_A X_A + \mu_B X_B \quad \text{with} \quad \mu_i = H(\{X_A, X_B\}, X_i), i \in \{A, B\}$$

- Bob sends 100 litecoins to a 2-of-2 MuSig public key

$$\widetilde{X}' = \mu'_A X'_A + \mu'_B X'_B \quad \text{with} \quad \mu'_i = H(\{X'_A, X'_B\}, X'_i), i \in \{A, B\}$$

- Alice and Bob must now compute two signatures:
  - $(R = (r_A + r_B)G, s)$ sending the bitcoin to Bob with

$$s = \underbrace{r_A + H(\widetilde{X}, R, m)\mu_A x_A}_{s_A} + \underbrace{r_B + H(\widetilde{X}, R, m)\mu_B x_B}_{s_B}$$

  - $(R' = (r'_A + r'_B)G, s')$ sending the 100 litecoins to Alice with

$$s' = \underbrace{r'_A + H(\widetilde{X}', R', m')\mu'_A x'_A}_{s'_A} + \underbrace{r'_B + H(\widetilde{X}', R', m')\mu'_B x'_B}_{s'_B}$$

# Application: private atomic swaps [Gib17]

- suppose Alice wants to trade 1 bitcoin for 100 litecoins with Bob
- Alice sends 1 bitcoin to a 2-of-2 MuSig public key

$$\widetilde{X} = \mu_A X_A + \mu_B X_B \quad \text{with} \quad \mu_i = H(\{X_A, X_B\}, X_i), i \in \{A, B\}$$

- Bob sends 100 litecoins to a 2-of-2 MuSig public key

$$\widetilde{X}' = \mu'_A X'_A + \mu'_B X'_B \quad \text{with} \quad \mu'_i = H(\{X'_A, X'_B\}, X'_i), i \in \{A, B\}$$

- Alice and Bob must now compute two signatures:
  - $(R = (r_A + r_B)G, s)$ sending the bitcoin to Bob with

$$s = \underbrace{r_A + H(\widetilde{X}, R, m)\mu_A x_A}_{s_A} + \underbrace{r_B + H(\widetilde{X}, R, m)\mu_B x_B}_{s_B}$$

  - $(R' = (r'_A + r'_B)G, s')$ sending the 100 litecoins to Alice with

$$s' = \underbrace{r'_A + H(\widetilde{X}', R', m')\mu'_A x'_A}_{s'_A} + \underbrace{r'_B + H(\widetilde{X}', R', m')\mu'_B x'_B}_{s'_B}$$

# Application: private atomic swaps [Gib17]

- suppose Alice wants to trade 1 bitcoin for 100 litecoins with Bob
- Alice sends 1 bitcoin to a 2-of-2 MuSig public key

$$\widetilde{X} = \mu_A X_A + \mu_B X_B \quad \text{with} \quad \mu_i = H(\{X_A, X_B\}, X_i), i \in \{A, B\}$$

- Bob sends 100 litecoins to a 2-of-2 MuSig public key

$$\widetilde{X}' = \mu'_A X'_A + \mu'_B X'_B \quad \text{with} \quad \mu'_i = H(\{X'_A, X'_B\}, X'_i), i \in \{A, B\}$$

- Alice and Bob must now compute two signatures:
  - $(R = (r_A + r_B)G, s)$ sending the bitcoin to Bob with

$$s = \underbrace{r_A + H(\widetilde{X}, R, m)\mu_A x_A}_{s_A} + \underbrace{r_B + H(\widetilde{X}, R, m)\mu_B x_B}_{s_B}$$

  - $(R' = (r'_A + r'_B)G, s')$ sending the 100 litecoins to Alice with

$$s' = \underbrace{r'_A + H(\widetilde{X}', R', m')\mu'_A x'_A}_{s'_A} + \underbrace{r'_B + H(\widetilde{X}', R', m')\mu'_B x'_B}_{s'_B}$$

# Application: private atomic swaps [Gib17]

- suppose Alice wants to trade 1 bitcoin for 100 litecoins with Bob
- Alice sends 1 bitcoin to a 2-of-2 MuSig public key

$$\widetilde{X} = \mu_A X_A + \mu_B X_B \quad \text{with} \quad \mu_i = H(\{X_A, X_B\}, X_i), i \in \{A, B\}$$

- Bob sends 100 litecoins to a 2-of-2 MuSig public key

$$\widetilde{X}' = \mu'_A X'_A + \mu'_B X'_B \quad \text{with} \quad \mu'_i = H(\{X'_A, X'_B\}, X'_i), i \in \{A, B\}$$

- Alice and Bob must now compute two signatures:
  - $(R = (r_A + r_B)G, s)$ sending the bitcoin to Bob with

$$s = \underbrace{r_A + H(\widetilde{X}, R, m)\mu_A x_A}_{s_A} + \underbrace{r_B + H(\widetilde{X}, R, m)\mu_B x_B}_{s_B}$$

  - $(R' = (r'_A + r'_B)G, s')$ sending the 100 litecoins to Alice with

$$s' = \underbrace{r'_A + H(\widetilde{X}', R', m')\mu'_A x'_A}_{s'_A} + \underbrace{r'_B + H(\widetilde{X}', R', m')\mu'_B x'_B}_{s'_B}$$

# Application: private atomic swaps [Gib17]

- Bob and Alice exchange nonces

$$R_A = r_A G, \quad R_B = r_B G$$
$$R'_A = r'_A G, \quad R'_B = r'_B G$$

- Bob sends two partial adaptor signatures $(R = (r_A + r_B)G, T, \bar{s}_B)$ and $(R' = (r'_A + r'_B)G, T, \bar{s}'_B)$ with the same $(t, T = tG)$

$$\bar{s}_B = s_B - t = r_B - t + H(\widetilde{X}, R, m)\mu_B x_B$$
$$\bar{s}'_B = s'_B - t = r'_B - t + H(\widetilde{X}', R', m')\mu'_B x'_B$$

- Alice checks them and sends her partial signature $s_A$ to Bob
- Bob claims the bitcoin with $s = s_A + s_B$, revealing $s_B$ and hence $t$
- Alice can compute $s'_B = \bar{s}'_B + t$ and claim the 100 litecoins

# Application: private atomic swaps [Gib17]

- Bob and Alice exchange nonces

$$R_A = r_A G, \quad R_B = r_B G$$
$$R'_A = r'_A G, \quad R'_B = r'_B G$$

- Bob sends two partial adaptor signatures $(R = (r_A + r_B)G, T, \bar{s}_B)$ and $(R' = (r'_A + r'_B)G, T, \bar{s}'_B)$ with the same $(t, T = tG)$

$$\bar{s}_B = s_B - t = r_B - t + H(\widetilde{X}, R, m)\mu_B x_B$$
$$\bar{s}'_B = s'_B - t = r'_B - t + H(\widetilde{X}', R', m')\mu'_B x'_B$$

- Alice checks them and sends her partial signature $s_A$ to Bob
- Bob claims the bitcoin with $s = s_A + s_B$, revealing $s_B$ and hence $t$
- Alice can compute $s'_B = \bar{s}'_B + t$ and claim the 100 litecoins

# Application: private atomic swaps [Gib17]

- Bob and Alice exchange nonces

$$R_A = r_A G, \quad R_B = r_B G$$
$$R'_A = r'_A G, \quad R'_B = r'_B G$$

- Bob sends two partial adaptor signatures $(R = (r_A + r_B)G, T, \bar{s}_B)$ and $(R' = (r'_A + r'_B)G, T, \bar{s}'_B)$ with the same $(t, T = tG)$

$$\bar{s}_B = s_B - t = r_B - t + H(\widetilde{X}, R, m)\mu_B x_B$$
$$\bar{s}'_B = s'_B - t = r'_B - t + H(\widetilde{X}', R', m')\mu'_B x'_B$$

- Alice checks them and sends her partial signature $s_A$ to Bob
- Bob claims the bitcoin with $s = s_A + s_B$, revealing $s_B$ and hence $t$
- Alice can compute $s'_B = \bar{s}'_B + t$ and claim the 100 litecoins

# Application: private atomic swaps [Gib17]

- Bob and Alice exchange nonces

$$R_A = r_A G, \quad R_B = r_B G$$
$$R'_A = r'_A G, \quad R'_B = r'_B G$$

- Bob sends two partial adaptor signatures $(R = (r_A + r_B)G, T, \bar{s}_B)$ and $(R' = (r'_A + r'_B)G, T, \bar{s}'_B)$ with the same $(t, T = tG)$

$$\bar{s}_B = s_B - t = r_B - t + H(\widetilde{X}, R, m)\mu_B x_B$$
$$\bar{s}'_B = s'_B - t = r'_B - t + H(\widetilde{X}', R', m')\mu'_B x'_B$$

- Alice checks them and sends her partial signature $s_A$ to Bob
- Bob claims the bitcoin with $s = s_A + s_B$, revealing $s_B$ and hence $t$
- Alice can compute $s'_B = \bar{s}'_B + t$ and claim the 100 litecoins

# Application: private atomic swaps [Gib17]

- Bob and Alice exchange nonces

$$R_A = r_A G, \quad R_B = r_B G$$
$$R'_A = r'_A G, \quad R'_B = r'_B G$$

- Bob sends two partial adaptor signatures $(R = (r_A + r_B)G, T, \bar{s}_B)$ and $(R' = (r'_A + r'_B)G, T, \bar{s}'_B)$ with the same $(t, T = tG)$

$$\bar{s}_B = s_B - t = r_B - t + H(\widetilde{X}, R, m)\mu_B x_B$$
$$\bar{s}'_B = s'_B - t = r'_B - t + H(\widetilde{X}', R', m')\mu'_B x'_B$$

- Alice checks them and sends her partial signature $s_A$ to Bob
- Bob claims the bitcoin with $s = s_A + s_B$, revealing $s_B$ and hence $t$
- Alice can compute $s'_B = \bar{s}'_B + t$ and claim the 100 litecoins

# Application: private atomic swaps [Gib17]

- the swap is perfectly private:
  - the two transactions look “standard” to an external observer
  - nobody can tell that an atomic swap took place or link the two transactions together
- what if Alice or Bob defects once the funds have been sent to the MuSig addresses?
- ⇒ use a time-lock:
  - Alice’s bitcoin can be spent either with the MuSig key $\widetilde{X}$ or by Alice alone after time $\tau_A$
  - Bob’s 100 litecoins can be spent either with the MuSig key $\widetilde{X}'$ or by Bob alone after time $\tau_B$
- note: the time-lock for Bob must be larger than the one for Alice
- using Taproot, this more complex script “sign with $\widetilde{X}$ OR sign with $X_A$ after time $\tau_A$” can be made indistinguishable from a standard P2PKH address

# Application: private atomic swaps [Gib17]

- the swap is perfectly private:
  - the two transactions look “standard” to an external observer
  - nobody can tell that an atomic swap took place or link the two transactions together
- what if Alice or Bob defects once the funds have been sent to the MuSig addresses?
- $\Rightarrow$ use a time-lock:
  - Alice’s bitcoin can be spent either with the MuSig key $\widetilde{X}$ or by Alice alone after time $\tau_A$
  - Bob’s 100 litecoins can be spent either with the MuSig key $\widetilde{X}'$ or by Bob alone after time $\tau_B$
- note: the time-lock for Bob must be larger than the one for Alice
- using Taproot, this more complex script “sign with $\widetilde{X}$ OR sign with $X_A$ after time $\tau_A$” can be made indistinguishable from a standard P2PKH address

# Application: private atomic swaps [Gib17]

- the swap is perfectly private:
    - the two transactions look “standard” to an external observer
    - nobody can tell that an atomic swap took place or link the two transactions together
- what if Alice or Bob defects once the funds have been sent to the MuSig addresses?
- $\Rightarrow$ use a time-lock:
    - Alice's bitcoin can be spent either with the MuSig key $\widetilde{X}$ or by Alice alone after time $\tau_A$
    - Bob's 100 litecoins can be spent either with the MuSig key $\widetilde{X}'$ or by Bob alone after time $\tau_B$
- note: the time-lock for Bob must be larger than the one for Alice
- using Taproot, this more complex script “sign with $\widetilde{X}$ OR sign with $X_A$ after time $\tau_A$” can be made indistinguishable from a standard P2PKH address

# Application: private atomic swaps [Gib17]

- the swap is perfectly private:
    - the two transactions look “standard” to an external observer
    - nobody can tell that an atomic swap took place or link the two transactions together
- what if Alice or Bob defects once the funds have been sent to the MuSig addresses?
- ⇒ use a time-lock:
    - Alice's bitcoin can be spent either with the MuSig key $\widetilde{X}$ or by Alice alone after time $\tau_A$
    - Bob's 100 litecoins can be spent either with the MuSig key $\widetilde{X}'$ or by Bob alone after time $\tau_B$
- note: the time-lock for Bob must be larger than the one for Alice
- using Taproot, this more complex script “sign with $\widetilde{X}$ OR sign with $X_A$ after time $\tau_A$” can be made indistinguishable from a standard P2PKH address

# Application: private atomic swaps [Gib17]

- the swap is perfectly private:
  - the two transactions look "standard" to an external observer
  - nobody can tell that an atomic swap took place or link the two transactions together
- what if Alice or Bob defects once the funds have been sent to the MuSig addresses?
- $\Rightarrow$ use a time-lock:
  - Alice's bitcoin can be spent either with the MuSig key $\widetilde{X}$ or by Alice alone after time $\tau_A$
  - Bob's 100 litecoins can be spent either with the MuSig key $\widetilde{X}'$ or by Bob alone after time $\tau_B$
- note: the time-lock for Bob must be larger than the one for Alice
- using Taproot, this more complex script "sign with $\widetilde{X}$ OR sign with $X_A$ after time $\tau_A$" can be made indistinguishable from a standard P2PKH address

# Application: private atomic swaps [Gib17]

- the swap is perfectly private:
    - the two transactions look "standard" to an external observer
    - nobody can tell that an atomic swap took place or link the two transactions together
- what if Alice or Bob defects once the funds have been sent to the MuSig addresses?
- $\Rightarrow$ use a time-lock:
    - Alice's bitcoin can be spent either with the MuSig key $\widetilde{X}$ or by Alice alone after time $\tau_A$
    - Bob's 100 litecoins can be spent either with the MuSig key $\widetilde{X}'$ or by Bob alone after time $\tau_B$
- note: the time-lock for Bob must be larger than the one for Alice
- using Taproot, this more complex script "sign with $\widetilde{X}$ OR sign with $X_A$ after time $\tau_A$" can be made indistinguishable from a standard P2PKH address

# Application: private atomic swaps [Gib17]

- the swap is perfectly private:
  - the two transactions look "standard" to an external observer
  - nobody can tell that an atomic swap took place or link the two transactions together
- what if Alice or Bob defects once the funds have been sent to the MuSig addresses?
- $\Rightarrow$ use a time-lock:
  - Alice's bitcoin can be spent either with the MuSig key $\widetilde{X}$ or by Alice alone after time $\tau_A$
  - Bob's 100 litecoins can be spent either with the MuSig key $\widetilde{X}'$ or by Bob alone after time $\tau_B$
- note: the time-lock for Bob must be larger than the one for Alice
- using Taproot, this more complex script "sign with $\widetilde{X}$ OR sign with $X_A$ after time $\tau_A$" can be made indistinguishable from a standard P2PKH address

# Application: private atomic swaps [Gib17]

- the swap is perfectly private:
    - the two transactions look “standard” to an external observer
    - nobody can tell that an atomic swap took place or link the two transactions together
- what if Alice or Bob defects once the funds have been sent to the MuSig addresses?
- $\Rightarrow$ use a time-lock:
    - Alice's bitcoin can be spent either with the MuSig key $\widetilde{X}$ or by Alice alone after time $\tau_A$
    - Bob's 100 litecoins can be spent either with the MuSig key $\widetilde{X}'$ or by Bob alone after time $\tau_B$
- note: the time-lock for Bob must be larger than the one for Alice
- using Taproot, this more complex script “sign with $\widetilde{X}$ OR sign with $X_A$ after time $\tau_A$” can be made indistinguishable from a standard P2PKH address

# Application: private atomic swaps [Gib17]

- the swap is perfectly private:
    - the two transactions look "standard" to an external observer
    - nobody can tell that an atomic swap took place or link the two transactions together
- what if Alice or Bob defects once the funds have been sent to the MuSig addresses?
- ⇒ use a time-lock:
    - Alice's bitcoin can be spent either with the MuSig key $\widetilde{X}$ or by Alice alone after time $\tau_A$
    - Bob's 100 litecoins can be spent either with the MuSig key $\widetilde{X}'$ or by Bob alone after time $\tau_B$
- note: the time-lock for Bob must be larger than the one for Alice
- using Taproot, this more complex script "sign with $\widetilde{X}$ OR sign with $X_A$ after time $\tau_A$" can be made indistinguishable from a standard P2PKH address

# Outline

Bitcoin Script

Refresher: Schnorr Signatures and MuSig

Taproot

Scriptless Scripts

Discreet Log Contracts

Conclusion

# Discreet Log Contracts (DLC) [Dry17]

- goal: enforce contracts based on external events
- example: gambling, insurance, . . .
- problem: the blockchain is not aware of external events
- existing solutions: Augur, Gnosis, ChainLink, Oraclize
- Discreet Log Contracts allow conditional payments based on an external event, in a private way
- rely on a tool called anticipated signatures

# Discreet Log Contracts (DLC) [Dry17]

- goal: enforce contracts based on external events
- example: gambling, insurance, . . .
- problem: the blockchain is not aware of external events
- existing solutions: Augur, Gnosis, ChainLink, Oraclize
- Discreet Log Contracts allow conditional payments based on an external event, in a private way
- rely on a tool called anticipated signatures

# Discreet Log Contracts (DLC) [Dry17]

- goal: enforce contracts based on external events
- example: gambling, insurance, . . .
- problem: the blockchain is not aware of external events
- existing solutions: Augur, Gnosis, ChainLink, Oraclize
- Discreet Log Contracts allow conditional payments based on an external event, in a private way
- rely on a tool called anticipated signatures

# Discreet Log Contracts (DLC) [Dry17]

- goal: enforce contracts based on external events
- example: gambling, insurance, . . .
- problem: the blockchain is not aware of external events
- existing solutions: Augur, Gnosis, ChainLink, Oraclize
- Discreet Log Contracts allow conditional payments based on an external event, in a private way
- rely on a tool called anticipated signatures

# Discreet Log Contracts (DLC) [Dry17]

- goal: enforce contracts based on external events
- example: gambling, insurance, . . .
- problem: the blockchain is not aware of external events
- existing solutions: Augur, Gnosis, ChainLink, Oraclize
- Discreet Log Contracts allow conditional payments based on an external event, in a private way
- rely on a tool called anticipated signatures

# Discreet Log Contracts (DLC) [Dry17]

- goal: enforce contracts based on external events
- example: gambling, insurance, . . .
- problem: the blockchain is not aware of external events
- existing solutions: Augur, Gnosis, ChainLink, Oraclize
- Discreet Log Contracts allow conditional payments based on an external event, in a private way
- rely on a tool called anticipated signatures

# Anticipated signatures

- Schnorr signature $(R = rG, s)$ on $m$ under key $(x, X = xG)$:

$$\begin{array}{ll} \text{secret eq.} & s = r + H(X, R, m)x \\ \text{public eq.} & sG = R + H(X, R, m)X \end{array}$$

- assume the signer draws $r$ and reveals $R = rG$ *before* choosing which message to sign
- for any message $m$, anyone can compute

$$S_m := s_m G = R + H(X, R, m)X$$

  where $(R, s_m)$ is the signature on $m$
- $(X, R)$ can be seen as a one-time public key
- $(s_m, S_m)$ can be seen as a key pair associated with $m$

# Anticipated signatures

- Schnorr signature $(R = rG, s)$ on $m$ under key $(x, X = xG)$:

$$\begin{aligned} &\text{secret eq.} && s = r + H(X, R, m)x \\ &\text{public eq.} && sG = R + H(X, R, m)X \end{aligned}$$

- assume the signer draws $r$ and reveals $R = rG$ *before* choosing which message to sign
- for any message $m$, anyone can compute

$$S_m := s_m G = R + H(X, R, m)X$$

  where $(R, s_m)$ is the signature on $m$
- $(X, R)$ can be seen as a one-time public key
- $(s_m, S_m)$ can be seen as a key pair associated with $m$

# Anticipated signatures

- Schnorr signature $(R = rG, s)$ on $m$ under key $(x, X = xG)$:

$$\begin{array}{ll} \text{secret eq.} & s = r + H(X, R, m)x \\ \text{public eq.} & sG = R + H(X, R, m)X \end{array}$$

- assume the signer draws $r$ and reveals $R = rG$ *before* choosing which message to sign
- for any message $m$, anyone can compute

$$S_m := s_m G = R + H(X, R, m)X$$

where $(R, s_m)$ is the signature on $m$

- $(X, R)$ can be seen as a one-time public key
- $(s_m, S_m)$ can be seen as a key pair associated with $m$

# Anticipated signatures

- Schnorr signature $(R = rG, s)$ on $m$ under key $(x, X = xG)$:

$$\begin{array}{ll} \text{secret eq.} & s = r + H(X, R, m)x \\ \text{public eq.} & sG = R + H(X, R, m)X \end{array}$$

- assume the signer draws $r$ and reveals $R = rG$ *before* choosing which message to sign
- for any message $m$, anyone can compute

$$S_m := s_m G = R + H(X, R, m)X$$

where $(R, s_m)$ is the signature on $m$

- $(X, R)$ can be seen as a one-time public key
- $(s_m, S_m)$ can be seen as a key pair associated with $m$

# Anticipated signatures

- Schnorr signature $(R = rG, s)$ on $m$ under key $(x, X = xG)$:

$$\begin{array}{ll} \text{secret eq.} & s = r + H(X, R, m)x \\ \text{public eq.} & sG = R + H(X, R, m)X \end{array}$$

- assume the signer draws $r$ and reveals $R = rG$ *before* choosing which message to sign
- for any message $m$, anyone can compute

$$S_m := s_m G = R + H(X, R, m)X$$

where $(R, s_m)$ is the signature on $m$

- $(X, R)$ can be seen as a one-time public key
- $(s_m, S_m)$ can be seen as a key pair associated with $m$

# DLC: Setup

- Alice and Bob want to execute a contract based on some external event with a predetermined number of outcomes $\{E_1, \ldots, E_n\}$
- Olivia: oracle in charge of observing the event and signing the outcome with public key $(X = xG, R = rG)$
- for each possible outcome $E_i$ of the event, anybody can compute

$$S_i := s_i G = R + H(X, R, E_i)X$$

- for each possible outcome $E_i$ of the event, Alice, resp. Bob compute public keys

$$\widehat{X}_{A,i} = x_A G + S_i, \quad \text{resp.} \quad \widehat{X}_{B,i} = x_B G + S_i$$

# DLC: Setup

- Alice and Bob want to execute a contract based on some external event with a predetermined number of outcomes $\{E_1, \ldots, E_n\}$
- Olivia: oracle in charge of observing the event and signing the outcome with public key $(X = xG, R = rG)$
- for each possible outcome $E_i$ of the event, anybody can compute

$$S_i := s_i G = R + H(X, R, E_i)X$$

- for each possible outcome $E_i$ of the event, Alice, resp. Bob compute public keys

$$\widehat{X}_{A,i} = x_A G + S_i, \quad \text{resp.} \quad \widehat{X}_{B,i} = x_B G + S_i$$

# DLC: Setup

- Alice and Bob want to execute a contract based on some external event with a predetermined number of outcomes $\{E_1, \ldots, E_n\}$
- Olivia: oracle in charge of observing the event and signing the outcome with public key $(X = xG, R = rG)$
- for each possible outcome $E_i$ of the event, anybody can compute

$$S_i := s_i G = R + H(X, R, E_i)X$$

- for each possible outcome $E_i$ of the event, Alice, resp. Bob compute public keys

$$\widehat{X}_{A,i} = x_A G + S_i, \quad \text{resp.} \quad \widehat{X}_{B,i} = x_B G + S_i$$

# DLC: Setup

- Alice and Bob want to execute a contract based on some external event with a predetermined number of outcomes $\{E_1, \ldots, E_n\}$
- Olivia: oracle in charge of observing the event and signing the outcome with public key $(X = xG, R = rG)$
- for each possible outcome $E_i$ of the event, anybody can compute

$$S_i := s_i G = R + H(X, R, E_i)X$$

- for each possible outcome $E_i$ of the event, Alice, resp. Bob compute public keys

$$\widehat{X}_{A,i} = x_A G + S_i, \quad \text{resp.} \quad \widehat{X}_{B,i} = x_B G + S_i$$

# DLC: Creating the contract

- to establish the contract, Alice and Bob create an opening transaction $T^{\mathrm{op}}$ sending funds to a 2-of-2 multisig address
- they also create $n$ pairs of closing transactions: $T^{\mathrm{cl}}_{A,i}$ for Alice and $T^{\mathrm{cl}}_{B,i}$ for Bob
- let $\mathsf{Bal}_{A,i}$ and $\mathsf{Bal}_{B,i}$ be the balances of Alice and Bob in case $E_i$ happens; then:
  - $T^{\mathrm{cl}}_{A,i}$ sends $\mathsf{Bal}_{B,i}$ to $X_B$ and $\mathsf{Bal}_{A,i}$ to script $\widehat{X}_{A,i} \vee (\tau \wedge X_B)$
  - $T^{\mathrm{cl}}_{B,i}$ sends $\mathsf{Bal}_{A,i}$ to $X_A$ and $\mathsf{Bal}_{B,i}$ to script $\widehat{X}_{B,i} \vee (\tau \wedge X_A)$
- once the opening transaction and the $n$ closing transaction pairs have been created, they include the opening transaction in the blockchain

# DLC: Creating the contract

- to establish the contract, Alice and Bob create an opening transaction $T^{\mathrm{op}}$ sending funds to a 2-of-2 multisig address
- they also create $n$ pairs of closing transactions: $T^{\mathrm{cl}}_{A,i}$ for Alice and $T^{\mathrm{cl}}_{B,i}$ for Bob
- let $\mathrm{Bal}_{A,i}$ and $\mathrm{Bal}_{B,i}$ be the balances of Alice and Bob in case $E_i$ happens; then:
  - $T^{\mathrm{cl}}_{A,i}$ sends $\mathrm{Bal}_{B,i}$ to $X_B$ and $\mathrm{Bal}_{A,i}$ to script $\widehat{X}_{A,i} \vee (\tau \wedge X_B)$
  - $T^{\mathrm{cl}}_{B,i}$ sends $\mathrm{Bal}_{A,i}$ to $X_A$ and $\mathrm{Bal}_{B,i}$ to script $\widehat{X}_{B,i} \vee (\tau \wedge X_A)$
- once the opening transaction and the $n$ closing transaction pairs have been created, they include the opening transaction in the blockchain

# DLC: Creating the contract

- to establish the contract, Alice and Bob create an opening transaction $T^{\mathrm{op}}$ sending funds to a 2-of-2 multisig address
- they also create $n$ pairs of closing transactions: $T^{\mathrm{cl}}_{A,i}$ for Alice and $T^{\mathrm{cl}}_{B,i}$ for Bob
- let $\mathsf{Bal}_{A,i}$ and $\mathsf{Bal}_{B,i}$ be the balances of Alice and Bob in case $E_i$ happens; then:
    - $T^{\mathrm{cl}}_{A,i}$ sends $\mathsf{Bal}_{B,i}$ to $X_B$ and $\mathsf{Bal}_{A,i}$ to script $\widehat{X}_{A,i} \vee (\tau \wedge X_B)$
    - $T^{\mathrm{cl}}_{B,i}$ sends $\mathsf{Bal}_{A,i}$ to $X_A$ and $\mathsf{Bal}_{B,i}$ to script $\widehat{X}_{B,i} \vee (\tau \wedge X_A)$
- once the opening transaction and the $n$ closing transaction pairs have been created, they include the opening transaction in the blockchain

# DLC: Creating the contract

- to establish the contract, Alice and Bob create an opening transaction $T^{\mathrm{op}}$ sending funds to a 2-of-2 multisig address
- they also create $n$ pairs of closing transactions: $T^{\mathrm{cl}}_{A,i}$ for Alice and $T^{\mathrm{cl}}_{B,i}$ for Bob
- let $\mathsf{Bal}_{A,i}$ and $\mathsf{Bal}_{B,i}$ be the balances of Alice and Bob in case $E_i$ happens; then:
    - $T^{\mathrm{cl}}_{A,i}$ sends $\mathsf{Bal}_{B,i}$ to $X_B$ and $\mathsf{Bal}_{A,i}$ to script $\widehat{X}_{A,i} \vee (\tau \wedge X_B)$
    - $T^{\mathrm{cl}}_{B,i}$ sends $\mathsf{Bal}_{A,i}$ to $X_A$ and $\mathsf{Bal}_{B,i}$ to script $\widehat{X}_{B,i} \vee (\tau \wedge X_A)$
- once the opening transaction and the $n$ closing transaction pairs have been created, they include the opening transaction in the blockchain

# DLC: Creating the contract

- to establish the contract, Alice and Bob create an opening transaction $T^{\mathrm{op}}$ sending funds to a 2-of-2 multisig address
- they also create $n$ pairs of closing transactions: $T^{\mathrm{cl}}_{A,i}$ for Alice and $T^{\mathrm{cl}}_{B,i}$ for Bob
- let $\mathsf{Bal}_{A,i}$ and $\mathsf{Bal}_{B,i}$ be the balances of Alice and Bob in case $E_i$ happens; then:
  - $T^{\mathrm{cl}}_{A,i}$ sends $\mathsf{Bal}_{B,i}$ to $X_B$ and $\mathsf{Bal}_{A,i}$ to script $\widehat{X}_{A,i} \vee (\tau \wedge X_B)$
  - $T^{\mathrm{cl}}_{B,i}$ sends $\mathsf{Bal}_{A,i}$ to $X_A$ and $\mathsf{Bal}_{B,i}$ to script $\widehat{X}_{B,i} \vee (\tau \wedge X_A)$
- once the opening transaction and the $n$ closing transaction pairs have been created, they include the opening transaction in the blockchain

# DLC: Creating the contract

- to establish the contract, Alice and Bob create an opening transaction $T^{\mathrm{op}}$ sending funds to a 2-of-2 multisig address
- they also create $n$ pairs of closing transactions: $T^{\mathrm{cl}}_{A,i}$ for Alice and $T^{\mathrm{cl}}_{B,i}$ for Bob
- let $\mathrm{Bal}_{A,i}$ and $\mathrm{Bal}_{B,i}$ be the balances of Alice and Bob in case $E_i$ happens; then:
    - $T^{\mathrm{cl}}_{A,i}$ sends $\mathrm{Bal}_{B,i}$ to $X_B$ and $\mathrm{Bal}_{A,i}$ to script $\widehat{X}_{A,i} \vee (\tau \wedge X_B)$
    - $T^{\mathrm{cl}}_{B,i}$ sends $\mathrm{Bal}_{A,i}$ to $X_A$ and $\mathrm{Bal}_{B,i}$ to script $\widehat{X}_{B,i} \vee (\tau \wedge X_A)$
- once the opening transaction and the $n$ closing transaction pairs have been created, they include the opening transaction in the blockchain

# DLC: Executing the contract

- when the external event happens, Olivia signs the observed outcome $E_{\bar{\imath}}$, revealing $s_{\bar{\imath}}$
- Alice and Bob can compute resp. $x_A + s_{\bar{\imath}}$ and $x_B + s_{\bar{\imath}}$; one of them (e.g. Alice) broadcasts the corresponding closing transaction $T^{\mathrm{cl}}_{A,\bar{\imath}}$; then:
    - Alice can claim $\mathrm{Bal}_{A,\bar{\imath}}$ using $\widehat{X}_{A,\bar{\imath}} = (x_A + s_{\bar{\imath}})G$
    - Bob can claim $\mathrm{Bal}_{B,\bar{\imath}}$ using $X_B$
- if Bob tries to cheat and sends an incorrect closing transaction $T^{\mathrm{cl}}_{B,j}$, $j \neq \bar{\imath}$, he is unable to claim the output worth $\mathrm{Bal}_{B,j}$ controlled by script $\widehat{X}_{B,j} \vee (\tau \wedge X_A)$, which can be claimed by Alice after time $\tau$
- NB: funds cannot be locked (Alice's closing transactions always return all funds to Bob after time $\tau$ and vice-versa)

# DLC: Executing the contract

- when the external event happens, Olivia signs the observed outcome $E_{\bar{\imath}}$, revealing $s_{\bar{\imath}}$
- Alice and Bob can compute resp. $x_A + s_{\bar{\imath}}$ and $x_B + s_{\bar{\imath}}$; one of them (e.g. Alice) broadcasts the corresponding closing transaction $T^{\mathrm{cl}}_{A,\bar{\imath}}$; then:
  - Alice can claim $\mathrm{Bal}_{A,\bar{\imath}}$ using $\widehat{X}_{A,\bar{\imath}} = (x_A + s_{\bar{\imath}})G$
  - Bob can claim $\mathrm{Bal}_{B,\bar{\imath}}$ using $X_B$
- if Bob tries to cheat and sends an incorrect closing transaction $T^{\mathrm{cl}}_{B,j}$, $j \neq \bar{\imath}$, he is unable to claim the output worth $\mathrm{Bal}_{B,j}$ controlled by script $\widehat{X}_{B,j} \vee (\tau \wedge X_A)$, which can be claimed by Alice after time $\tau$
- NB: funds cannot be locked (Alice's closing transactions always return all funds to Bob after time $\tau$ and vice-versa)

# DLC: Executing the contract

- when the external event happens, Olivia signs the observed outcome $E_{\bar{\imath}}$, revealing $s_{\bar{\imath}}$
- Alice and Bob can compute resp. $x_A + s_{\bar{\imath}}$ and $x_B + s_{\bar{\imath}}$; one of them (e.g. Alice) broadcasts the corresponding closing transaction $T^{\mathrm{cl}}_{A,\bar{\imath}}$; then:
    - Alice can claim $\mathsf{Bal}_{A,\bar{\imath}}$ using $\widehat{X}_{A,\bar{\imath}} = (x_A + s_{\bar{\imath}})G$
    - Bob can claim $\mathsf{Bal}_{B,\bar{\imath}}$ using $X_B$
- if Bob tries to cheat and sends an incorrect closing transaction $T^{\mathrm{cl}}_{B,j}$, $j \neq \bar{\imath}$, he is unable to claim the output worth $\mathsf{Bal}_{B,j}$ controlled by script $\widehat{X}_{B,j} \vee (\tau \wedge X_A)$, which can be claimed by Alice after time $\tau$
- NB: funds cannot be locked (Alice's closing transactions always return all funds to Bob after time $\tau$ and vice-versa)

# DLC: Executing the contract

- when the external event happens, Olivia signs the observed outcome $E_{\bar{\imath}}$, revealing $s_{\bar{\imath}}$
- Alice and Bob can compute resp. $x_A + s_{\bar{\imath}}$ and $x_B + s_{\bar{\imath}}$; one of them (e.g. Alice) broadcasts the corresponding closing transaction $T^{\mathrm{cl}}_{A,\bar{\imath}}$; then:
  - Alice can claim $\mathrm{Bal}_{A,\bar{\imath}}$ using $\widehat{X}_{A,\bar{\imath}} = (x_A + s_{\bar{\imath}})G$
  - Bob can claim $\mathrm{Bal}_{B,\bar{\imath}}$ using $X_B$
- if Bob tries to cheat and sends an incorrect closing transaction $T^{\mathrm{cl}}_{B,j}$, $j \neq \bar{\imath}$, he is unable to claim the output worth $\mathrm{Bal}_{B,j}$ controlled by script $\widehat{X}_{B,j} \vee (\tau \wedge X_A)$, which can be claimed by Alice after time $\tau$
- NB: funds cannot be locked (Alice's closing transactions always return all funds to Bob after time $\tau$ and vice-versa)

# DLC: Executing the contract

- when the external event happens, Olivia signs the observed outcome $E_{\bar{\imath}}$, revealing $s_{\bar{\imath}}$
- Alice and Bob can compute resp. $x_A + s_{\bar{\imath}}$ and $x_B + s_{\bar{\imath}}$; one of them (e.g. Alice) broadcasts the corresponding closing transaction $T^{\mathrm{cl}}_{A,\bar{\imath}}$; then:
  - Alice can claim $\mathsf{Bal}_{A,\bar{\imath}}$ using $\widehat{X}_{A,\bar{\imath}} = (x_A + s_{\bar{\imath}})G$
  - Bob can claim $\mathsf{Bal}_{B,\bar{\imath}}$ using $X_B$
- if Bob tries to cheat and sends an incorrect closing transaction $T^{\mathrm{cl}}_{B,j}$, $j \neq \bar{\imath}$, he is unable to claim the output worth $\mathsf{Bal}_{B,j}$ controlled by script $\widehat{X}_{B,j} \vee (\tau \wedge X_A)$, which can be claimed by Alice after time $\tau$
- NB: funds cannot be locked (Alice's closing transactions always return all funds to Bob after time $\tau$ and vice-versa)

# DLC: Executing the contract

- when the external event happens, Olivia signs the observed outcome $E_{\bar{\imath}}$, revealing $s_{\bar{\imath}}$
- Alice and Bob can compute resp. $x_A + s_{\bar{\imath}}$ and $x_B + s_{\bar{\imath}}$; one of them (e.g. Alice) broadcasts the corresponding closing transaction $T^{\mathrm{cl}}_{A,\bar{\imath}}$; then:
    - Alice can claim $\mathrm{Bal}_{A,\bar{\imath}}$ using $\widehat{X}_{A,\bar{\imath}} = (x_A + s_{\bar{\imath}})G$
    - Bob can claim $\mathrm{Bal}_{B,\bar{\imath}}$ using $X_B$
- if Bob tries to cheat and sends an incorrect closing transaction $T^{\mathrm{cl}}_{B,j}$, $j \neq \bar{\imath}$, he is unable to claim the output worth $\mathrm{Bal}_{B,j}$ controlled by script $\widehat{X}_{B,j} \vee (\tau \wedge X_A)$, which can be claimed by Alice after time $\tau$
- NB: funds cannot be locked (Alice's closing transactions always return all funds to Bob after time $\tau$ and vice-versa)

# Outline

Bitcoin Script

Refresher: Schnorr Signatures and MuSig

Taproot

Scriptless Scripts

Discreet Log Contracts

Conclusion

# Conclusion

- Schnorr signatures can help improve privacy and fungibility:
    - multisigs made indistinguishable from P2PKH (MuSig)
    - complex scripts made indistinguishable from P2PKH (Taproot)
    - stealthy enforcement of contracts (Scriptless Scripts, Discreet Log Contracts)
- all this also implies space and computational gains (less data to verify and store in the blockchain)
- BIP for Schnorr is currently under review

# Conclusion

- Schnorr signatures can help improve privacy and fungibility:
  - multisigs made indistinguishable from P2PKH (MuSig)
  - complex scripts made indistinguishable from P2PKH (Taproot)
  - stealthy enforcement of contracts (Scriptless Scripts, Discreet Log Contracts)
- all this also implies space and computational gains (less data to verify and store in the blockchain)
- BIP for Schnorr is currently under review

# Conclusion

- Schnorr signatures can help improve privacy and fungibility:
    - multisigs made indistinguishable from P2PKH (MuSig)
    - complex scripts made indistinguishable from P2PKH (Taproot)
    - stealthy enforcement of contracts (Scriptless Scripts, Discreet Log Contracts)
- all this also implies space and computational gains (less data to verify and store in the blockchain)
- BIP for Schnorr is currently under review

# Conclusion

- Schnorr signatures can help improve privacy and fungibility:
    - multisigs made indistinguishable from P2PKH (MuSig)
    - complex scripts made indistinguishable from P2PKH (Taproot)
    - stealthy enforcement of contracts (Scriptless Scripts, Discreet Log Contracts)
- all this also implies space and computational gains (less data to verify and store in the blockchain)
- BIP for Schnorr is currently under review

# Conclusion

- Schnorr signatures can help improve privacy and fungibility:
    - multisigs made indistinguishable from P2PKH (MuSig)
    - complex scripts made indistinguishable from P2PKH (Taproot)
    - stealthy enforcement of contracts (Scriptless Scripts, Discreet Log Contracts)
- all this also implies space and computational gains (less data to verify and store in the blockchain)
- BIP for Schnorr is currently under review

# Conclusion

- Schnorr signatures can help improve privacy and fungibility:
    - multisigs made indistinguishable from P2PKH (MuSig)
    - complex scripts made indistinguishable from P2PKH (Taproot)
    - stealthy enforcement of contracts (Scriptless Scripts, Discreet Log Contracts)
- all this also implies space and computational gains (less data to verify and store in the blockchain)
- BIP for Schnorr is currently under review

# The end…

Thanks for your attention!

Comments or questions?

# References I

- Thaddeus Dryja. Discreet Log Contracts, 2017. Available at https://adiabat.github.io/dlc.pdf.
- Adam Gibson. Flipping the scriptless script on Schnorr, 2017. Available at https://joinmarket.me/blog/blog/flipping-the-scriptless-script-on-schnorr.
- Gregory Maxwell. Taproot: Privacy preserving switchable scripting, January 2018. Post on Bitcoin development mailing list, https://lists.linuxfoundation.org/pipermail/bitcoin-dev/2018-January/015614.html.
- Tier Nolan. Alt chains and atomic transfers, May 2013. BitcoinTalk post, https://bitcointalk.org/index.php?topic=193281.0.
- Jeremy Rubin, Manali Naik, and Nitya Subramanian. Merkelized Abstract Syntax Trees, 2014. Available at https://rubin.io/public/pdfs/858report.pdf.

# References II

- Claus-Peter Schnorr. Efficient Identification and Signatures for Smart Cards. In *Advances in Cryptology - CRYPTO '89*, pages 239–252.
- Claus-Peter Schnorr. Efficient Signature Generation by Smart Cards. *J. Cryptology*, 4(3):161–174, 1991.