Proof Systems and SNARKs

Benedikt Bünz, Stanford

Managing assets on a blockchain: key principles

- **Universal verifiability** of blockchain rules
  - ⇒ all data written to the blockchain is public; everyone can verify
  - ⇒ added benefit: interoperability between chains

- Assets are **controlled by signature keys**
  - ⇒ assets <u>cannot</u> be transferred without a valid signature

    (of course, users can choose to custody their keys)

Privacy?

Naïve reasoning:

universal verifiability ⇒ blockchain data is public

⇒ all transactions data is public

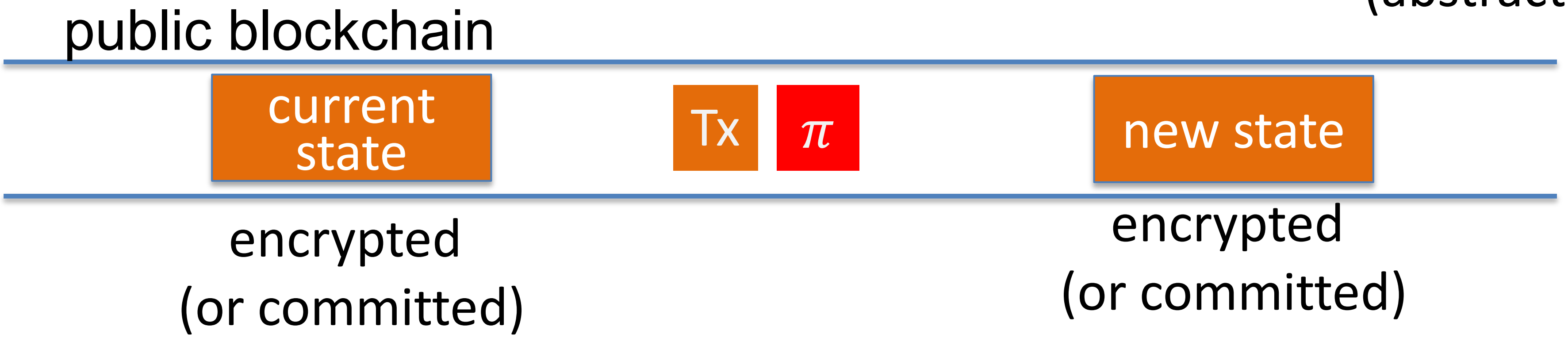
otherwise, how we can verify Tx?

not quite …

crypto magic ⇒ private Tx on a publicly verifiable blockchain

Public blockchain & universal verifiability

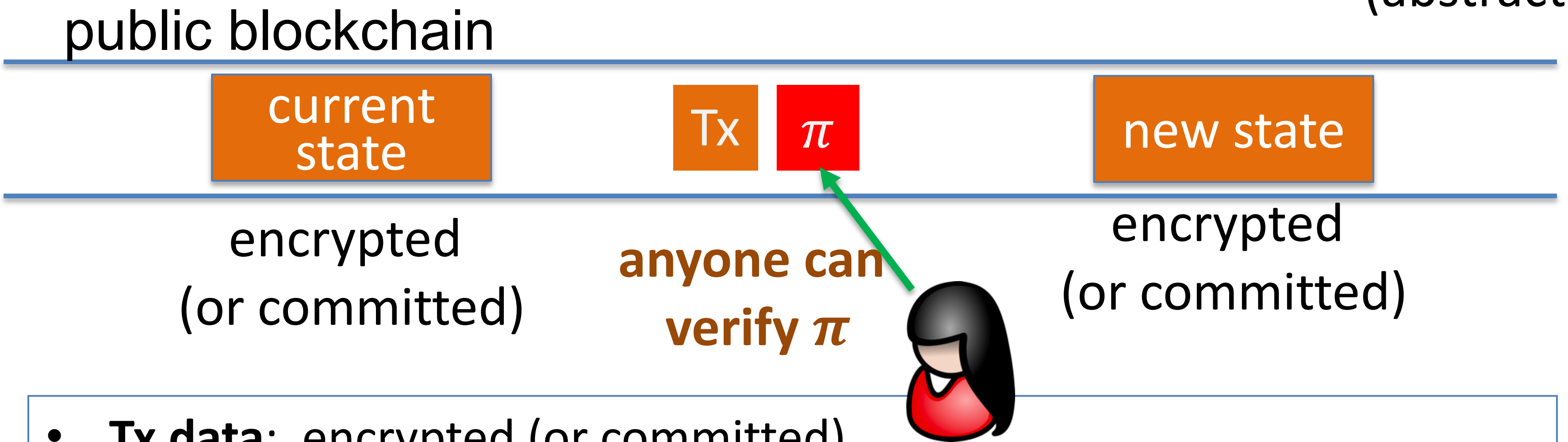
(abstractly)



- **Tx data**: encrypted (or committed)
- **Proof π**: *zero-knowledge proof* that (reveals nothing about Tx data)
  (1) plaintext Tx data is consistent with plaintext current state
  (2) plaintext new state is correct

Public blockchain & universal verifiability

(abstractly)

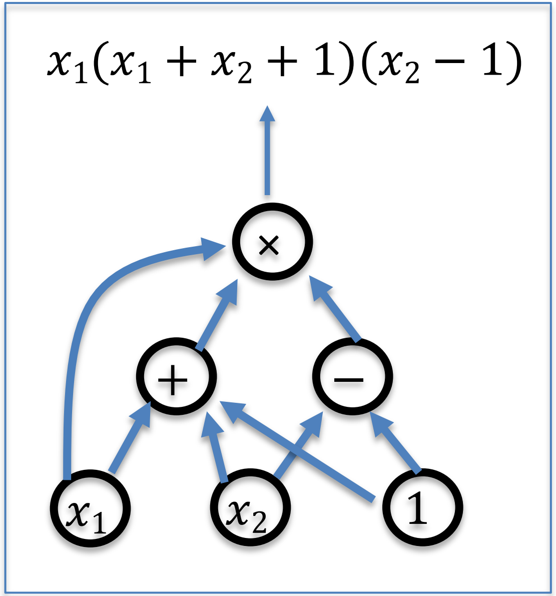


- **Tx data**: encrypted (or committed)
- **Proof π**: *zero-knowledge proof* that (reveals nothing about Tx data)
  (1) plaintext Tx data is consistent with plaintext current state
  (2) plaintext new state is correct

Zero Knowledge Proof Systems

(1) arithmetic circuits

- Fix a finite field $\mathbb{F} = \{0, \dots, p-1\}$ for some prime p>2.
- **Arithmetic circuit:** $C\colon \mathbb{F}^n \rightarrow \mathbb{F}$
  - directed acyclic graph (DAG) where
    - internal nodes are labeled +, −, or ×
    - inputs are labeled $1, x_1, \dots, x_n$
  - defines an n-variate polynomial with an evaluation recipe
- $|C| = \#$ multiplication gates in C



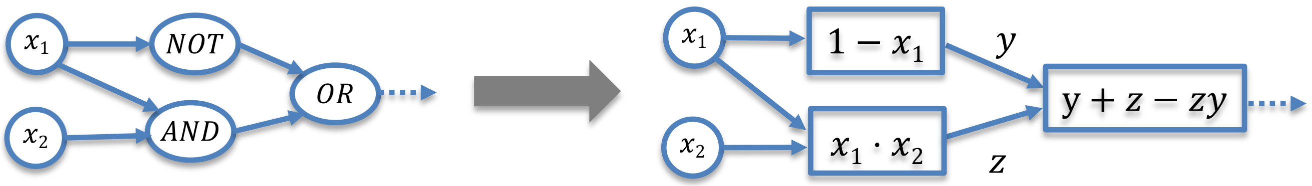
Boolean circuits as arithmetic circuits

Boolean circuits: circuits with AND, OR, NOT gates

Encoding a boolean circuit as an arithmetic circuit over $\mathbb{F}_p$:

- AND(x, y) encoded as $x \cdot y$
- OR(x, y) encoded as $x + y - x \cdot y$
- NOT(x) encoded as $1 - x$

x	y	OR(x, y)
0	0	0
0	1	1
1	0	1
1	1	1



Interesting arithmetic circuits

- $C_{hash}(h, \mathbf{m})$: outputs 0 if SHA256($\mathbf{m}$) = h , and ≠0 otherwise

  $C_{hash}(h, \mathbf{m}) = (h - \text{SHA256}(\mathbf{m}))$, $|C_{hash}| \approx$ 20K gates

- $C_{sig}((pk, m), \sigma)$: output 0 if σ is a valid ECDSA signature of m under pk

(2) non-interactive proof systems (for NP)

Public arithmetic circuit: $C(x, w) \rightarrow \mathbb{F}_p$

public statement in $\mathbb{F}_p^n$ → x ; w ← secret witness in $\mathbb{F}_p^m$

Let $x \in \mathbb{F}_p^n$. Two standard goals for prover P:

(1) **Soundness**: convince Verifier that $\exists w$ s.t. $C(x, w) = 0$

(e.g., $\exists w$ such that $[\, H(w) = x$ and $0 < w < 2^{60} \,]$)

(2) **Knowledge**: convince Verifier that P "knows" w s.t. $C(x, w) = 0$

(e.g., P knows a w such that $H(w) = x$)

The trivial proof system

Why can't prover simply send $\boldsymbol{w}$ to verifier?

- Verifier checks if $C(\boldsymbol{x}, \boldsymbol{w}) = 0$ and accepts if so.

<u>Problems with this</u>:

(1) $\boldsymbol{w}$ might be secret: prover cannot reveal $\boldsymbol{w}$ to verifier

(2) $\boldsymbol{w}$ might be long: we want a "short" proof

(3) computing $C(\boldsymbol{x}, \boldsymbol{w})$ may be hard: want to minimize Verifier's work

Non-interactive Proof Systems (for NP)

Public arithmetic circuit: $C(x, w) \to \mathbb{F}_p$

public input in $\mathbb{F}_p^n$ → x; secret witness in $\mathbb{F}_p^m$ → w

setup: $\mathbf{S}(C) \to$ public parameters (S_p, S_v)

Prover $P(S_p, x, w)$ — Verifier $V(S_v, x, \pi)$

proof π

output accept or reject

Non-interactive Proof Systems (for NP)

A **non-interactive proof system** is a triple (S, P, V):

- **S**(C) → public parameters (S_p, S_v) for prover and verifier
- **P**(S_p, $\boldsymbol{x}$, $\boldsymbol{w}$) → proof π
- **V**(S_v, $\boldsymbol{x}$, $\boldsymbol{\pi}$) → accept or reject

proof systems: properties (informal)

Prover P(***pp***, $\boldsymbol{x}$, $\boldsymbol{w}$) Verifier V (***pp***, $\boldsymbol{x}$, $\boldsymbol{\pi}$)

proof π

accept or reject

Complete: $\forall x, w$: $C(\boldsymbol{x}, \boldsymbol{w}) = 0 \Rightarrow$ V(S_v, x, P(S_p, $\boldsymbol{x}$, $\boldsymbol{w}$)) = accept

Proof of knowledge: V accepts $\Rightarrow$ P "knows" $\boldsymbol{w}$ s.t. $C(\boldsymbol{x}, \boldsymbol{w}) = 0$

Zero knowledge (optional): $(\boldsymbol{x}, \pi)$ "reveals nothing" about $\boldsymbol{w}$

(b) Zero knowledge

(S, P, V) is **zero knowledge** if proof π "reveals nothing" about $\boldsymbol{w}$

Formally: (S, P, V) is **zero knowledge** for a circuit C

if there is an efficient simulator ***Sim***,

such that for all $x \in \mathbb{F}_p^n$ s.t. $\exists w: C(x, w) = 0$ the distribution:

(S_p, S_v, x, π) where $(S_p, S_v) \leftarrow S(C)$, $\pi \leftarrow P(x, \boldsymbol{w})$

is indistinguishable from the distribution:

(S_p, S_v, x, π) where $(S_p, S_v, \pi) \leftarrow \textit{Sim}(x)$

key point: ***Sim***(x) simulates proof π without knowledge of $\boldsymbol{w}$

(3) Succinct arguments: SNARKs

Goal: P wants to show that it knows $\boldsymbol{w}$ s.t. $C(\boldsymbol{x}, \boldsymbol{w}) = 0$

Succinct:

- Proof π should be **short** [i.e., $|\pi| = O(\mathbf{log}(|\boldsymbol{C}|), \lambda)$]

- Verifying π should be **fast** [i.e., time(V) = $O(|x|, \mathbf{log}(|\boldsymbol{C}|), \lambda)$]

note: if SNARK is zero-knowledge, then called a **zkSNARK**

(3) Succinct arguments: SNARKs

Goal: P wants to show that it knows $\boldsymbol{w}$ s.t. $C(\boldsymbol{x}, \boldsymbol{w}) = 1$

verifier cannot read C !! Instead,
V relies on setup(C) to pre-process (summarize) C in S_v

<u>**Succinct**</u>:

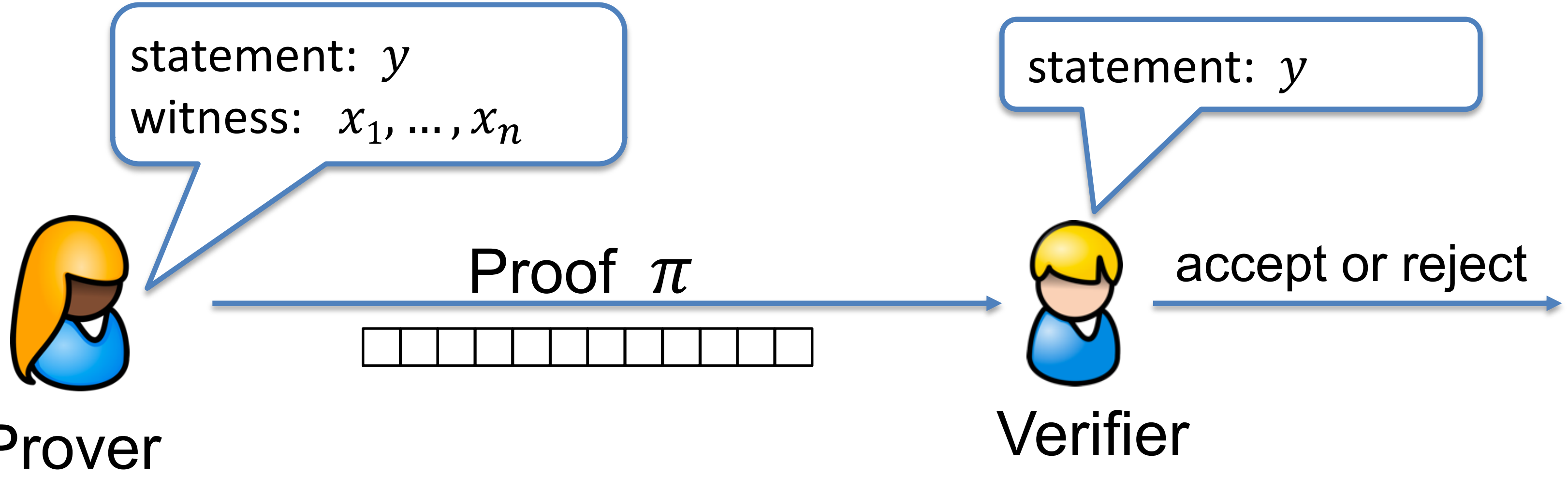
- Proof π should be **short** [i.e., $|\pi| = O(\mathbf{log}(\ldots \lambda)$]
- Verifying π should be **fast** [i.e., time(V) = $O(|x|, \mathbf{log}(|C|), \lambda)$]

note: if SNARK is zero-knowledge, then called a **zkSNARK**

An example

Prover says: I know $(x_1, \ldots, x_n) \in X$ such that $H(x_1, \ldots, x_n) = y$

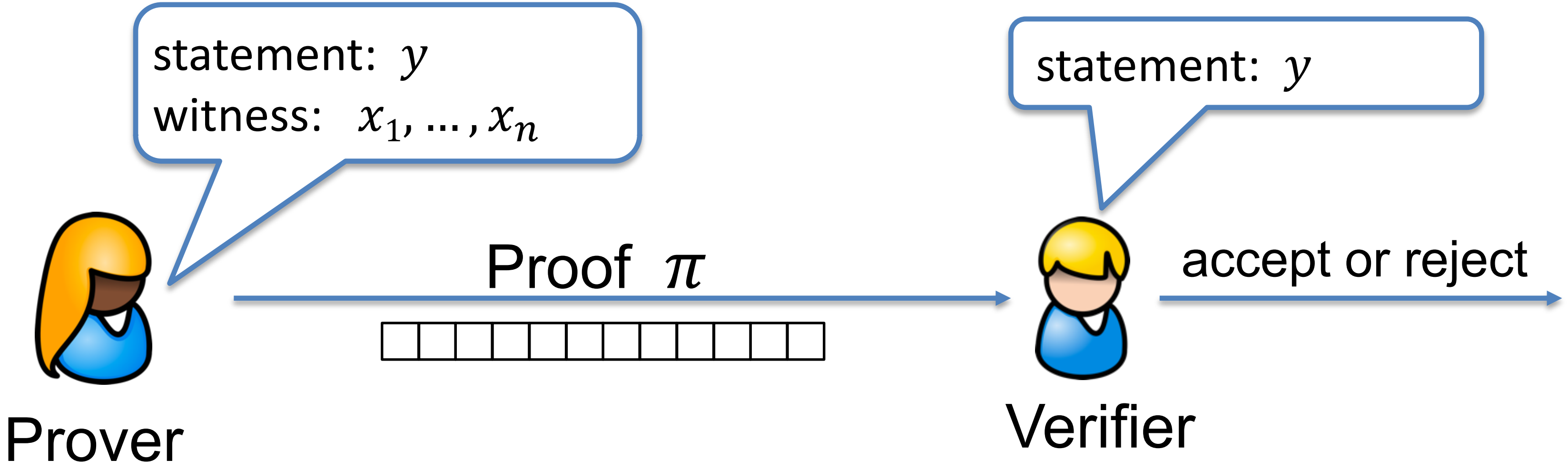
SNARK: $\text{size}(\pi)$ and $\text{VerifyTime}(\pi)$ should be $O(\log n)$!!



An example

How is this possible ???

SNARK: $\text{size}(\pi)$ and $\text{VerifyTime}(\pi)$ should be $O(\log n)$!!



Types of pre-processing Setup

Recall setup for circuit C: **S**(C) → public parameters (S_p, S_v)

Types of setup:

trusted setup per circuit: **S**(C) uses data that must be kept secret

compromised trusted setup ⇒ can prove false statements

updatable universal trusted setup: (S_p, S_v) can be updated by anyone

transparent: **S**() does not use secret data (no trusted setup)

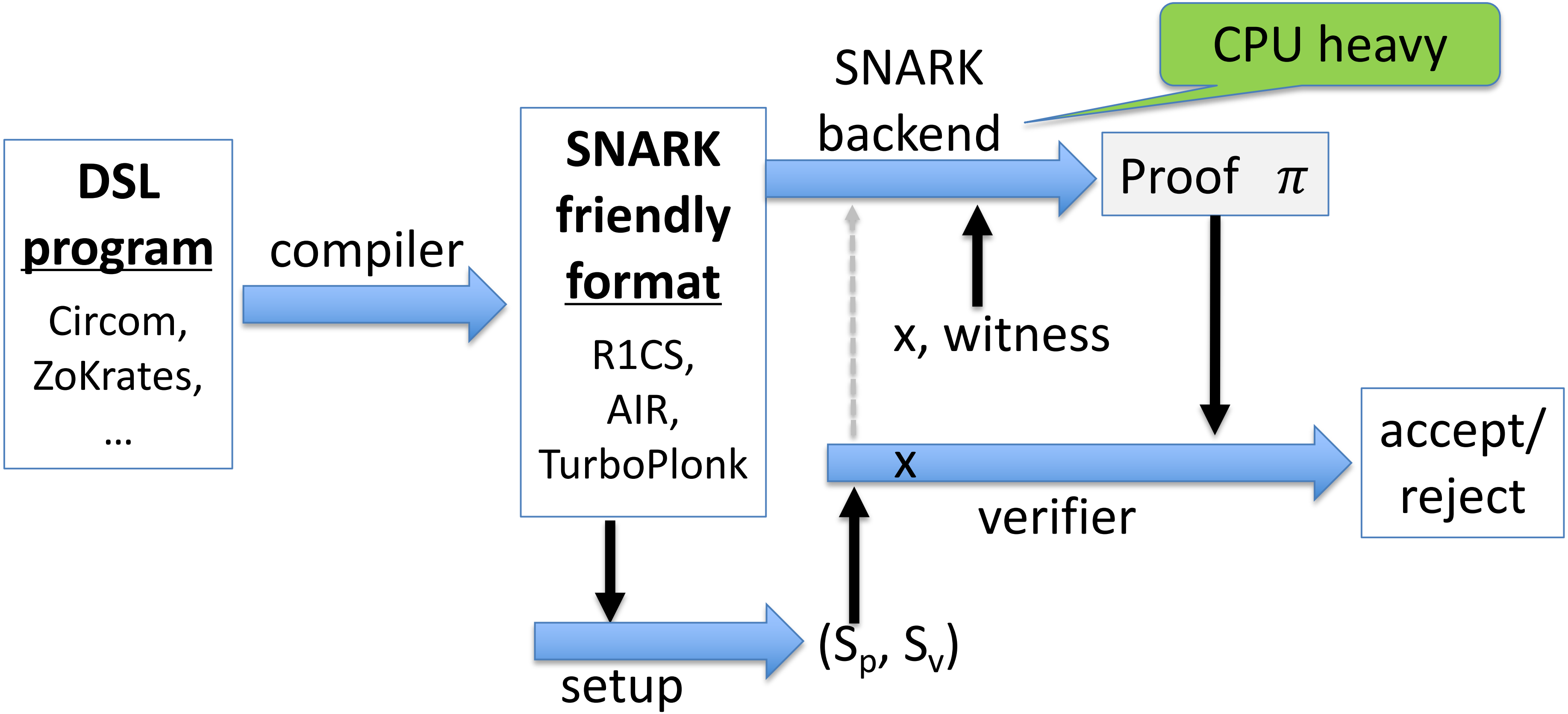
Significant progress in recent years

- **Kilian'92, Micali'94**: succinct transparent arguments from PCP
  - impractical prover time
- **GGPR'13**, **Groth'16, …**: linear prover time, **constant size proof** ($O_\lambda(1)$)
  - **trusted setup per circuit** (setup alg. uses secret randomness)
  - compromised setup $\Rightarrow$ proofs of false statements
- **Sonic'19, Marlin'19, Plonk'19**, … : universal trusted setup
- **DARK'19, Halo'19, STARK**, … : no trusted setup (transparent)

Types of SNARKs (partial list)

	size of $\lvert\pi\rvert$	size of $\lvert S_p\rvert$	verifier time	trusted setup?
Groth'16	O(1)	$O(\lvert C \rvert)$	O(1)	yes/per circuit
PLONK/MARLIN	O(1)	$O(\lvert C \rvert)$	O(1)	yes/updatable
Bulletproofs	$O(\log\lvert C \rvert)$	O(1)	$O(\lvert C \rvert)$	no
STARK	$O(\log\lvert C \rvert)$	O(1)	$O(\log\lvert C \rvert)$	no
DARK	$O(\log\lvert C \rvert)$	O(1)	$O(\log\lvert C \rvert)$	no
⋮	⋮			⋮

A typical SNARK software system



zkSNARK applications

Blockchain Applications

Scalability:

- SNARK Rollup (zkSNARK for privacy from public)

Privacy: Private Tx on a public blockchain

- Confidential transactions
- Zcash

Compliance:

- Proving solvency in zero-knowledge
- Zero-knowledge taxes

Blockchain Applications

Scalability:

- SNARK Rollup (zkSNARK for privacy from public)

Privacy: Private Tx on a public blockchain

- Confidential transactions
- Zcash

Compliance:

- Proving solvency in zero-knowledge
- Zero-knowledge taxes

… but first: commitments

Cryptographic commitment: emulates an envelope

Many applications: e.g., a DAPP for a sealed bid auction

- Every participant **commits** to its bid,
- Once all bids are in, everyone opens their commitment

Cryptographic Commitments

Syntax: a commitment scheme is two algorithms

- **commit**(*msg*, *r*) → *com*

  secret randomness in R — commitment string

- **verify**(*msg*, *com*, *r*) → accept or reject

  anyone can verify that commitment was opened correctly

Commitments: security properties

- **binding**: Bob cannot produce two valid openings for ***com***.

  Formally: no efficient adversary can produce

  com, (m_1, r_1), (m_2, r_2)

  such that verify(m_1, ***com***, r_1) = verify(m_2, ***com***, r_2) = accept

  and $m_1 \neq m_2$.

- **hiding**: ***com*** reveals nothing about committed data

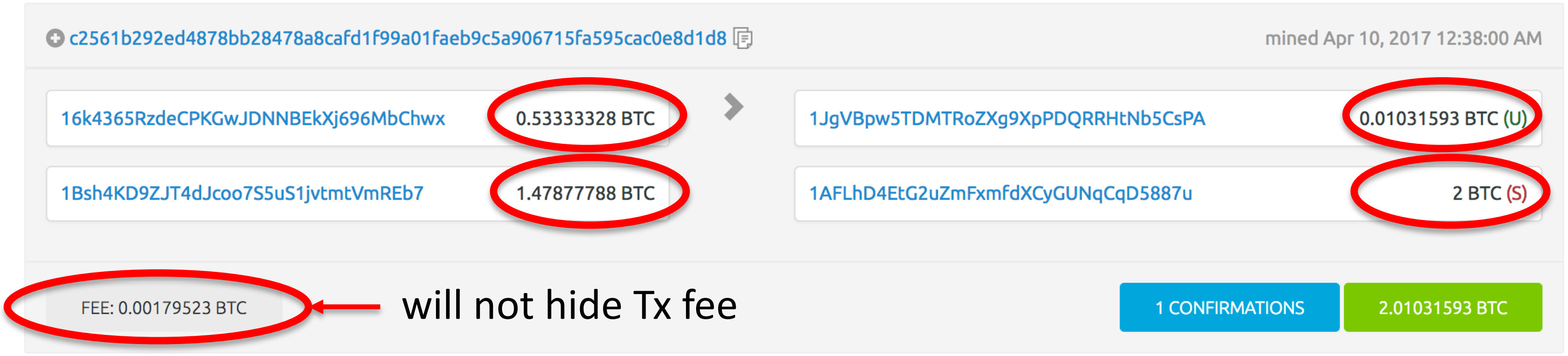
  commit(m, r) → ***com***, and r is uniform in R $(r \leftarrow R)$,

  then ***com*** is statistically independent of m

Confidential Transactions

Confidential Tx (CT)

Goal: hide amounts in Bitcoin transactions.



⇒ businesses cannot use for supply chain payments

Confidential Tx: how?

Bitcoin Tx today: Google: **30** → Alice: **1**, Google: **29**

8 bytes

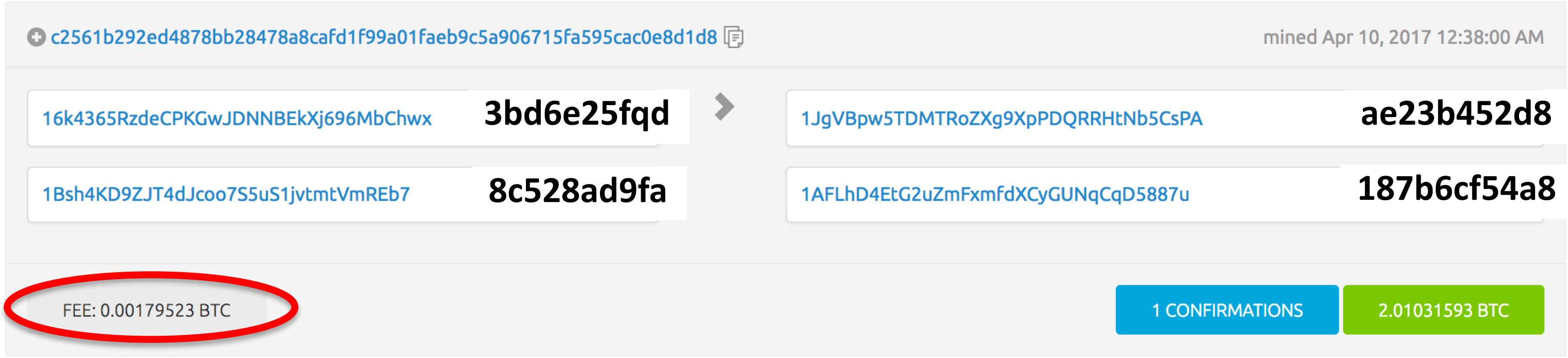
The plan: replace amounts by commitments to amounts

Google: $\mathbf{com_1}$ → Alice: $\mathbf{com_2}$, Google: $\mathbf{com_3}$

32 bytes

where $\mathbf{com_1}$ = commit(30, r_1), $\mathbf{com_2}$ = commit(1, r_2), $\mathbf{com_3}$ = commit(29, r_3)

Now blockchain hides amounts



How much was transferred ???

The problem: how will miners verify Tx?

Google: $\mathbf{com_1}$ → Alice: $\mathbf{com_2}$, Google: $\mathbf{com_3}$

$\mathbf{com_1}$ = commit(30, r_1), $\mathbf{com_2}$ = commit(1, r_2), $\mathbf{com_3}$ = commit(29, r_3)

<u>Solution: zkSNARK</u> (special purpose, optimized for this problem)

- Google: (1) privately send r_2 to Alice

  (2) construct a zkSNARK π where statement = x = ($\mathbf{com_1}$, $\mathbf{com_2}$, $\mathbf{com_3}$)

  witness = w = (m_1, r_1, m_2, r_2, m_3, r_3)

  and circuit C(x,w) outputs 0 if:

CT arithmetic circuit:

(i) $\mathbf{com_i}$ = commit(m_i, r_i) for i=1,2,3,

(ii) $m_1 = m_2 + m_3 + \text{TxFees}$,

(iii) $m_2 \geq 0$ and $m_3 \geq 0$

The problem: how will miners verify Tx?

- Google: (1) privately send r_2 to Alice

  (2) construct zkSNARK proof π that Tx is valid

  (3) append π to Tx (need short proof! $\Rightarrow$ zkSNARK)

Tx: | proof π , Google: $\mathbf{com_1}$ $\rightarrow$ Alice: $\mathbf{com_2}$, Google: $\mathbf{com_3}$ |

- Miners: accept Tx if proof π is valid (need fast verification)

  $\Rightarrow$ learn Tx is valid, but amounts are hidden

Zcash (simplified)

Zcash

Goal: fully private payments ... like cash, but across the Internet

challenge: will governments allow this ???

Zcash blockchain supports two types of TXOs:

- transparent TXO (as in Bitcoin)
- shielded (anonymized)

a Tx can have both types of inputs, both types of outputs

Addresses and TXOs

H_1, H_2, H_3: cryptographic hash functions.

sk needed to spend TXO for address pk

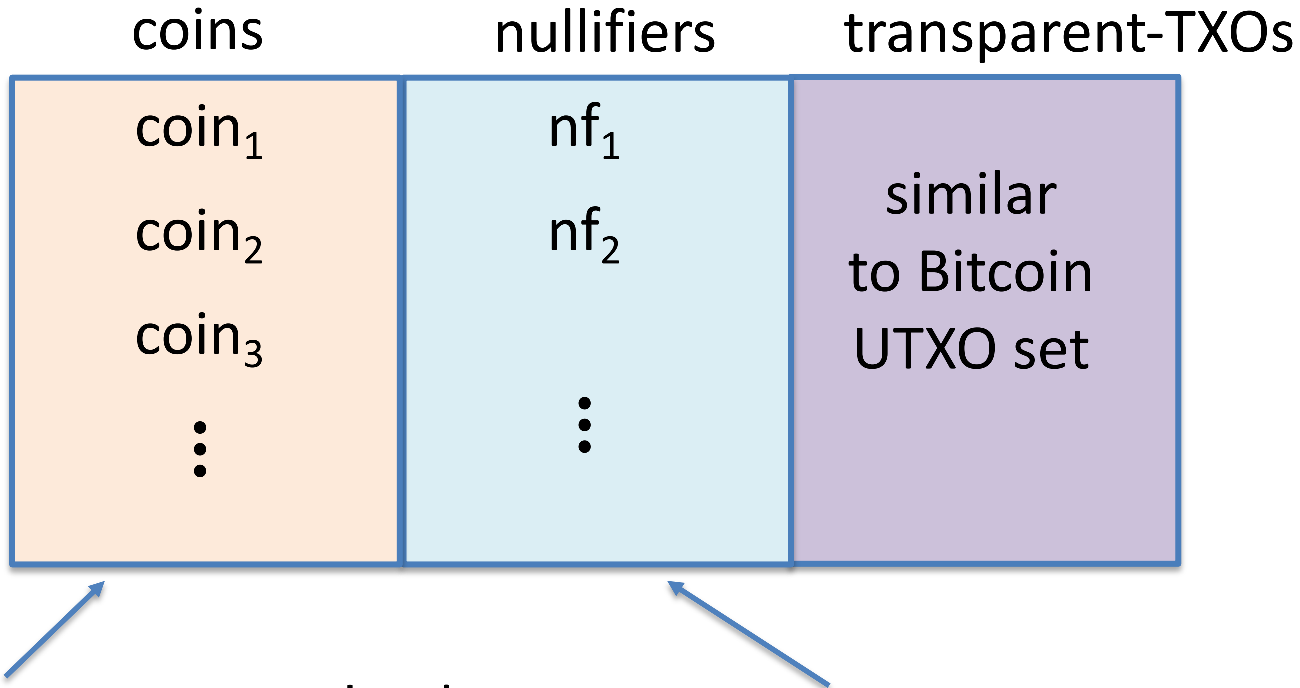
(1) shielded address: random $sk \leftarrow X$, $pk = H_1(sk)$

(2) **shielded TXO** (note) owned by address pk:

- TXO owner has (from payer): value v and $r \leftarrow R$
- on blockchain: $coin = H_2(\ (pk, v)\ ,\ r)$ (commit to pk, v)

pk: addr. of owner, v: value of coin, r: random chosen by payer

The blockchain



just Merkle root … append only tree
(coins are never removed)

explicit list:
one entry per **spent coin**

Transactions: an example

owner of **coin** = H_2((pk, v) , r) (Tx input)

wants to send **coin** funds to:

(v = v' + v'')

shielded	pk', v'	(Tx output)
transp.	pk'', v''	

---

step 1: construct new coin: **coin'** = H_2((pk', v') , r')

by choosing random r' ⟵ R (and sends v', r' to owner of pk')

step 2: compute **nullifier** for spent coin **nf** = H_3(sk, index of coin in Merkle tree)

nullifier **nf** is used to "cancel" **coin** (no double spends)

key point: miners learn that some coin was spent, but not which one!

Transactions: an example

step 3: construct a zkSNARK proof π for

statement = x = (current Merkle root, **coin'**, **nf**, v'')

witness = w = (sk, (v, r), (pk', v', r'), Merkle proof for **coin**)

C(x, w) outputs 0 if: with **coin** := H_2((pk=H_1(sk), v), r) check

The Zcash circuit

(1) Merkle proof for **coin** is valid,

(2) **coin'** = H_2((pk', v') , r')

(3) v = v' + v'' and v' ≥ 0 and v'' ≥ 0,

(4) **nf** = H_3(sk, index-of-coin-in-Merkle-tree)

from Merkle proof

What is sent to miners

step 4: send (**coin'**, **nf**, transparent-TXO, proof π) to miners,

send (v' , r') to owner of pk'

step 5: miners verify

(i) proof π and transparent-TXO

(ii) verify that **nf** is not in nullifier list (prevent double spending)

if so, add **coin'** to Merkle tree, add **nf** to nullifier list,

add transparent-TXO to UTXO set.

Summary

- Tx hides which coin was spent

  ⇒ **coin** is never removed from Merkle tree,
  but cannot be double spent thanks to nullifer

  note: prior to spending **coin**, only owner knows **nf**:

  $$\mathbf{nf} = H_3(sk, \text{index of coin in Merkle tree})$$

- Tx hides address of **coin'** owner

- Miners can verify Tx is valid, but learn nothing about Tx details.

END OF LECTURE