Recursive SNARKs

Benedikt Bünz
Recap: Non-interactive Proof Systems

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

- \(S(C) \rightarrow\) public parameters \((S_p, S_v)\) for prover and verifier
  \((S_p, S_v)\) is called a reference string

- \(P(S_p, x, w) \rightarrow\) proof \(\pi\)

- \(V(S_v, x, \pi) \rightarrow\) accept or reject
Recap: zkRollup

Today: every miner must verify every posted Tx

⇒ short proof $\pi$

verifying proof is much easier than verifying 10K Tx
Recap: zkRollup

Today: every miner must verify every posted Tx

- verify all Tx
  ⇒ short proof $\pi$

- verifying proof is much easier than verifying 10K Tx
Rollup with many coordinators

Server 1

summary1, $\pi_1$

Server 2

summary2, $\pi_2$

summary, $\pi$

verify $\pi$

verify $\pi$

verify $\pi$
Zk-zk-Rollup

- Multiple servers
- Each responsible for subset of users (no overlaps)
- Rollup aggregator (can be one of the servers)
- Rollup aggregator combines summaries (balance table) and creates one proof that
- How can we combine proofs?
- Trivial solution:
  - All servers forward all Tx
  - Rollup aggregator creates one SNARK
  - Does not save work
Recap: Non-interactive Proof Systems

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

- \(S(C) \rightarrow\) public parameters \((S_p, S_v)\) for prover and verifier
  \((S_p, S_v)\) is called a reference string

- \(P(S_p, x, w) \rightarrow\) proof \(\pi\)

- \(V(S_v, x, \pi) \rightarrow\) accept or reject
A non-interactive proof system is a triple \((S, P, V)\):

- \(S(C) \rightarrow\) public parameters \((S_p, S_v)\) for prover and verifier
  \((S_p, S_v)\) is called a reference string

- \(P(S_p, x, w) \rightarrow\) proof \(\pi\)

- \(V(S_v, x, \pi) \rightarrow\) accept or reject
Now write a circuit $C'$ that verifies $\pi$:

- Input $x'$ is $x$
- Witness $w'$ is $\pi$
- $C'(x', w') = 0$ iff $V(S_V, \pi, x) = \text{Accept}$

Finally:

$S(C') \rightarrow S'_P, S'_V$

$\pi' \leftarrow P(S'_P, x', w')$
SNARK of SNARKs

How can we aggregate proofs?

\[ S(C) \rightarrow S_P, S_V \]
\[ \pi_1 \leftarrow P(S_P, x_1, w_1) \quad \pi_2 \leftarrow P(S_P, x_2, w_2) \]

Now write a circuit \( C' \) that verifies \( \pi \):

• Input \( x' \) is \( x_1 || x_2 \)
• Witness \( w' \) is \( \pi_1 || \pi_2 \)
• \( C'(x', w') = 0 \) iff \( V(S_V, x_1, \pi_1) = \text{Accept} \) and \( V(S_V, x_2, \pi_2) = \text{Accept} \)

Finally:

\[ S(C') \rightarrow S'_P, S'_V \]
\[ \pi' \leftarrow P(S'_P, x', w') \]
Note that $C'$ depends only on $V$ and $S_v$ (not on $C_1, C_2$).

We can express $V$ as a circuit:

$0 = \text{"Accept"}$

$w' = \pi'$ is independent of $w_1, w_2$

$|C'| = 2*|V| < |2* C|$
Building SNARK of SNARKs

• How big is C’?
• Comparison $|\text{SHA256 circuit}| = 20k$ gates
• First SNARK of SNARK $\sim 1$ million gates with trusted setup (BCTV14)
• Today less than 50k gates (Halo, BCLMS20, Nova)
  • no trusted setup
• Independent of inner SNARK circuits!
Rollup with many coordinators
Zk-zk-Rollup

- Let $\text{root}_i$ be the Merkle Tree Root of summary $i$

\[
\text{root}_1 \quad \text{root}_2 \quad \text{root}_3 \quad \text{root}_4
\]

- $S_V, S_P \leftarrow S(C_R)$ \quad // $C_R$ rollup circuit

\[
C_{zk^2}(x = S_V, \text{root}; w = \text{root}_1, \text{root}_2 \ldots, \pi_1, \pi_2, \ldots):
\]

\[
V(S_V, x = \text{root}_i, \pi_i) \text{ for all } i \text{ and } \text{root} = \text{MT(root}_i \text{s})
\]
**Tornado cash**

**100 DAI pool:**
each coin = 100 DAI

Withdraw coin #3 to addr A:

nf, proof $\pi$, A
(over Tor)

100 DAI to address A

**Treasury:** 300 DAI

nullifiers

nf$_1$
nf$_2$
nf

coins

Merkle root

(next = 5)

contract state

H$_1$, H$_2$: $R \rightarrow \{0,1\}^{256}$

Merkle root

tree of height 20
(2$^{20}$ leaves)

public list of coins
... but observer does not know which are spent

nf and $\pi$ reveal nothing about which coin was spent.

But, coin #3 cannot be spent again, because $nf = H_2(k')$ is now nullified.
zk³-Rollup (tornado cash rollup)
zk³-Rollup

- Users create SNARK for TC Circuit $C_T$
  - $S_V, S_P \leftarrow S(C_T)$
  - $\pi_T \leftarrow P(S_P, tx, w)$
- Rollups create SNARKs for $C_R = \forall_i V(S_V, tx_i, \pi_i) = \text{"accept"}$
  - $tx \ root = MT(tx_1, ..., tx_n)$
  - $\pi' = \pi_{T,1} || ... || \pi_{T,n}$
  - $S_V', S_P' \leftarrow S(C_R)$
  - $\pi_R = P(S_P', tx \ root, \pi')$
- Rollup Aggregator creates SNARK for $C_A = \forall_i V(S_V', root_i, \pi_{R,i})$
  - $S_V'', S_P'' \leftarrow S(C_A)$
  - $root = MT(root_1, ..., root_k)$
  - $\pi_{R'} = \pi_{R,1} || ... || \pi_{R,k}$
  - $\pi_A = P(S_P'', root, \pi_{R'})$
Enhancing transactions with SNARKs

- We’ve seen that private transactions require zero-knowledge proofs
- Add ZK-SNARKs to every transaction
- Level 1 coordinators verify transaction by verifying transaction ZK-SNARKs
- Additionally, we can have more complicated transactions (Smart Contracts)
  - Transaction verification is constant time regardless of proof complexity
- *Can we also hide the smart contract?*
ZEXE private execution

• ZEXE is a model of computation (like UTXOs/Scripts or Accounts/EVM)
• The basic unit is a record (similar to a UTXO)
• Every transaction consumes records and creates records
• Universal predicate: Prevents double spends
• Birth predicate: Says how a record can be created
• Death predicate: Says how a record can be consumed
ZEXE private execution

Record 1:
Birth predicate 1
Death predicate 1
Payload 1

Record 2:
Birth predicate 1
Death predicate 1
Payload 1

Record 3:
Birth predicate 3
Death predicate 3
Payload 3

TX checks that Record 1 and Record 2 have not been spent
Birth3(R1, R2,R3) and Death1(R1, R2,R3) and Death2(R1,R2,R3)
ZEXE private execution

• Universal predicate (similar to tornado cash)
  • Uses nullifiers
  • Checks that nullifier=H(sk,records) is properly created
  • Checks that nullifier only appears once
  • Prevents double spends

Merkle root
tree of height 20
(2^{20} leaves)
R₁ R₂ R₃ ... 0 0 0
Implementing assets with ZEXE

- Record payload has a value $v$ and an asset id
- Birth predicate
  - Defines the token
  - New record id needs to match consumed predicate ids
  - New record value is sum of inputs
- Death predicate
  - Defines the SCRIPT
  - E.g. spendable by signature
  - E.g. Spendable by multisigature + preimage of hash
Implementing smart contracts with ZEXE

- Record payload is state of smart contract, smart contract instance id
- Birth predicate
  - Either creates smart contract or
  - One of the inputs needs to be the old smart contract record
- Death predicate
  - Defines the smart contract logic
ZEXE game of Chess

- Record payload is state of smart contract, smart contract instance id
- Birth predicate
  - Starts new game (and assigns pks to black/white) or
  - One of the inputs needs to be the old chess game
- Death predicate
  - If game finished then pay money to the winner
  - Otherwise input records must be game record + one move record
  - Move record must be signed by the right player
  - Move record must contain a valid move
Making ZEXE private

- \( S_{P_U}, S_{V_U} \leftarrow S(C_U) \) (Universal predicate)
- \( S_{P_B}, S_{V_B} \leftarrow S(C_B) \) (Birth predicate)
- \( S_{P_D}, S_{V_D} \leftarrow S(C_D) \) (Death predicate)
- \( S_{P_{TX}}, S_{V_{TX}} \leftarrow S(C_{TX}) \) (TX circuit)
- \( C_{TX} = V(S_{V_U}, \ldots) = 0 \) and \( V(S_{V_B}, \ldots) = 0 \) and \( V(S_{V_D}, \ldots) = 0 \)

And Record=\( H(\text{payload}, S_{V_B}, S_{V_D}, r) \) // \( r \) random

- TX: Input records || Output records
- Compute nullifiers \( n_{f_1}, \ldots, n_{f_n} \) from input records
- To create a TX, create three ZK-SNARKS (now ZK is important)
  - \( x = TX, w = \text{payloads}, S_{V_B}, S_{V_D} \)
  - \( \pi_U \leftarrow P(S_{P_U}, x | n_{f_1}, \ldots, n_{f_n}, w | \mid \text{MT proofs}) \)
  - \( \pi_B \leftarrow P(S_{P_B}, x, w) \)
  - \( \pi_D \leftarrow P(S_{P_D}, x, w) \)
  - Create \( \pi_{TX} \leftarrow P(S_{P_{TX}}, x, w | \pi_U, \pi_B, \pi_D) \)

Birth and death predicate as well as records are private!
What if we want to verify that computation?
SNARKs for long computations

Issues:
- \( P \) takes very long
- Starts after proving \textit{after} computation finished
- Can’t hand off computation
- \( S \) also runs at least linear in \(|C|\)
  (ok if many proofs)

C – Circuit for long computation
\( S(C) \rightarrow (S_p, S_v) \)
\( x = \text{(input, output)} \)
\( w = \text{transcript} \)

Input 

Long Computation, Transcript

Output (42)
\( P(S_p, x, w) \rightarrow \pi \)
\( V(S_v, x, \pi) \rightarrow \text{accept} \)
Handing off computation

$C_I$ — Circuit for long intermediate computation

$S(C_I) \rightarrow (S_p, S_v)$

$x_1 = (\text{input, int}_1), w_1 = \text{transcript}_1$

$x_2 = (\text{int}_1, \text{int}_2), w_2 = \text{transcript}_2$

$x_3 = (\text{int}_2, \text{output}), w_3 = \text{transcript}_3$

$P(S_p, x_i, w_i) \rightarrow \pi_i$

$V(S_v, x_1, \pi_1)$

$V(S_v, x_2, \pi_2)$

$V(S_v, x_3, \pi_3)$

$|\pi| / V$ linear in #handoffs

Input $\rightarrow$ Int$_1, \pi_1$ $\rightarrow$ Int$_2, \pi_2$ $\rightarrow$ Output (42), $\pi_3$
Incremental Proofs

• We need updatable/incremental proofs

\( C_I \) – Circuit per computation step, \( t \) number of steps/handoffs

\( S(C_I) \rightarrow (S_p, S_v) \)

\( P(S_p, x_i, w_i, \pi_{i-1}) \rightarrow \text{updated proof } \pi_i \quad // \pi_0 = \bot \)

\( V(S_v, x_0, x_t, \pi_t, t) \rightarrow \text{accept/reject} \)

\(|\pi_i| = |\pi_{i-1}| \quad // \text{proofs don’t grow} \)
PhotoProof

Allow valid updates of photo and provide proof

Viewer can still verify authenticity
PhotoProof

Proof allows valid edits only, Incrementally updated
Constant size blockchains

• Rollup reduces the verification cost
• Still linear in the number of state updates
• When a node joins the network they need to verify one rollup proof per block!
• In general starting a full node requires verification of all blocks
  • Can take days!
\( \pi_i \) proves that transactions are valid with respect to the state AND \( \pi_{i-1} \) was valid for the previous block.
Constant size Blockchain

π₁
State-MT1
TX-MT1

π₂
State-MT2
TX-MT2

π₃
State-MT3
TX-MT3

π₄
State-MT4
TX-MT4

Transactions

Merkle tree

Old miner

Head and State 4
Verifies State-MT4 and π₄

New miner
Constant size Blockchain

- Light clients can verify every block!
  - Low memory, low computation
  - Independent of length of chain or #transactions
- Relies on data serving nodes for synching
- Practical today!
Next lecture:  Crypto tricks and open discussion
Please attend last two lectures if you can