Recursive SNARKs

Benedikt Bünz
What if we want to verify that computation?
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
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
\( S(C) \rightarrow (S_p, S_v) \)
\( x = (input, output) \)
\( w = transcript \)

Input
- Long Computation, Transcript

Output (42)
\( P(S_p, x, w) \rightarrow \pi \)
\( V(S_v, x, \pi) \rightarrow accept \)
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), $\pi_3$

transcript$_1$ transcript$_2$ transcript$_3$
We need updatable/incremental proofs

$C_I$ – Circuit per computation step, $t$ number of steps/handoffs

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

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

$V(S_v, x_0, x_t, \pi_t, t) \rightarrow$ accept/reject

$|\pi_i| = |\pi_{i-1}|$  // 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
Recursive Proofs

• How can we build incremental proofs?

“Proof of a proof”: A proof that $\pi_2$ that I know a proof $\pi_1$ that $C(x_1, w_1) = 0$

“Proof of a proof of a proof ...”: A proof that $\pi_2$ that I know a proof $\pi_1$ which proves knowledge of a proof $\pi_0$ that $C(x_0, w_0) = 0$
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) = $Accept
• Note that $C'$ depends only on $V$ and $S_v$
• We can express $V$ as a circuit:

\[ S_v \times x \pi + 0 = \text{"Accept"} \]
We can also make $C'$ more complex...

- Input $x'$ is $x_0, x_1$
- Witness $w'$ is $\pi, w_1$
- $C'(x', w') = 0$ iff $V(S_V, \pi, x_0) = \text{Accept}$ AND $C(x_1, w_1) = 0$
• We can also make $C'$ more complex...
  • Input $x'$ is $x_0, x_1$
  • Witness $w'$ is $\pi, w_1, x_0$
  • $C'(x', w') = 0$ iff $V(S_V, \pi, x_0) = \text{Accept}$ AND $C(x_0, x_1, w_1) = 0$

• What about proving verification of $\pi'$?

• Needs new $C''$ that has $S'_V$ hardcoded?

• Do we need to re-run setup for every additional level of recursion?

• In some applications we can use the same $S_V$ over and over again
  • Repeated application of single function

C implements $x_1 = F(x_0)$
• Do verifiers need to re-run setup for every additional level of recursion? **Proposed solution: make $S_V$ part of the witness**

• Modify definition $C_i'$:
  • Input $x'$ is $x_0, x_1, ..., x_i$
  • Witness $w'$ is $\pi, w_i, S_V$
  • $C'(x', w') = 0$ iff $V(S_V, \pi, x_0, ..., x_{i-1}) = \text{Accept}$ AND $S(C'_{i-1}) \rightarrow (S_P, S_V)$ AND $C(x_i, w_i) = 0$

• Now $C_i'$ may accept as witness a proof $\pi$ that is a proof for $C'_{i-1}$ generated using parameters $(S_P^{i-1}, S_V^{i-1}) \leftarrow S(C'_{i-1})$. **Prover still needs to re-run setup as part of proving.**
Universal SNARK Verifier

• What goes wrong when the setup is trusted? **Making $S_V$ a witness does not work.**
  - $S_V$ ”exists” even for proofs of false statements
  - Proof system only sound when prover does not know the secrets involved in the generation of $S_V$.
  - The existence of $(S_V, \pi)$ that the algorithm $V$ would accept is meaningless.

• New solution: **universal SNARK verifier**
Universal SNARK verifier

- UC is a circuit that takes inputs $(C, x, w)$ and outputs $C(x, w)$
- $S(UC) \rightarrow (US_p, US_V)$ are parameters of SNARK system for UC
- Think of UC as an x86 processor
  - Takes in input and instructions and executes them
- Define $C_i'$:
  - Input $x'$ is $x_0, x_1, ..., x_i$
  - Witness $w'$ is $\pi, w_i, S_V$
  - $C'(x', w') = 0$ iff $V(US_V, \pi, C_{i-1}', x_0, ..., x_{i-1}) = \text{Accept AND AND } C(x_i, w_i) = 0$
Recap: SNARKRollup

Today: every miner must verify every posted Tx

- verify all Tx \Rightarrow \text{short proof } \pi
- verify \pi

- verifying proof is much easier than verifying 10K Tx
Recap: Rollup

Today: every miner must verify every posted Tx

verify all Tx \implies \text{short proof } \pi \text{ verify all Tx}

verifying proof is much easier than verifying 10K Tx

Coordinator
Rollup with many coordinators

Coordinator 1

Coordinator 2

summary1, \( \pi_2 \)

summary2, \( \pi_1 \)

summary, \( \pi \)

verify \( \pi \)

verify \( \pi \)

verify \( \pi \)
SNARK$^2$-Rollup

- Multiple coordinators/servers
- Each responsible for subset of users (no overlaps)
- Super coordinator (can be one of the coordinators)
- Super coordinator combines summaries (balance table) and creates one proof that
SNARK²-Rollup

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

$$\text{root}_1 \text{root}_2 \text{root}_3 \text{root}_4$$

- $S_V, S_P \leftarrow S(C_C) // C_C$ coordinator circuit

- $C_{SC}(x = S_V, \text{root}; w = \text{root}_1, \text{root}_2, ..., \pi_1, \pi_2, ...)$:

- $V(S_V, x = \text{root}_i, \pi_i)$ for all $i$ and $\text{root} = \text{MT}(\text{root}_i)$s
Let $\text{root}_i$ be the Merkle Tree Root of summary $i$.

$\text{root} = \text{root}_1 \text{root}_2 \text{root}_3 \text{root}_4$

$S_V, S_P \leftarrow S(C_C) // C_C$ coordinator circuit

$C_{SC}(x = S_V, \text{root}; \ w = \text{root}_1, \text{root}_2 ..., \pi_1, \pi_2, ...)$

$V(S_V, x = \text{root}_i, \pi_i)$ for all $i$ and $\text{root} = \text{MT}(\text{root}_i s)$
SNARK$^3$-Rollup

Coordinator 1

Coordinator 2

π$_{tx}$

π$_{tx}$

π$_{tx}$

π$_{tx}$

summary$_1$, π$_2$

summary$_2$, π$_1$

verify π

verify π

verify π

verify π
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
SNARK$^3$-Rollup

- Smart Contract $SC_i$
- $C_t = \text{Commit}(\text{Smart Contract State } t, r_t)$
- $C_{t+1} = \text{Commit}(\text{Smart Contract State } t+1, r_{t+1})$
- $S_{V_i}, S_{P_i} \leftarrow S(\text{ZK} - SC_i)$ // Zero-Knowledge SC
- Key-root=MT($S_{V_i} s$) // Merke tree of all verification keys
- $\text{ZK} - SC_i (x_i = C_t, C_{t+1}; w_i = \text{states } t, t + 1, r_t, r_{t+1})$:
  - $C_t, C_{t+1}$ commit to states $t, t + 1$ and transition is valid
SNARK$^3$-Rollup

- Each user creates $\pi_i \leftarrow P(S_{P_i}, x_i, w_i) (ZK)$ and outputs
  - $C_{t+1,i}, \pi_i$
- Level 1 coordinator takes many outputs and produces one proof using Key-root to select the correct verification key
- The coordinator does not care about which key is used just that it’s the correct one
- Possible to hide state transition AND smart contract details
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!
\[ \pi_i \text{ proves that transactions are valid with respect to the state AND } \pi_{i-1} \text{ was valid for the previous block} \]
Constant size Blockchain

\[ \pi_1 \]
State-MT1
TX-MT1

\[ \pi_2 \]
State-MT2
TX-MT2

\[ \pi_3 \]
State-MT3
TX-MT3

\[ \pi_4 \]
State-MT4
TX-MT4

Merkle tree

Transactions

Old miner

Head and State 4
Verifies State-MT4
and \( \pi_4 \)

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