Verifying Computations Faster than Re-Executing them

SNARKs, STARKs, IOPPs and other swear words

Christian Reitwiessner

Ethereum Meetup Berlin
2018-01-30

chris@ethereum.org
@chriseth / @ethchris

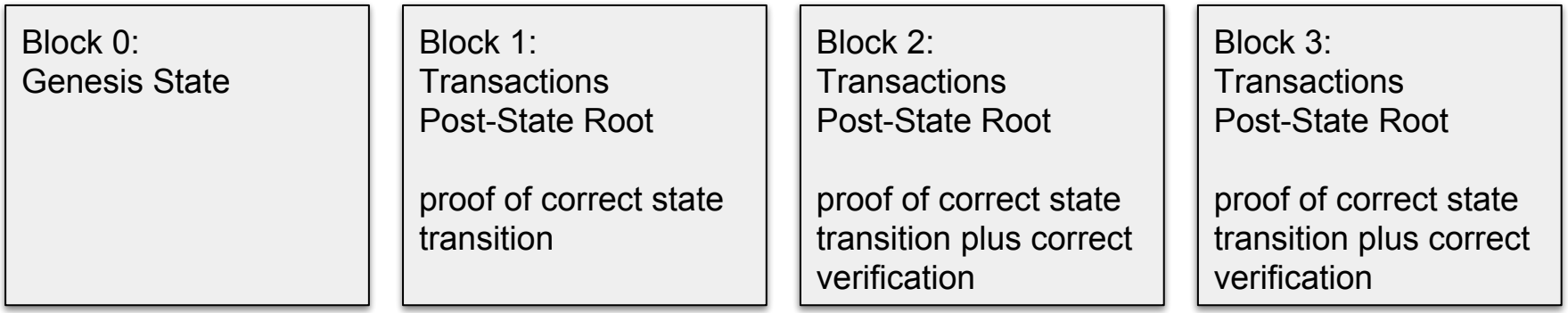
Outline

- Applications for Blockchains
- General Approach - SNARKs / STARKs
- STARKs: Computation Trace + Polynomial Encoding
- IOPPs for Low-Degree Testing
- Adding Zero Knowledge

Results by
Eli Ben-Sasson, Iddo Bentov, Alessandro Chiesa, Michael A. Forbes, Ariel Gabizon, Ynon Horesh, Michael Riabzev, Nicholas Spooner, Madars Virza and others.

Warning: Some details are left out here and there!

Verifying Blockchains



Verifying the proof of a single block recursively verifies the validity of the full chain
Syncing means downloading the state and checking a single proof
(does not yet work in practice)

General Approach

Interactive Protocol: Prover and Verifier exchange messages
Verifier should run fast (miner) and can use randomness
On correct input, Verifier has to accept. On incorrect input, Verifier has to reject with high probability.

Interactive Protocols can often be made non-interactive using Fiat-Shamir

Method	**proof size**	**run-once**	**transparent**	**crypto assumptions**
SNARKs	short	no	no	yes
STARKs	longer	yes	yes	only hash functions

General Approach (2)

Encode computation as statements about equality of polynomials.

Check polynomials on random set of points.
(Fundamental theorem of algebra: If $p \neq q$, then $p(x) \neq q(x)$ for almost all x)

Problem: Avoid transferring full poly, but still evaluate correctly.

SNARKs use partially homomorphic encryption: Prover evaluates poly at unknown / encrypted point generated during trusted setup and only sends value.

In STARKs, prover commits to the full evaluation of the poly using a Merkle tree (sending only the root hash), Verifier requests random points.
Requires additional proof that prover used a poly of small degree (IOPP).

Arithmetrization of Computation Traces

Step	A_1	A_2	A_3
0	1	4	2
1	1	4	6
2	10	4	6
3	40	4	6
...			
7999	12	3	34
8000	15	3	34

Input: 1, 4, 2
Claimed output: 15, 3, 34

Computation is correct if every single step is correct.

Prover interprets registers as functions of the step and encodes them as degree-8000 polys $A_1(x)$, $A_2(x)$, $A_3(x)$.

C is pre-specified poly that checks single steps.

Computation is correct $\Leftrightarrow$ input and output is correct and $C(x, A_1(x), A_2(x), A_3(x), A_1(x+1), A_2(x+1), A_3(x+1)) = 0$ for all $x \in \{0, \ldots, 7999\}$

Arithmetrization of Computation Traces (2)

Computation is correct ⇔ input and output is correct and
$C(x, A_1(x), A_2(x), A_3(x), A_1(x+1), A_2(x+1), A_3(x+1)) = 0$ for all $x \in \{0, \ldots, 7999\}$

Combine A_i into single polynomial A: Stretch steps using $a_i(x) = 3x + i$:
$C(x, A(a_1(x)), A(a_2(x)), A(a_3(x)), A(a_1(x+1)), A(a_2(x+1)), A(a_3(x+1))) = 0$
for all $x \in \{0, \ldots, 7999\}$

Turn statement about discrete points in poly into statement about poly identity:

Computation is correct ⇔ input and output is correct and there is poly D such that
$C(x, A(a_1(x)), \ldots, A(a_3(x+1))) = Z(x)\, D(x)$ for all x (in the whole field)
where $Z(x) = (x-0)(x-1)(x-2)...(x-7999)$

Note that all polys have “small” degree (tiny in comparison to the field size)!

Protocol

Reminder: $C(x, A(a_1(x)), \ldots, A(a_3(x+1))) = Z(x)\,D(x)$

Given: $C(...)$, $a_1(x)$, $a_2(x)$, $a_3(x)$, $Z(x)$

Prover: Computes A and D (poly division)
Sends root of Merkle tree for A and D to Verifier (on e.g. 1 000 000 points)

Verifier: Requests $A(a_i(x'))$, $A(a_i(x'+1))$, $D(x')$ for random x' from Prover

Prover: Provides $A(a_i(x'))$, $A(a_i(x'+1))$, $D(x')$ together with Merkle proof

Verifier: Checks $C(x, A(a_1(x')), \ldots, A(a_3(x'+1))) = Z(x')\,D(x')$

What is missing?

Missing Piece: IOPP (simplified)

Prover can cheat unless A and D are polys of “low” degree (8000)!

Use IOPP - Interactive Oracle Proof of Proximity

Task: Given Merkle tree of values of f(x), prove that f is a poly of degree at most d.

Main idea: Divide and Conquer - Split into odd and even powers:

$f(x) = g(x^2) + x\, h(x^2)$, for some polys g and h with degree at most d/2.
Note: domain size is halved, too (only squares)

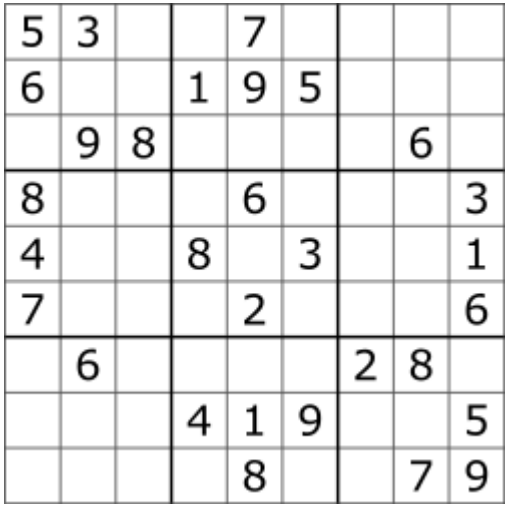
Prover commits to Merkle tree of g and h, Verifier requests random check

Both continue recursively to check $\deg(g), \deg(h) \leq d/2$

How to add Zero Knowledge (simplified)

Zero Knowledge: Verifier is convinced that computation is correct but does not learn anything else.

Example: Sudoku



How to add Zero Knowledge (simplified) (2)

For the IOPP, Prover has to show that A and D are arbitrary low-degree polys.

Let P be the set of polys of degree at most d

Lemma: For any $u \in P$ and any function f: $f \in P \Leftrightarrow (f + u) \in P$

Prover chooses random $u \in P$ and performs the IOPP for $A' = A + u$.

If A' is in P, then it is just any random element of P, so Verifier learns nothing about A.

(Prover has to evaluate A once to show that $A' = A + u$, but details show this is not a problem for the STARK itself)

More Details

Reducing computations through polynomials to low-degree testing:
Ben-Sasson, Sudan: Short PCPs with Polylog Query Complexity

ZK-IOPP Construction:
Ben-Sasson, Chiesa, Gabizon, Virza: Quasilinear-Size Zero Knowledge from Linear-Algebraic PCPs

Putting it all together and optimizing some constants:
Ben-Sasson, Bentov, Horesh, Riabzev: Scalable, transparent, and post-quantum secure computational integrity