STARKS

Christian Reitwießner
chris@ethereum.org

Ingredients for STARKs:

Encode the computational task into a polynomial problem.

IOP: Prover and verifier exchange messages. In many cases, prover sends gigantic message, but verifier only reads a tiny bit. In such a setting, prover can merkelize, only send root hash and verifier can request parts of the message plus Merkle proof.

(Probabilistically Checkable) Proof of Proximity: Ensures that the polynomial used by the prover is "close" to a low-degree polynomial.

General proof of proximity: Accept if input is in the set, reject with high probability if it has a certain distance or more to the set.

From "Computational integrity with a public random string from quasi-linear PCPs.":

Encode the computation trace as a sequence of N elements from $\mathcal{F}_{2^64}$ and encode this in a polynomial a of degree $N-1$ over $\mathcal{F}_{2^64}$.

ACSP (algebraic constraint satisfaction problem) - NTIME(n)-complete. Instances: $(\mathcal{F}, \{AFF_1, \ldots, AFF_k\}, H, C)$, where $\mathcal{F}$ finite field, AFF_i is affine map on $\mathcal{F}$, i.e. $ax + b$, $a, b \in \mathcal{F}$, $H \subseteq \mathcal{F}$, $C\colon \mathcal{F}^k \to \mathcal{F}$ is a poly of degree at most $|H|$ in its first variable. A univariate poly A is said to satisfy the instance iff $\deg(A) \leq |H| - 1$ and for all $x \in H$: $C(x, A(AFF_1(x)), \ldots, A(AFF_{k'}(x))) = 0$.

Compute $b = \Phi(a)$, where Φ is an ACSP instance and append PCPPs π_a, π_b for both a and b.

Verifier: Input is Φ, oracle access to a, b and π_a, π_b. Run RS-PCPP on (a, π_a) and (b, π_b) and then use sampling to check that the two polynomials a and $\Phi(b)$ are equal.

Interesting: This sampling could be done by outside actors. If an error is found, it is reported.

Reed-solomon-Codes: Univariate polynomial of high degree.

PCPPs only have oracle access to the statement and proof and check that the proof is "close" to a true statement. In contrast, PCPs have full access to the statement and oracle access to the proof and check that the statement is true.

Low-degree test: Procudure to check if a function we have oracle access to (i.e. can ask to be evaluated) is a multivariate polynomial of "low" degree.

Zero-tester: efficient procedure to verify if a function given by an oracle is close to a multivariate polynomial that is zero on every point in a prespecified subset of its domain

New approach in BS08: - reduce SAT to zero-testing. Reduce zero-testing to low-degree testing.

In BS08, all oracle queries are non-adaptive

Definition 0.1. $\mathrm{RS}[\mathbb{F}, S, \rho] = \{f \colon S \to \mathbb{F} \mid$ *there is a poly* p *of degree at most* $\rho\|S\|$ *such that* $p(x) = f(x)$ *on* $S\}$

Definition 0.2. *IOP - interactive oracle proof: IP for a nondeterministic language, where the prover's input includes the witness, the verifier has oracle access to the prover's messages. Proof length* $\ell(x)$ *includes the messages, query complexity* $\mathrm{q}(x)$ *only the number of queries. Perfect completeness and* ε*-soundness. Prover complexity includes the witness size as input, verifier complexity only the input.*

Definition 0.3. *STIK - scalable transparent IOP of Knowledge - soundness against unbounded provers. Language* L*, witness relation* R

- *transparent: all verifier messages and queries are public random coins*
- *scalable: verifier complexity is poly in* n*, polylog in the ntime-complexity of* L *and polylog in* $\frac{1}{\varepsilon}$*. Prover complexity is linear in the ntime-complexity of* L *times the bounds for the verifier*
- *proof of knowledge: there exists a knowledge error function* $\varepsilon'(n)$ *and a randomized extractor* E *that, given oracle access to any prover* $\mathrm{P}*$ *that causes the verifier to accept* x *with probability at least* $\varepsilon'(|x|)$*, outputs in expected time poly of something a witness* w *such that* $(x, w) \in R$*.*
- *privacy preservation: there exists a randomized simulator that samples perfectly the distribution on transcripts of interactions between verifier and prover and runs in time poly* $T(n)$*.*

Complexity in the following is arithmetic complexity.

STARKS do not reduce via arithmetic circuits because those would result in quadratic polynomials. The STARK polynomials have a degree of up to 8.

STIK is based on the IOP model, which does not require any cryptographic assumptions. Actual realizations, though, is unlikely to be possible without cryptographic assumptions.

If you restrict the computational power of the prover, you end up with an argument system, instead of a proof system -¿ STARK.

Using collision resistant hash functions, you can convert a STIK into an interactive STARK, it turns perfect zero knowledge into computational zero knowledge. (Kilian, note on efficient zero-knowledge proofs and arguments. - notarized envelopes, PCP and hash functions)

Micali and Valiant: Turn it into a non-interactive argument system. Micali: Computationally sound proofs., Valiant: Incrementally verifiable computation or proofs of knowledge imply time/space efficiency.

The first STARK was in "'Computational integrity with a public random string from quasi-linear PCPs."' (eurocrypt)

0.1 from AIR to ZK-Stark

AIR: Algebraic Intermediate representation

Computational integrity: Input is state transition relation plus boundary conditions (input and output). Witness is execution trace (sequence of states)

Question: Why machine instead of circuit? Circuit could be more efficient. - turns out reduction uses higher degree polynomials, circuits would result in quadratic only.

turn this into an AIR: elements of the execution trace are turned into sequences of elements of a finite field, and transition relation is specified by a set of polynomials

AIR is turned into an affine graph representation (nodes are states, edges are transitions). -¿ APR - algebraic placement routing

use a 1-round IOP to produce a pair of instances of the RS proximity testing (RPT) problem. Verifier uses randomness to link the numerous constraints into a single one. -¿ algebraic linking iop (ALI)

FRI as final step

Definition 0.4. *AIR - looks like PUZZLE Instances: field, width w, height T, '"border'" (set of coordinates plus value), constraints: set of polys in $2 \cdot w$ variables (they have to be zero)*

The constraints match one row to the next. Witness is filling of the table.

We seem to get zero knowledge since the prover can pick a random low-degree polynomial and add it to an existing low-degree polynomial.

Definition of zero knowledge: For every verifier there exists a simulator that can generated the messages betwene prover and verifier without having access to the witness. More specifcially: It is not possible to distinguish between the simulator and the actual protocol. This works since most of the messages from prover to verifier are just random numbers.

Computational zero knowledge: Only efficient deciders cannot distinguish. (if one-way functions exist, CZK-IP)

statistical zero knowlede (indistinguisable to anyone)

perfect zero knowledge: identical distributions

PCP: Interactive proof where the prover is non-adaptive (the proof string is fixed)

From Quasi-Linear size Zero-Knowledge from Linear Algebraic PCPs

duplex PCP: Prover sends oracle, verifier (without querying the oracle) sends message, prover sends second oracle, verifier queries the oracles and accepts / rejects

Ideal zk-construction: There is a t-wise independent set of witnesses. Prover samples from it and permutes such that any k bits in the permutation jointly only depend on t bits of the original witness.

Since the verifier can only query k bits in the proof, and the witness set is t-wise independent, the verifier only learns t random bits from a uniform distribution.

A code is t-wise independent, if for any index set of cardinality t, reading at those t bits from a random codeword is the uniform distribution on Σ^t.

A linear code is t-wise independent if and only if its dual code has distance at least t.

Informally: For any f, if for a random low-degree poly u and a random number α, $\alpha f + u$ is a low-degree poly, then also f is with high probability.

Prover samples w' from the t-wise independent witness set, picks random u, sends (w', u), then verifier sends random α, prover sends $v := \alpha w' + u$ plus a proof of proximity of low-degreeness for that polynomial. Verfier checks proof and does random spot-checks that $v = \alpha w' + u$.

This means the PCPP is run on the random permutation and allows (and requires) many queries. The verifier only checks with a tiny number of queries that v actually is this random permutation of w', and only queries w' in this part of the protocol. v is in essence just a random element in the set of low-degree polynomials.

Since the prover first commits to u, and the verifier is still able to multiply w' by α, the verifier will not succeed in reducing the degree of w'.

Question: How does this t-independent witness set look like and how exactly does the probability behind this work? Question: How can the verifier be confident that the spot checks are enough?

Zero-knowledge in general: Given a certain input, it is hard to find a solution, but once you have a solution, you can modify the input and the solution and still have a valid input-solution-pair. Rough idea how to get ZK: Prover only sends a random permutation to the verifier, but also proves that it is a random permutation of a solution corresponding to the concrete input. The permuted solution is checked in detail. The verifier basically only learns a single element of the set of all possible input-solution-pairs.

Why is checking low-degreeness the central point?

With AIR we have to show that a list of points can be extended to a common zero of certain polynomials.

How does FRI work?

starks are probably post-quantum secure, they only require randomness and collision-resistant hash functions.

public randomness!

snark proof size: 288 bytes, stark proof size: few hundred kilobytes

General reduction: Have computation over t steps, constraints over all these steps. Whan to prove for all t such that $C(P(x)) = 0$. Transform this restricted statement to universal statement: $C(P(X)) = Z(x) * D(x)$, where Z is the poly that is zero on exactly $1, 2, \ldots, n$.

Verifier can randomly sample to check equality of polynomials. Prover has to show that these are actually polynomials (otherwise, it is easy). Verifier knows C completely.

We need low-degree-extension (i.e. prover has to evaluate polys on more than just $1, 2, \ldots, t$), otherwise everything is a low-degree poly. This also affects soundness error!

Take table of computation steps. State consists of k registers. $H = \{1, \ldots, t\}$. For all steps, the transition has to be correct. The prover provides the states at each step in the form of k polynomials A_i of degree $|H|$. C is the checking polynomial, checks that transition is correct given the pre and the post-state: $C(x, A_1(x), \ldots, A_k(x)) = 0$ for all $x \in H$.

We stretch H to $H' = \{1, \ldots, kt\}$ and combine the A_i into a single poly using $a_i(x) = kx + i$: $C(x, A(a_1(x)), \ldots, A(a_k(x)) = 0$ for all $x \in H'$ ($H' = \{1, \ldots, kt\}$).

Note that $x \in H'$ is really important here - we have $|H'|$ many discrete checks. Turn this into a statement about polynomials:

There is a poly D such that $C(x, A(a_1(x)), \ldots, A(a_k(x)) = Z_{H'}(x) \cdot D(x)$ for all x, where $Z_{H'}(x) = \pi_{h \in H'}(x - h)$ is zero on all elements of H'

Note that in all polynomials here we have a certain upper bound on the degree that is much smaller than the size of the field.

The prover comptues D using polynomial division and then evaluates A and D on all elements of the field, or at least on a much larger set than H', creates merkle trees out of them and sends them to the verifier.

The verifier requests random positions x' from the prover, the prover provides the values $A(a_1(x')), \ldots, A(a_k(x'))$ and $D(x')$. Verifier computes $C(x', A(a_1(x')), \ldots, A(a_k(x'))$ and $Z_{H'}(x') \cdot D(x')$ and checks for equality.

Since all functions are polynomials, it is really hard to first commit to the function and then fake the equality, because polynomials (of relatively low degree) are almost never equal. So the only missing point is to convince the verifier that the Merkle tree the prover committed to is actually the evaluation of a polynomial of the right degree.

IOPP: Interactive oracle proof of proximity.

Task: Given evaluation of function $f(x)$, prove that f is represented by a poly of degree $\leq d$.

Main idea: We do not check that f is poly, we only check that it agrees with a poly at an overwhelming number of points (proximity - it is close to a poly).

Turn poly f into polys g, h such that $f(x) = g(x) \cdot h(x)$ and g and h have degree at most $\sqrt{d}$ - basically split the powers: $f(x) = g(x^2) + x \cdot h(x^2)$.

Note that this also recudes the domain size in half because we only have x^2.

Again, verifier checks equality at some random points and then applies the degree-check recursively.

If the degree is one, verifier can check itself.

How to get zero knowledge:

zero knowledge in degree testing:

Informally: For any f, if for a random low-degree poly u and a random number α, $\alpha f + u$ is a low-degree poly, then also f is with high probability.

Prover samples w' from the t-wise independent witness set, picks random u, sends (w', u), then verifier sends random α, prover sends $v := \alpha w' + u$ plus a proof of proximity of low-degreeness for that polynomial. Verfier checks proof and does random spot-checks that $v = \alpha w' + u$.

This means the PCPP is run on the random permutation and allows (and requires) many queries. The verifier only checks with a tiny number of queries that v actually is this random permutation of w', and only queries w' in this part of the protocol. v is in essence just a random element in the set of low-degree polynomials.

Since the prover first commits to u, and the verifier is still able to multiply w' by α, the verifier will not succeed in reducing the degree of w'.

Talk outline:

not my results - authors

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

verifying computations faster than re-executing them

useful for blockchains: proof that block 1 is valid, proof that block 2 is valid (including validity of proof for 1) -¿ all proofs have same size, checking validity of full blockchain is the same as checking single block. (this does not yet work in practice)

Several approaches: SNARKs, extremely short proofs, well-suited for checking the same thing mulitple times, trusted setup, cryptographic assumptions

STARKs: much larger proofs, also efficient for one-shot-verification, no trusted setup, no assumptions apart from hash functions (quantum secure)

In both models: prover / verifier. Usually interactive, but can be made non-interactive. (Fiat-Shamir Heuristic)

Formulate correctness of computation in terms of equality of certain polynomials. Checking the proof is checking equality on a random set of points. If two polys are not equal, they are not equal on many points (fundamental theorem of algebra).

SNARKs use partially homomorphic encryption - prover evaluates the polynomials on an unknown/encrypted point. This point is generated during the trusted setup.

In STARKs, prover commits to the full evaluation of the poly using a Merkle tree (sending only the root hash), Verifier requests random points. This requires an additional proof that this Merkle tree is the Merkle tree of a polynomial of a certain max degree (it does not matter which). -¿ IOPP.

More in depth:

Computation trace

encoding into polynomial

checking

low-degree testing

how to make zero knowledge

how to make non-interactive: All queries by the verifier where just random numbers. Either use some randomness beacon or just use the hash of the previous segment of the proof as source of randomness. This can be checked offline.