CS 5594: BLOCKCHAIN TECHNOLOGIES

Spring 2021

THANG HOANG, PhD

(ZERO-KNOWLEDGE) VERIFIABLE COMPUTATION

Most slides derived from the one by Dmitry Khovratovich

Overview

Motivation

zk-STARK

Zk-SNARK

Bulletproofs

MOTIVATION

Verifiable Computation

Sometimes we need to delegate computation to remote agents whom we do not fully trust:

- Database is searched or updated on a remote server;
- Secure hardware signs the input.
- Privacy-preserving AI training;
- Blind auctions, **blockchain**, etc..

We might need to pay the agents for the work if it is done correctly.

Summary

Alice needs program C to be computed on input X;

Bob takes the task (C,X);

Bob returns answer A and proof of correctness P;

Alice verifies P spending much less time than Bob.

Alice rewards Bob.

How to do that so that Bob can not cheat?

Summary

Alice needs program C to be computed on input X;

Bob takes the task (C,X);

Bob returns answer A and proof of correctness P;

Alice verifies P spending much less time than Bob.

Alice rewards Bob.

How to do that so that Bob can not cheat?

A mistake in just one step can ruin the entire computation.

zk-STARK

Simple Example

Program:

Take input $X_0 = X$;

Compute $X_i \leftarrow (X_{i-1}^2 + 3)$ up to $i = 100$.

Return $A = X_{100}$.

No big number arithmetic, only lowest 10 digits (modulo 10^{10}).

Simple Example

Program:

Take input $X_0 = X$;

Compute $X_i \leftarrow (X_{i-1}^2 + 3)$ up to $i = 100$.

Return $A = X_{100}$.

No big number arithmetic, only lowest 10 digits (modulo 10^{10}).

Alice says X = 1.

Bob returns A = 5251434499 and some proof P (just a few bytes).

How can that be?

Protocol

Program:

Take input $X_0 = X$;

Compute $X_i \leftarrow (X_{i-1}^2 + 3)$ up to $i = 100$.

Return $A = X_{100}$.

No big number arithmetic, only lowest 10 digits (modulo 10^{10}).

Very simple protocol:

Bob computes some function f on 10000 inputs, from 1 to 10000.

Bob computes another function g on the same 10000 inputs.

Alice selects random 0 < s < 10000.

Bob returns f(s),f(s +1),g(s).

Alice verifies just <u>one equation</u> and any cheat is detected with probability 99%.

Protocol

Program:

Take input $X_0 = X$;

Compute $X_i \leftarrow (X_{i-1}^2 + 3)$ up to $i = 100$.

Return $A = X_{100}$.

No big number arithmetic, only lowest 10 digits (modulo 10^{10}).

Very simple protocol:

Bob computes some function f on 10000 inputs, from 1 to 10000.

Bob computes another function g on the same 10000 inputs.

How exactly?

Alice selects random 0 < s < 10000.

Bob returns f(s),f(s +1),g(s).

Alice verifies just <u>one equation</u> and any cheat is detected with probability 99%.

Details

Code	Value	f
X_0	1	$f(0)$
X_1	4	$f(1)$
X_2	19	$f(2)$
X_3	364	$f(3)$
…		
X_{100}	5251434499	$f(100)$

Let Bob's program be a table of 101 entries

- Compute polynomial f of degree 100 that interpolates on the memory

Details

Code	Value	f
X_0	1	$f(0)$
X_1	4	$f(1)$
X_2	19	$f(2)$
X_3	364	$f(3)$
…		
X_{100}	5251434499	$f(100)$

Let Bob's program be a table of 101 entries

- Compute polynomial f of degree 100 that interpolates on the memory
- Define **constraint**
  $$C(x, y) = y - x^2 - 3.$$
- Bob executed the program if
  $$C(f(x), f(x+1)) = 0 \textit{ for all } x$$
- Note that $C(f(x), f(x+1))$ has degree 200, and $D(x) = x(x-1)(x-2) \cdot (x-99)$ divides it.
- Define
  $$g(x) = C(f(x), f(x+1))/D(x)$$

Details

Code	Value	f
X_0	1	$f(0)$
X_1	4	$f(1)$
X_2	19	$f(2)$
X_3	364	$f(3)$
…		
X_{100}	5251434499	$f(100)$
…	…	…
	?	f(10000)

$$C(x, y) = y - x^2 - 3.$$

$$D(x) = x(x-1)(x-2) \cdot (x-99)$$

$$g(x) = C(f(x), f(x+1))/D(x)$$

Details

Code	Value	f
X_0	1	$f(0)$
X_1	4	$f(1)$
X_2	19	$f(2)$
X_3	364	$f(3)$
…		
X_{100}	5251434499	$f(100)$
…	…	…
	?	f(10000)

$C(x, y) = y - x^2 - 3.$

$D(x) = x(x - 1)(x - 2) \cdot (x - 99)$

$g(x) = C(f(x), f(x + 1))/D(x)$

Bob goes on

- Compute f and g up to 10000

Details

Code	Value	f
X_0	1	$f(0)$
X_1	4	$f(1)$
X_2	19	$f(2)$
X_3	364	$f(3)$
…		
X_{100}	5251434499	$f(100)$
…	…	…
	?	f(10000)

$C(x, y) = y - x^2 - 3.$

$D(x) = x(x - 1)(x - 2) \cdot (x - 99)$

$g(x) = C(f(x), f(x + 1))/D(x)$

Bob goes on

- Compute f and g up to 10000
- Commit to the evaluations:
  $H_1 = H(f(0), f(1), \ldots, f(10000));$
  $H_2 = H(g(0), g(1), \ldots, g(10000));$
- Send H_1, H_2 to Alice with proofs that f, g of degree 100.
- Alice sends random s between 0 and 10000 to Bob.
- Bob sends back $f(s), f(s + 1), g(s)$.

Details

Recall

$$C(x, y) = y - x^2 - 3.$$

$$D(x) = x(x-1)(x-2) \cdot (x-99)$$

$$g(x) = C(f(x), f(x+1))/D(x)$$

Alice verifies

$$C(f(s), f(s+1))/D(s) = g(s).$$

It works if Bob is honest by definition.

Cheat

Code	Value	f
X_0	1	$f(0)$
X_1	4	$f(1)$
X_2	20	$f'(2) \neq f(2)$
X_3	365	$f'(3)$
…		
X_{100}	5251434499	$f(100)$
…	…	…
	?	f(10000)

What if Bob cheats and does not know the true f?

- He cannot compute proper $g = C(f, f)/D$ of degree 100
- $C(f', f')/D$ will differ from g on at least 1 point
- As polynomials they can agree on at most 100 points (they have degree 100) out of 10000.
- Thus for random s Alice detects the cheat with probability 99%

Cheat

What if Bob cheats and does not know the true f?

Code	Value	f
X_0	1	f(0)
X_1	4	f(1)
X_2	20	$f'(2) \neq f(2)$
X_3	365	$f'(3)$
…		
X_{100}	5251434499	f(100)
…	…	…
	?	f(10000)

- He cannot compute proper $g = C(f, f)/D$ of degree 100
- $C(f', f')/D$ will differ from g on at least 1 point
- As polynomials they can agree on at most 100 points (they have degree 100) out of 10000.
- Thus for random s Alice detects the cheat with probability 99%

Extensions

Zero knowledge: Bob can convince Alice revealing only X_i , $i > 100$.

Complex programs

Arbitrary Programs

Let C be a code of T steps. I can prove that

I executed the code on (secret) input K and got result X.

Let C_P be the code of my CPU (handling registers, function calls, memory, etc.).

Prepare T CPU-state variables, $\mathbf{S} = (S_1, S_2, \ldots, S_T)$.

Using T copies of C_P, prove correct transitions.

Let $\mathbf{W} = (W_1, W_2, \ldots, W_T)$ be the list of states S sorted by the memory address they access.

- Prove that successive memory accesses yield the same data.
- Prove that $\mathbf{W}$ is a sort of $\mathbf{S}$ using permutation networks/proof of shuffle, etc.

zk-SNARK

Pairings

Group G with generator g, for example a set of integers modulo a prime p

Pairing e is a function of two arguments such that

$$e(g^a, g^b) = e(g, g)^{ab}$$

and $e(g, g)$ is also a generator

Factorization Proof

Suppose you want to prove you know p and q

$$N = p \cdot q.$$

Then you provide $p' = g^p, q' = g^q$ and everyone can verify that

$$e(p', q') = e(g, g)^N$$

since

$$e(p', q') = e(g^p, g^q)$$

Sophisticated Programs

a_1, a_2 – inputs, a_n – output.

$$a_3 \leftarrow a_1 \cdot a_2;$$
$$a_4 \leftarrow a_2 \cdot a_3;$$
$$a_5 \leftarrow a_1 \cdot (a_4 + a_2);$$
$$\ldots$$

Quite many real programs can be represented this way.

Suppose I have a correct program execution: $(a_1, a_2, a_3, \ldots)$. How to prove it is correct?

- Selecting a random equation? Then it will be easy to cheat in the others
- Supply all a^i as g^{a_i} ? Too expensive.

Sophisticated Programs

Program with n lines

$$a_3 \leftarrow a_1 \cdot a_2;$$
$$a_4 \leftarrow a_2 \cdot a_3;$$
$$a_5 \leftarrow a_1 \cdot (a_4 + a_2);$$
$$\ldots$$

Instead, try the following concept:

Trusted party squeezes the entire program into n polynomials $\{u_i, v_i, w_i\}$ of degree n which encodes which a_i gets into which equation with which coefficient so that $\{a_i\}$ is the program execution only if

$$\underbrace{\left(\sum_i a_i u_i(X)\right)}_{A} \cdot \underbrace{\left(\sum_i a_i v_i(X)\right)}_{B} = \underbrace{\left(\sum_i a_i w_i(X)\right)}_{C} + d(X)$$

Sophisticated Programs

Trusted party squeezes the entire program into n polynomials $\{u_i, v_i, w_i\}$ of degree n which encodes which a_i gets into which equation with which coefficient so that $\{a_i\}$ is the program execution only if

$$\underbrace{\left(\sum_i a_i u_i(X)\right)}_{A} \cdot \underbrace{\left(\sum_i a_i v_i(X)\right)}_{B} = \underbrace{\left(\sum_i a_i w_i(X)\right)}_{C} + d(X)$$

Then compute the polynomial on a secret input s and stores (exponentiated) all $g^{u_i(s)}$ and $g^{d(s)}$. This is called a proving key P.

Prover runs the program on his own input and computes the internal variables a_i. They should satisfy program equations. Then Prover computes g^A, g^B, g^C as a short proof π.

Verifier checks the proof in constant time by computing a few pairings to verify the equation above.

Form Single Equation From Many

For $x = 0, x \neq 1,2$ $\quad a_3 = a_1 \cdot a_2$

For $x = 1, x \neq 0,2$ $\quad a_4 = a_2 \cdot a_3$

For $x = 2, x \neq 0,1$ $\quad a_5 = a_1 \cdot (a_4 + a_2)$

Proper multiplication:

$$a_3(x-1)(x-2)/2 = \big((x-1)(x-2)/2\big)a_1 \cdot \big((x-1)(x-2)/2\big)a_2$$

$$-a_4 x(x-2)/2 = (x(x-2)/2)a_2 \cdot (x(x-2)/2)a_3$$

$$x(x-1)a_5 = x(x-1)a_1 \cdot (x(x-1)a_4 + x(x-1)a_2)$$

Altogether

$$a_1 a_2(x^2 - 3x + 2) + a_2 a_3(x^2 - 2x) + \ldots = 0$$

Polynomial Relation for the Entire Scheme

$(a_1, a_2, \dots, a_n)$ are scheme execution if and only if the following polynomials are equal

$$\left(\sum_i a_i u_i(X)\right) \cdot \left(\sum_i a_i v_i(X)\right) = \left(\sum_i a_i w_i(X)\right) + h(X)t(X)$$

Testing for correctness reduces to testing of polynomial equivalence

How to test the latter?

Polynomial Relation for the Entire Scheme

$(a_1, a_2, \ldots, a_n)$ are scheme execution if and only if the following polynomials are equal

$$\left(\sum_i a_i u_i(X)\right) \cdot \left(\sum_i a_i v_i(X)\right) = \left(\sum_i a_i w_i(X)\right) + h(X)t(X)$$

Testing for correctness reduces to testing of polynomial equivalence

In the proving key a random point s is taken, and $g^{u_i(s)}, g^{v_i(s)}, g^{w_i(s)}$ are computed and published with $z' = g^{h(s)t(s)}$

Polynomial Relation for the Entire Scheme

$(a_1, a_2, \ldots, a_n)$ are scheme execution if and only if the following polynomials are equal

$$\left(\sum_i a_i u_i(X)\right) \cdot \left(\sum_i a_i v_i(X)\right) = \left(\sum_i a_i w_i(X)\right) + h(X)t(X)$$

Testing for correctness reduces to testing of polynomial equivalence

In the proving key a random point s is taken, and $g^{u_i(s)}, g^{v_i(s)}, g^{w_i(s)}$are computed and published with $z' = g^{h(s)t(s)}$

The prover can then compute $g^{a_i u_i(s)}$ by taking $g^{u_i(s)}$ to the power of a_i. He can compute $x = g^{\sum_i a_i u_i(s)}$, also $y = g^{\sum_i a_i v_i(s)}$ and $z = g^{\sum_i a_i w_i(s)}$.

Polynomial Relation for the Entire Scheme

$(a_1, a_2, \dots, a_n)$ are scheme execution if and only if the following polynomials are equal

$$\left(\sum_i a_i u_i(X)\right) \cdot \left(\sum_i a_i v_i(X)\right) = \left(\sum_i a_i w_i(X)\right) + h(X)t(X)$$

Testing for correctness reduces to testing of polynomial equivalence

In the proving key a random point s is taken, and $g^{u_i(s)}, g^{v_i(s)}, g^{w_i(s)}$are computed and published with $z' = g^{h(s)t(s)}$

The prover can then compute $g^{a_i u_i(s)}$ by taking $g^{u_i(s)}$ to the power of a_i. He can compute $x = g^{\sum_i a_i u_i(s)}$, also $y = g^{\sum_i a_i v_i(s)}$ and $z = g^{\sum_i a_i w_i(s)}$.

Now verifier can check if $e(x, y) = z \cdot z'$

Polynomial Relation for the Entire Scheme

$(a_1, a_2, \dots, a_n)$ are scheme execution if and only if the following polynomials are equal

$$\left(\sum_i a_i u_i(X)\right) \cdot \left(\sum_i a_i v_i(X)\right) = \left(\sum_i a_i w_i(X)\right) + h(X)t(X)$$

Testing for correctness reduces to testing of polynomial equivalence

In the proving key a random point s is taken, and $g^{u_i(s)}, g^{v_i(s)}, g^{w_i(s)}$are computed and published with $z' = g^{h(s)t(s)}$

The prover can then compute $g^{a_i u_i(s)}$ by taking $g^{u_i(s)}$ to the power of a_i. He can compute $x = g^{\sum_i a_i u_i(s)}$, also $y = g^{\sum_i a_i v_i(s)}$ and $z = g^{\sum_i a_i w_i(s)}$.

Now verifier can check if $e(x, y) = z \cdot z'$

Wait, what if he cheats and just computes z to be as needed?

Missing Details

To prove that

$$\left(\sum_i a_i u_i(X)\right) \cdot \left(\sum_i a_i v_i(X)\right) = \left(\sum_i a_i w_i(X)\right) + h(X)t(X)$$

Proving key also contains for random $\alpha, \beta, \gamma, \delta$

$$g^{\alpha}, g^{\beta}, g^{\gamma}, g^{\delta}, g^{\frac{\beta u_i(s)+\alpha v_i(s)+w_i(s)}{\delta}}, z' = g^{\frac{h(s)t(s)}{\delta}}$$

Prover computes

$$A = g^{\alpha+\left(\sum_i a_i u_i(s)\right)}, B = g^{\beta+\left(\sum_i a_i v_i(s)\right)}, C = g^{\sum_i a_i \frac{\beta u_i(s)+\alpha v_i(s)+w_i(s)}{\delta}}$$

Verifier checks if

$$e(A, B) = e\left(g^{\alpha}, g^{\beta}\right) \cdot e(Cz', g^{\delta})$$

Only 2 uncacheable pairing computations! Any incorrect a_i will make C inconsistent with A, B, and the inconsistency is impossible to correct if you do not know α, β, δ, s

More Missing Details

Some more complexity:

- Prover randomizes his outputs so extra variables r, x are introduced and another pairing operation is performed by Verifier.
- Pairing is of type-III, so three different G groups and three generators.
- Input variables are treated differently, and another pairing is needed.
- g^{s^j} for all j are published instead of $g^{u_i(s)}, g^{v_i(s)}$ in order to make proving key smaller. This makes Prover to do extra work to recompute the polynomial values using FFT.