SNARKs, STARKs, and Bulletproofs

Dmitry Khovratovich

Sikoba
Dusk
Evernym
ABDK Consulting

June 25th, 2019

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, cryptocurrencies, 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?

How to do that so that Bob can not cheat?

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

ZK-STARKs

Simple example

Program:

1. Take input $X_0 = X$;
2. Compute $X_i \leftarrow (X_{i-1}^2 + 3)$ up to $i = 100$.
3. Return $A = X_{100}$.
4. No big number arithmetics, only lowest 10 digits (modulo 10^{10}.

Simple example

Program:

1. Take input $X_0 = X$;
2. Compute $X_i \leftarrow (X_{i-1}^2 + 3)$ up to $i = 100$.
3. Return $A = X_{100}$.
4. No big number arithmetics, 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:

1. Take input $X_0 = X$;
2. Compute $X_i \leftarrow (X_{i-1}^2 + 3)$ up to $i = 100$.
3. Return $A = X_{100}$.
4. No big number arithmetics, 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 one equation and any cheat is detected with probability 99%.

Protocol

Program:

1. Take input $X_0 = X$;
2. Compute $X_i \leftarrow (X_{i-1}^2 + 3)$ up to $i = 100$.
3. Return $A = X_{100}$.
4. No big number arithmetics, 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 one equation and any cheat is detected with probability 99%.

How exactly?

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 *for all* x $C(f(x), f(x+1)) = 0$.

Details

Code	Value	f
X_0	1	$f(0)$
X_1	4	$f(1)$
X_2	19	$f(2)$
X_3	364	$f(3)$
$\ldots$		
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 *for all* x $C(f(x), f(x+1)) = 0$.
- 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

$$C(x, y) = y - x^2 - 3;$$
$$D(x) = x(x-1)(x-2)\cdot(x-99);$$
$$g(x) = C(f(x), f(x+1))/D(x).$$

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	$f(100)$
...	...	...
	?	$f(10000)$

Details

$$C(x, y) = y - x^2 - 3;$$
$$D(x) = x(x-1)(x-2)\cdot(x-99);$$
$$g(x) = C(f(x), f(x+1))/D(x).$$

Bob goes on.

- Compute f and g up to 10000.

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	$f(100)$
...	...	...
	?	$f(10000)$

Details

$$C(x, y) = y - x^2 - 3;$$
$$D(x) = x(x-1)(x-2) \cdot (x-99);$$
$$g(x) = C(f(x), f(x+1))/D(x).$$

Bob goes on.

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	$f(100)$
...	...	...
	?	$f(10000)$

- Compute f and g up to 10000.
- Commit to the evaluations:
  $H_1 = Hash(f(0), f(1), \ldots, f(10000));$
  $H_2 = Hash(g(0), g(1), \ldots, g(10000));$
- 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 $f(s), f(s+1), g(s)$.

Equation

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	$f'(3)$
$\ldots$	$\ldots$	$\ldots$
X_{100}	5251434499	$f(100)$
$\ldots$	$\ldots$	$\ldots$
	?	$f(10000)$

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

- He can not 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 $X_i, i > 100$.

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.

ZK-SNARKs

Pairings

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

Pairing e is a function of two arguments such that

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

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.

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

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

- 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.

How to get a single equation from many

$$\begin{aligned}
&\text{For } x = 0, x \neq 1, 2 \quad a_3 = a_1 \cdot a_2;\\
&\text{For } x = 1, x \neq 0, 2 \quad a_4 = a_2 \cdot a_3;\\
&\text{For } x = 2, x \neq 0, 1 \quad a_5 = a_1 \cdot (a_4 + a_2).
\end{aligned}$$

Proper multiplication:

$$\begin{aligned}
a_3(x-1)(x-2)/2 &= ((x-1)(x-2)/2)a_1 \cdot ((x-1)(x-2)/2)a_2;\\
-a_4x(x-2)/2 &= (x(x-2)/2)a_2 \cdot (x(x-2)/2)a_3;\\
x(x-1)a_5 &= x(x-1)a_1 \cdot (x(x-1)a_4 + x(x-1)a_2).
\end{aligned}$$

Altogether:

$$a_1a_2(x^2 - 3x + 2) + a_2a_3(x^2 - 2x) + \ldots = 0$$

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.

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

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.

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

Thus 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.

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

Thus testing for correctness reduces to testing of polynomial equivalence. How to test the latter?

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.

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

Thus testing for correctness reduces to testing of polynomial equivalence. How to test the latter?

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 $x = g^{\left(\sum_i a_i u_i(s)\right)}$, also $y = g^{\left(\sum_i a_i v_i(s)\right)}$ and $z = g^{\left(\sum_i a_i w_i(s)\right)}$.

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.

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

Thus testing for correctness reduces to testing of polynomial equivalence. How to test the latter?

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 $x = g^{\left(\sum_i a_i u_i(s)\right)}$, also $y = g^{\left(\sum_i a_i v_i(s)\right)}$ and $z = g^{\left(\sum_i a_i w_i(s)\right)}$.

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

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.

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

Thus testing for correctness reduces to testing of polynomial equivalence. How to test the latter?

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 $x = g^{\left(\sum_i a_i u_i(s)\right)}$, also $y = g^{\left(\sum_i a_i v_i(s)\right)}$ and $z = g^{\left(\sum_i a_i w_i(s)\right)}$.

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

$$\left(\sum_i a_i u_i(s)\right)\left(\sum_i a_i v_i(s)\right) = \left(\sum_i a_i w_i(s)\right) + h(s)t(s).$$

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

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

Verifier checks if

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

Only 2 uncacheable pairing computations! Note that 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.

Bulletproofs

Overview

1. Circuit is represented as a system of equations over variables $\mathbf{a}$;
2. We commit to $\mathbf{a}$.
3. Given challenges y, z the system is equivalent to proving

$$F(\mathbf{a}, y, z) = c$$

   for some constant c.
4. As F is quadratic in $\mathbf{a}$, we create two polynomials $\mathbf{l}$ and $\mathbf{r}$ so that $t(X) = \langle \mathbf{l}, \mathbf{r} \rangle$ has its coefficient by X^2 equal to $F(\mathbf{a}, y, z)$.
5. We commit to all other coefficients of t.
6. Given challenge x, we provide $\mathbf{l}(x)$ and $\mathbf{r}(x)$.
7. Prove that $\langle \mathbf{l}(x), \mathbf{r}(x) \rangle$ matches the committed coefficients of t and constant c.
8. Prove that $\mathbf{l}(x), \mathbf{r}(x)$ are correctly built from $\mathbf{a}$.
9. This proves that the term by X^2 of $t(x)$ is equal to c, as claimed

Program is described by nonlinear

$$\mathbf{a_L} \circ \mathbf{a_R} = \mathbf{a_O}, \tag{1}$$

and linear equations:

$$W_L \mathbf{a_L} + W_R \mathbf{a_R} + W_O \mathbf{a_O} + W_V \mathbf{v} = \mathbf{c}. \tag{2}$$

Details 1

Program is described by nonlinear

$$\mathbf{a_L} \circ \mathbf{a_R} = \mathbf{a_O}, \tag{1}$$

and linear equations:

$$W_L \mathbf{a_L} + W_R \mathbf{a_R} + W_O \mathbf{a_O} + W_V \mathbf{v} = \mathbf{c}. \tag{2}$$

One equation for everything given challenges y, z:

$$\mathbf{z}^q W_L \mathbf{a_L} + \mathbf{z}^q W_R \mathbf{a_R} + \mathbf{z}^q W_O \mathbf{a_O} + \langle \mathbf{a_L} \circ \mathbf{a_R} - \mathbf{a_O}, \mathbf{y}^n \rangle = \langle \mathbf{z}^q, \mathbf{c} \rangle - \mathbf{z}^q W_V \mathbf{v}. \tag{3}$$

where $\mathbf{z}^q = (z, z^2, \ldots, z^q), \mathbf{y}^n = (1, y, y^2, \ldots, y^{n-1})$.

$$\mathbf{z}^q W_L \mathbf{a_L} + \mathbf{z}^q W_R \mathbf{a_R} + \mathbf{z}^q W_O \mathbf{a_O} + \langle \mathbf{a_L} \circ \mathbf{a_R} - \mathbf{a_O}, \mathbf{y}^n \rangle = \langle \mathbf{z}^q, \mathbf{c} \rangle - \mathbf{z}^q W_V \mathbf{v}. \tag{4}$$

$$\mathbf{z}^q W_L \mathbf{a_L} + \mathbf{z}^q W_R \mathbf{a_R} + \mathbf{z}^q W_O \mathbf{a_O} + \langle \mathbf{a_L} \circ \mathbf{a_R} - \mathbf{a_O}, \mathbf{y}^n \rangle = \langle \mathbf{z}^q, \mathbf{c} \rangle - \mathbf{z}^q W_V \mathbf{v}. \tag{4}$$

Consider

$$\mathbf{l}(X) = \mathbf{a_L} X + \mathbf{a_O} X^2 + \mathbf{s_L} X^3 + \mathbf{y}^{-n} \mathbf{z}^q W_R X;$$
$$\mathbf{r}(X) = (\mathbf{y}^n \circ \mathbf{a_R}) X - \mathbf{y}^n + \mathbf{z}^q W_O + \mathbf{z}^q W_L X + (\mathbf{y}^n \circ \mathbf{s_R}) X^3.$$

Let $t(X) = \langle \mathbf{l}(X), \mathbf{r}(X) \rangle = \sum_{i \in [1,6]} t_i X^i$. Then (3) holds if

$$t_2 = \mathbf{y}^{-n} \mathbf{z}^q W_R \mathbf{z}^q W_L + \langle \mathbf{z}^q, c \rangle - \mathbf{z}^q W_v \mathbf{v}$$

It remains to prove the correctness of the scalar product, which is a cool but tedious logarithmic-size protocol.

Tradeoffs

1. Circuit is represented as a system of equations over variables $\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_k$;
2. We commit to $\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_k$.
3. Given challenges y, z the system is equivalent to proving

$$F(\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_k, y, z) = c$$

   for some constant c.
4. As F is quadratic in $\mathbf{a}$, we create two polynomials $\mathbf{l}$ and $\mathbf{r}$ so that $t(X) = \langle \mathbf{l}, \mathbf{r} \rangle$ has its coefficient by X^{2k} equal to $F(\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_k, y, z)$.
5. We commit to all other coefficients of t.
6. Given challenge x, we provide $\mathbf{l}(x)$ and $\mathbf{r}(x)$.
7. Prove that $\langle \mathbf{l}(x), \mathbf{r}(x) \rangle$ matches the committed coefficients of t and constant c – this is k times faster!
8. The rest is the same.

Resulting tradeoff

k	Proof size	Verifier cost
1	$2\log(n) + 10$	1
2	$2\log(n) + 15$	0.5
4	$2\log n + 26$	0.25
8	$2\log n + 46$	0.125
$(\log n)/3 + 3$	$4\log n + 18 - 2\log\log n$	$3/\log(n)$
$n^{1/3}$	$6n^{1/3}$	$1/n^{1/3}$

OVerall performance

1 mln multiplication gates			
	Proof time	Proof size	Verification time
libsnark Pinocchio	32 s	250 B	40 ms
Ligero	5 s	200 KB	2 s
ZK-STARK	1 s	78 KB	18ms
ZKBoo	5 s	100 MB	2 s
Hyrax	100 min	100 KB	1 min
Aurora	7 mins	250 KB	1.5 sec
Bulletproofs	5 mins	1.7 KB	15 sec

Table: Performance of notable VC implementations.