0.1 Introduction

ZK-STARK (Zero-Knowledge Scalable Transparent Argument of Knowledge) is a cryptographic proof system that allows one party to prove to another the knowledge of a piece of information without revealing the information itself, while ensuring scalability and transparency.

In this context, "scalable" implies that the time required by the prover grows at most quasilinearly (linear up to the logarithmic factor) relative to the runtime of the witness-checking process. Additionally, the verification (including both time and proof size) is limited to a polylogarithmic growth concerning this runtime.

In turn, "transparent" means there is no requirement for a trusted setup, unlike SNARKs. STARK protocol utilizes advanced mathematical techniques like Fast Reed-Solomon IOP of Proximity and Merkle trees to achieve this. The security of STARK lies on the difficulty of computing the inverse function to the hash function, so we can consider STARK as a quantum-safe protocol if the used hash function also owns this property.

Note, that the term "STARK" does not specify the protocol interactivity. But today, most of the STARK protocols (or probably all of the existing protocols) are deployed in the non-interactive environment (which makes all of them SNARKs). This means that we really do not need the additional abbreviation for the existing STARK protocols – all of them can be considered as a "transparent SNARKs".

0.2 STARK-friendly fields

In general, STARK protocol can work over any field $\mathbb{F}$ with high two-adicity.

Definition 0.1. We call two-adicity fields, the fields where we can select the multiplicative subgroup of order 2^k.

To be honest, all protocol steps are followed with powers of two. It will be shown, why the groups we are working over must be of size 2^k and why the input data also follows this rule. As the result, the maximum size of the statement that we can prove using the STARK protocol is strictly depends on the size of two-adicity subgroup (that is why we label some fields to have *high two-adicity* or *low two-adicity*).

Remark. In our example we consider using field over prime modulus $N = 3 \cdot 2^{30} + 1$ and subgroups of size 2^{13} and 2^{10}.

As we will work in the new subgroup we may want to specify the subgroup generator to be used in future equations. So, for the multiplicative group generator $w \in \mathbb{F}_N^\times$, the generator of the subgroup will be $w^{\frac{N-1}{2^k}}$.

Example. For the prime field $\mathbb{F}_N$ where $N = 3 \cdot 2^{30} + 1$, the order of $\mathbb{F}^\times$ is $N - 1 = 3 \cdot 2^{30}$. If we take $w = 5$ as the primitive element, the multiplicative subgroup of 2^{13} elements generator will be $5^{3 \cdot 2^{17}}$

This kind of subgroups comes with very useful property: for each element in two-adicity subgroup H, the additive inverse element can be calculated by a simple equation over the element power.

Let's denote the generator of this subgroup h and, as we already examined, $h = w^{\frac{N-1}{|H|}}$, for

simplicity. Then, each element can be represented as $x = h^i = w^{\frac{N-1}{|H|} \cdot i}$ where its additive inverse is $-x = h^j = w^{\frac{N-1}{|H|} \cdot j}$. Then, the i and j values obtain the following property: $j = i + \frac{|H|}{2} \mod |H|$.

Evaluation of the additive inverse element. The sum of x and $-x$ must equal to zero by modulus N, so:

$$x + (-x) \equiv w^{\frac{N-1}{|H|} \cdot i}(1 + w^{\frac{N-1}{|H|} \cdot \frac{|H|}{2}}) \equiv 0 \mod N$$

If the multiplication of two equations equals to zero by modulus N, then we can try to check whether one of these equations is zero modulo N:

$$\begin{aligned}
1 + w^{\frac{N-1}{|H|} \cdot \frac{|H|}{2}} &\equiv 0 \mod N \\
1 + w^{\frac{N-1}{2}} &\equiv 0 \mod N \\
w^{\frac{N-1}{2}} &\equiv N - 1 \mod N \\
w^{N-1} &\equiv (N-1)^2 \mod N \\
w^{N-1} &\equiv N^2 - 2N + 1 \mod N \\
1 &\equiv 1 \mod N
\end{aligned}$$

□

Remark. The equation $w^{N-1} \equiv 1 \mod N$ is obtained from the order property of the primitive element w in the multiplicative group $\mathbb{F}^\times$.

Remark. This provides us with an additional important property beyond element's power computation: when working with a negative element, its power shift equals half the size of the subgroup so, squaring the elements within this subgroup results in a smaller subgroup, reduced by a factor of two. Consequently, to compute the square of the subgroup, it suffices to square only the first half of its elements (powers $0, 1, 2, 3, \ldots, \frac{H}{2}$).

0.3 Protocol definition

0.3.1 Trace, evaluation domain and commitment

Now, we are going to prove that some statement holds on the given sequence of elements.

Definition 0.2. We call **trace** a sequence of elements from $\mathbb{F}$ that represents our witness. This sequence contains private and public values together and follows certain constraints.

Example. The Fibonacci square sequence is a sequence of elements defined as follows:

$$a_i = a_{i-1}^2 + a_{i-2}^2$$

Then, we can for example prove the following statement: *I know a field element x such that the* 1023*rd element of the Fibonacci square sequence starting with* 1 *and x is* 2338775057. (The private x in this case equals to 3141592).

Following the Unisolvence Theorem, the trace a is implied to be an evaluation of some unknown **trace polynomial** of degree $|a|-1$. Also, to be evaluable on the two-adicity subgroup, the size of the trace has to be a power of two.

Definition 0.3. We call **domain** a two-adicity subgroup $G \in \mathbb{F}$ where we evaluate our polynomials.

Example. In our example, we put trace a sequence a of first 1023 elements of the Fibonacci square sequence over $\mathbb{F}_N$, where $N = 3 \cdot 2^{30} + 1$.

$$1, 1, 2, 5, 29, ...$$

To interpolate our trace polynomial we select as a domain a two-adicity subgroup of 2^{10} elements from $\mathbb{F}^\times$ with generator $g = 5^{\frac{3 \cdot 2^{30}}{2^{10}}}$ (here 5 stands for the primitive element in $\mathbb{F}_N^\times$):

$$G = \{g^i \mid g = 5^{3 \cdot 2^{20}} \wedge i \in [0; 1024)\}$$

Next, using the Lagrange interpolation over $(g^i, a_i)_0^{|a|-1}$ points we compute a trace polynomial $f \in \mathbb{F}[x]$.

Remark. Note, that in practice, you may want to use more efficient algorithm for interpolation, for example – *Fast Fourier Transform (FFT)*.

Definition 0.4. We call **evaluation domain** a two-adicity coset $E = wH \in \mathbb{F}$, where $H \in \mathbb{F}$ is a two-adicity subgroup, that is larger ρ times (some small constant) then the domain.

Example. In our case we select a two-adicity subgroup of 2^{13} elements from $\mathbb{F}^\times$ ($\rho = 8$):

$$H = \{h^i \mid h = 5^{3 \cdot 2^{17}} \wedge i \in [0; 8192)\}$$

Then, we define the evaluation domain as:

$$E = \{5 \cdot h_i \mid \forall h_i \in H\}$$

We build a Merkle tree over the values $f(e_i),\ \forall e_i \in E$ and label it's root as a **trace polynomial commitment**. This approach will also be used to commit other polynomials during the protocol walkthrough.

The **constraints** in STARK protocol are expressed as polynomials evaluated over the trace cells, which are satisfied if and only if the computations are correct.

Example. Obviously, our initial statement consists of the following three requirements:

1. The element a_0 is equal to 1;
2. The element a_{1022} is equal to 2338775057;
3. Each element a_{i+2} is equal to $a_{i+1}^2 + a_i^2 \mod N$.

To verify that our committed trace polynomial satisfies all constraints, we can check that

it has corresponding roots. In particular, according to the selected interpolation points (g^i, a_i), the relation $r(a_i, a_j) = 0$ can be rewritten as $r(f(g^i), f(g^j)) = 0$.

Example. For our Fibonacci trace we have the following constraints to be checked over the interpolated polynomial:

1. *The element* a_0 *is equal to* 1 translated to: $f(x) - 1$ has root at $x = g^0 = 1$;
2. *The element* a_{1022} *is equal to* 2338775057 translated to: $f(x) - 2338775057$ has root at $x = g^{1022}$;
3. *Each element* a_{i+2} *is equal to* $a_{i+1}^2 + a_i^2$ translated to: $f(g^2x) - f(gx)^2 - f(x)^2$ has roots in $G \setminus \{g^{1021}, g^{1022}, g^{1023}\}$

To ensure that the specified polynomials have roots in given values, we can use the following property: if polynomial $f(x) \in \mathbb{F}[x]$ has root in x_0 then the $\frac{f(x)}{x - x_0}$ is also a polynomial in $\mathbb{F}[x]$.

Example. Finally, we define the following STARK constraints:

$$p_0(x) = \frac{f(x) - 1}{x - 1}$$
$$p_1(x) = \frac{f(x) - 2338775057}{x - g^{1022}}$$
$$p_2(x) = \frac{f(g^2x) - f(gx)^2 - f(x)^2}{\prod_{i=0}^{1020} x - g^i}$$

Unfortunately, the p_2 polynomial still looks inconvenient to work with, so we may want to simplify it (this is not a part of the protocol in general, but you always may want to simplify your equations to achieve better proving time). Using the following property we can reduce the denominator of p_2:

$$x^{|G|} - 1 = \prod_{g \in G} (x - g), \forall x \in G$$

This equation works because both sides are polynomials whose roots are exactly the elements of G. Note, that while evaluating our polynomial on lager domain then G we should only ensure that the resulting polynomial still holds the relation $f(g^i) = a_i$, so it is acceptable to use properties that only work over G. So, finally we have:

$$p_2(x) = \frac{(f(g^2x) - f(gx)^2 - f(x)^2)(x - g^{2021})(x - g^{2022})(x - g^{2024})}{x^{1024} - 1}$$

In addition, there is one obvious requirement for the STARK constraints: the verifier should be able to compute the constraints polynomials $p_i(x)$ using only the given trace polynomial evaluations for the certain x.

Remark. In our Fibonacci example, verifier can check the constraint polynomials evaluation by requesting only $f(x)$, $f(gx)$ and $f(g^2x)$ – the values committed in the trace polynomial commitment.

To combine all our constraints into a single polynomial, we can follow a commonly used principle by taking a linear combination with the challenges from the verifier. In particular, after

receiving trace polynomial commitment from the prover, the verifier selects $\{\alpha_i \in \mathbb{F}\}$ and sends it to the prover. Then, the prover puts the **composition polynomial** as:

$$CP(x) = \sum \alpha_i \cdot p_i(x)$$

Additionally, prover also commits this polynomial by evaluating on the evaluation domain and building a Merkle tree.

Example. The Fibonacci composition polynomial looks like as follows:

$$CP(x) = \alpha_0 p_0(x) + \alpha_1 p_1(x) + \alpha_2 p_2(x) =$$
$$\alpha_0 \frac{f(x) - 1}{x - 1} + \alpha_1 \frac{f(x) - 2338775057}{x - g^{1022}} +$$
$$\alpha_2 \frac{(f(g^2x) - f(gx)^2 - f(x)^2)(x - g^{2021})(x - g^{2022})(x - g^{2024})}{x^{1024} - 1}$$

0.3.2 FRI protocol

In general, our goal is to verify that the committed polynomial $CP(x)$ satisfies all our constraints, by checking it's evaluation at a random point from the evaluation domain that the verifier selects. Anyway, we can face the problem when the malicious prover constructs a lager polynomial that accepts lots of possible roots from our field (even 2^{64} field is still insecure for just checking the evaluation at one point). That is why we have to make sure that the committed polynomial degree lies in the acceptable range (the upper bound depends on the trace size).

The final stage of the STARK protocol is a **Fast Reed-Solomon IOP of Proximity (FRI)**. FRI is a protocol between a prover and a verifier, which establishes that a given evaluation belongs to a polynomial of low-degree. In this context *low* means no more than ρ times bigger than the trace.

The key idea of FRI protocol is to move from a polynomial of degree n to a polynomial of degree $n/2$ until we get a constant value. Let's consider the polynomial $z_0(x) = \sum a_i \cdot x^i$ of degree $n = 2^t$ and the evaluation domain $E_0 = E$. We suppose to group the *odd* and the *even* coefficients of the z_0 together into the two separate polynomials(z_0^o and z_0^e respectively):

$$z_0^o(x^2) = \sum_{i=0}^{n/2} (a_{2i+1} \cdot x^{2i})$$
$$z_0^e(x^2) = \sum_{i=0}^{n/2} (a_{2i} \cdot x^{2i})$$

Or, in more comfortable form (we have already examined why searching of $-x$ can be done easily in our two-adicity subgroup):

$$z_0^e(x^2) = \frac{z_0(x) + z_0(-x)}{2}$$
$$z_0^o(x^2) = \frac{z_0(x) - z_0(-x)}{2x}$$

Then, we define a next-layer of the FRI polynomial as $z_1(x^2) = z_0^e(x^2) + \beta z_0^o(x^2)$, where β is a challenge received from verifier. The next-layer evaluation domain is also simple to compute: $E_1 = \{(w \cdot h_i)^2 \mid i \in [0; \frac{|E_0|}{2})\}$, because squaring the other elements in E_0 will result in the same values.

Next, we commit to the $z_1(x^2)$ using a next-layer evaluation domain E_1 (is also reduced by a factor two) and continue to repeat the described operations until $z_i(x^{2^i})$ becomes constant.

Interactive ZK-STARK protocol

The prover and the verifier run the interactive version of the ZK-STARK protocol. Both know the statement to be proved, that is defined by the constraint polynomials and the field $\mathbb{F}$ to work over. Prover also knows the witness to be able to generate the trace.

Preparation

- ✓ The prover interpolates trace polynomial $f(x)$ and submits it's commitment to the verifier.
- ✓ The verifier selects challenges random $\alpha_0, \alpha_1, \alpha_2 \in \mathbb{F}$ and sends to the prover.
- ✓ The prover builds the composition polynomial $CP(x)$ and submits it's commitment to the verifier.

FRI

- ✓ The verifier selects random $i \in [0; |E|)$, puts $c = w \cdot h^i$ and sends it to the prover.
- ✓ The prover responds with the $CP(c), CP(-c)$ and all $f(x)$ required to check CP evaluation with corresponding Merkle proofs to them.
- ✓ The verifier checks Merkle proofs and the evaluation of $CP(c)$ by evaluating the constraints polynomials $p_i(c)$.
- ✓ The prover and the verifier go through the FRI protocol for $z_0(x) = CP(x)$ where the prover commits to the layer-i polynomial $z_i(x)$, the verifier selects a challenge β and queries from the prover $z_i(c), z_i(-c)$ to compute $z_{i+1}(c)$ until $z_i(x), i \leq log_2(\deg CP(x))$ becomes constant.

The non-interactive version of the presented protocol can be easily built obtaining the Fiat-Shamir heuristics.

The soundness of the presented STARK protocol follows from the impossibility to commit any possible evaluation of the forgery $CP(x)$ over evaluation domain E and simultaneously prove that $CP(x)$ is a low-degree polynomial by the FRI protocol. Since the size of E is ρ times bigger then the maximum allowed polynomial degree (that directly depends on the size of the trace), the attacker either can't construct such a polynomial or can't construct a low-degree polynomial, so a valid low-degree composition polynomial can only be obtained using a valid trace.

Example. Finally, let's overview the first steps of the ZK-STARK protocol applied to our Fibonacci example:

1. The protocol defines the public constraints such as 2023-th element of sequence, field $\mathbb{F}$, etc.
2. The prover generates the trace a where $a_0 = 1, a_1 = 3141592, a_i = a_{i-1}^2 + a_{i-2}^2$, evaluates the trace polynomial $f(x)$ over the evaluation domain and sends it's commitments to the verifier.
3. The verifier selects challenges $\alpha_0, \alpha_1, \alpha_2 \in \mathbb{F}$ and shares them with the prover.
4. The prover evaluates the composition polynomial $CP(x)$ over evaluation domain and sends it's commitments to the verifier .
5. The verifier selects random $i \in [0; 8192 - 16)$, puts $c = 5 \cdot h^i$ and sends it to the prover.
6. The prover responds with the $f(c), f(gc), f(g^2c), CP(c), CP(-c)$ and corresponding Merkle proofs to them.
7. The verifier checks Merkle proofs and the evaluation of $CP(c)$ by evaluating the constraint polynomials $p_0(c), p_1(c), p_2(c)$.
8. The prover and the verifier go through the FRI protocol for $z_0(x) = CP(x)$ until $z_i(x), i \in [1; 12)$ becomes constant.

0.4 Protocol security

Most of the existing versions of the STARK protocol leverage on several optimizations to achieve better proving and verification time. The key point here is that each FRI query check adds $log_2(\rho)$ bits of security, so we can skip some of these checks if the security level is already satisfied. One more optimization is to include a proof-of-work computation into the protocol that should be done before FRI with dependency on the committed values. It can be useful because the verification of the proof-of-work is less expensive then the verification of the FRI step while still increases the computation cost for the malicious prover.

More precisely, let's assume that the desired security level of the protocol is λ. First of all, we obviously have to use a proper collision-resistant hash function with 2λ bits output. Then, according to the StarkWare's definition of the STARK protocol, the resulting security is defined as follows:

$$\lambda \geq min\{\delta + log_2(\rho) \cdot s, log_2(|F|)\} - 1$$

where δ – number of the proof-of-work bits, s – number of the FRI queries.

Example. If the protocol is deployed over 256-bit field and the domain ratio is $\rho = 3$, to achieve the 128 bit security we can for example execute 33 FRI query and evaluate 29 proof-of-work bits: $min\{29 + 3 \cdot 33, 256\} = 128$.