0.1 Introduction

ZK-STARK (**Z**ero-**K**nowledge **S**calable **T**ransparent **A**rgument of **K**nowledge) 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 to the hash function, so we can consider STARK as a quantum-safe protocol if the used hash function also inherits 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. The primary reason for that is that STARKs can work only with NTT-friendly fields, and the NTT-friendly fields are the fields where we can select the multiplicative subgroup of order 2^k for sufficiently many values of k.

Definition 0.1. We call **two-adicity fields**, the fields where we can select the multiplicative subgroup of order 2^k for sufficiently many values of k. In this case, the field order p is typically of form $p = 2^m \cdot p' + 1$ where p' is a small integer.

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 initial discussion we consider using field over prime modulus $p = 3 \cdot 2^{30} + 1$ 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 $w \in \mathbb{F}_p^\times$, the generator of the subgroup of order 2^k is $\omega_k = w^{\frac{p-1}{2^k}}$, as was shown in the NTT section.

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

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

Proposition 0.2. Suppose $\mathbb{H} \leq \mathbb{F}_p$ is a subgroup of order r with generator $h = w^{(p-1)/r}$. Then, the additive inverse for $x = h^i \in \mathbb{H}$ is h^j where $j = i + \frac{r}{2} \pmod r$.

Proof. The sum of x and $-x$ must equal to zero modulo p, so:

$$x + (-x) = w^{(p-1)i/r} + w^{(p-1)j/r} = w^{(p-1)i/r}(1 + w^{(p-1)(j-i)/r}) = w^{(p-1)i/r}(1 + w^{(p-1)/2})$$

Now note that $w^{(p-1)/2} = -1$ which completes the proof. ■

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

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 $0, 1, 2, 3, \ldots, \frac{r}{2}$).

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 0.3. 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 over $\mathbb{F}$ as follows:

$$a_{j+2} = a_{j+1}^2 + a_j^2$$

Then we can, for example, prove the following statement: *I know a field element $w \in \mathbb{F}$ such that the k^{th} element of the Fibonacci square sequence (a_k) starting with x and w is y.* Formally, this can be written as:

$$\mathcal{R}_{\text{Fib}} = \left\{ \begin{array}{c} \textbf{Public Statement: } (x, y, k) \\ \textbf{Witness: } w \end{array} \;\middle|\; \begin{array}{c} a_0 = x, a_1 = w, a_k = y \text{ with} \\ a_{j+2} = a_{j+1}^2 + a_j^2 \text{ for all } j \in [k] \end{array} \right\}$$

For concreteness, let us take $k = 1023$, $x = 1$, and $y = 2338775057$.

Following the Unisolvence Theorem, the trace $\{a_j\}_j$ is implied to be an evaluation of some unknown **trace polynomial** of degree equal to the length of the sequence $\{a_j\}_j$. Also, to be evaluable on the two-adicity subgroup, the size of the trace has to be a power of two.

Definition 0.4. We call **domain** a two-adicity subgroup $\mathbb{G} \leq \mathbb{F}^\times$ where we evaluate our polynomials.

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

$$1, 1, 2, 5, 29, \ldots$$

To interpolate our trace polynomial we select as a domain a two-adicity subgroup of 2^{10} elements from $\mathbb{F}_p^\times$ with a generator $g = 5^{\frac{3 \cdot 2^{30}}{2^{10}}} = 5^{3 \cdot 2^{20}}$ (here 5 is the primitive element in the multiplicative group $\mathbb{F}_p^\times$). That being said, $\mathbb{G} = \{g^i\}_{i \in [1024]}$.

Next, using the Lagrange interpolation over $(g^j, a_j)_{j \in [k]}$ points we compute a trace polynomial $f \in \mathbb{F}[x]$. Note that the interpolation can be done in $O(k \log k)$, as shown in NTT section.

Definition 0.5. We call **evaluation domain** a two-adicity coset $\mathbb{E} = w\mathbb{H} \leq \mathbb{F}_p^\times$, where $\mathbb{H} \leq \mathbb{F}_p^\times$ is a two-adicity subgroup, that is larger $\rho \in \mathbb{N}$ times (typically a relatively small constant) than the domain. In other words, $\mathrm{ord}(\mathbb{H}) = \rho \cdot \mathrm{ord}(\mathbb{G})$.

Example. In our case we select a two-adicity subgroup $\mathbb{H}$ of 2^{13} elements from $\mathbb{F}_p^\times$ with $\rho = 8$ as $\mathbb{H} = \{h^i\}_{i \in [8192]}$ where $h = 5^{3 \cdot 2^{17}}$. Then, we define the *evaluation domain* as $\mathbb{E} = 5\mathbb{H} = \{5h^i\}_{i \in [8192]}$.

We build a Merkle tree over the values $\{f(e)\}_{e \in \mathbb{E}}$ and label its 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 $a_{i+1}^2 + a_i^2$.

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 $\{(g^i, a_i)\}_{i \in [k]}$, 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 $\mathbb{G} \setminus \{g^{1021}, g^{1022}, g^{1023}\}$

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). Note that p_2 is *almost* a vanishing polynomial of $\mathbb{G}$, which has a form $x^{\text{ord}(\mathbb{G})} - 1$, except for points $g^{1021}, g^{1022}, g^{1023}$. In other words, we can simplify the denominator as:

$$\prod_{i=0}^{1020}(x - g^i) = \frac{x^{1024} - 1}{(x - g^{1021})(x - g^{1022})(x - g^{1023})}$$

Note, that while evaluating our polynomial on a larger domain then $\mathbb{G}$ 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 $\mathbb{G}$. 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 scalars $\alpha_1, \ldots, \alpha_m$ and sends it to the prover. Then, the prover puts the **composition polynomial** as:

$$\text{CP}(x) := \sum_{j=1}^{m} \alpha_j \cdot p_j(x)$$

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:

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

0.3.2 FRI protocol

In general, our goal is to verify that the committed polynomial $\mathrm{CP}(x)$ 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 larger 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 $z_0(x) = \sum_i a_i \cdot x^i$ of degree $n = 2^t$ and the evaluation domain $\mathbb{E}_0 = \mathbb{E}$. 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 a 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: $\mathbb{E}_1 = \{(w \cdot h_i)^2\}_{i \in [\mathrm{ord}(\mathbb{E}_0)/2]}$ as squaring the other elements in $\mathbb{E}_0$ will result in the same values.

Next, we commit to the $z_1(x^2)$ using a next-layer evaluation domain $\mathbb{E}_1$ (is also reduced by a factor two) and continue to repeat the described operations until $z_j(x^{2^j})$ 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 $\mathbb{F}_p$ 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 its commitment to the verifier.
- ✓ The verifier selects challenges random $\alpha_i \in \mathbb{F}_p$ and sends to the prover.
- ✓ The prover builds the composition polynomial $\mathrm{CP}(x)$ and submits its commitment to the verifier.

FRI

- ✓ The verifier selects random $j \in [\mathrm{ord}(\mathbb{E})]$, sets $c \leftarrow w \cdot h^j$ and sends it to the prover.
- ✓ The prover responds with the $\mathrm{CP}(c), \mathrm{CP}(-c)$ and all $f(x)$ required to check CP evaluation with corresponding Merkle proofs to them.
- ✓ The verifier checks Merkle proofs and the evaluation of $\mathrm{CP}(c)$ by evaluating the constraints polynomials $p_j(c)$.
- ✓ The prover and the verifier go through the FRI protocol for $z_0(x) = CP(x)$ where the prover commits to the layer-j polynomial $z_j(x)$, the verifier selects a challenge β and queries from the prover $z_j(c), z_j(-c)$ to compute $z_{j+1}(c)$ until $z_k(x), j \leq \log_2(\deg \mathrm{CP})$ 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 $\mathrm{CP}(x)$ over evaluation domain $\mathbb{E}$ and simultaneously prove that $\mathrm{CP}(x)$ is a low-degree polynomial by the FRI protocol. Since the size of $\mathbb{E}$ 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 $\mathbb{F}_p$, 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 $\mathrm{CP}(x)$ over evaluation domain and sends it's commitments to the verifier.
5. The verifier selects random $i \in [8192 - 16]$, puts $c = 5 \cdot h^i$ and sends it to the prover.
6. The prover responds with the $f(c), f(gc), f(g^2c), \mathrm{CP}(c), \mathrm{CP}(-c)$ and corresponding Merkle proofs to them.
7. The verifier checks Merkle proofs and the evaluation of $\mathrm{CP}(c)$ 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 $z_0(x) = \mathrm{CP}(x)$ until $z_i(x), i \in [12]$ 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:

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

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: $\min\{29 + 3 \cdot 33, 256\} = 128$.