Eigen-zkVM Stark example

Eigen Labs

1 Commitments

1.1 Merkle Tree

In a Merkle tree, the green nodes represent the values of the polynomial committed to by the prover (such as the polynomial values $a = f(0), b = f(1), c = f(2), d = f(3)$. Each committed hash value forms the leaf nodes of the Merkle tree (nodes 4, 5, 6, and 7 in the diagram). All other non-leaf nodes are the hash of the concatenation of their corresponding child nodes (for example, $\mathsf{hash}(\#2) = \mathsf{hash}\,(\mathsf{hash}(\#4), \mathsf{hash}(\#5))$. The root node is referred to as the Merkle root.

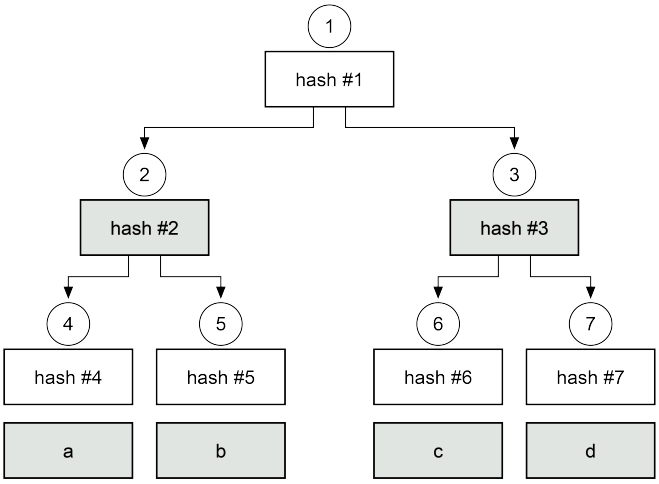

Fig. 1 Merkle Tree.

As illustrated in Fig 1, a distinctive feature of the Merkle tree is that any change in a single leaf node will result in a change in the Merkle root. Conversely, if it can be proven that the root hash remains constant, the entire tree must also be constant. This means that the leaf node values have not been modified.

1.2 Polynomial Commitments Using Merkle Trees

- `Commit:` Generate a Merkle root based on the values of the polynomial (like the four nodes shown in the diagram).
- `Decommit:` Calculate a random point (such as node c) based on $root(f)$. The data transmitted is $(root(f), c, path(c))$ where, $path(c)$ is the hash of nodes 7 and 2, serving as the verification path for node c.
- `Verify:` Compute a new Merkle $root'$, using $(c, path(c))$. Perform a consistency check by comparing $root'$ and the received Merkle root $root$. If $root' = root$, accept; otherwise, reject.

2 Low-degree Detection

2.1 Direct Tests

For a constant polynomial $f(x) = c$, at a fixed point z_1 and a random point w calculated based on Fiat-Shamir, the verifier checks if

$$f(z_1) = f(w). \tag{1}$$

If it is true, the verifier accepts that the degree of the polynomial $f(x)$ is less than 1. For a linear polynomial

$$f(x) = bx + c \tag{2}$$

at fixed points $(z_1, f(z_1))$ and $(z_2, f(z_2))$ and a random point w calculated based on Fiat-Shamir, the verifier checks if the third point $(w, f(w))$ lies on the same line as $(z_1, f(z_1))$ and $(z_2, f(z_2))$.

If this is the case, the verifier accepts that the degree of the polynomial $f(x)$ is less than 2. Extension: A polynomial $f(x)$ of degree d or lower requires d fixed points and one random point for verification. Given three points, a quadratic curve is determined, and a random point is selected for verification.

2.2 Linear Combination Test

Suppose there are two polynomials $f(x)$ and $g(x)$ of degree at most d. Using the direct testing method above requires $2(d + 1)$ points for execution testing. To optimize this testing method, use the Fiat-Shamir heuristic to compute a random number α, and combine polynomials $f(x)$ and $g(x)$ as:

$$h(x) := f(x) + \alpha \cdot g(x). \tag{3}$$

This formula is called a linear combination. The prover constructs a Merkle tree based on the values of $h(x)$ and sends the Merkle root to the verifier. Then the verifier can verify the Merkle tree and only needs $d + 1$ points to verify that the degree of $h(x)$ is less than d. Note that the degree of $h(x)$ is $\max\{\deg(f), \deg(g)\}$.

2.3 Polynomial Folding Lemma

Given that polynomial $f(x)$ have degree d, where $d = 2^n$. Let $g(x^2)$ be the even terms of $f(x)$, and $h(x^2)$ be the odd terms of $f(x)$. Then

$$f(x) = g(x^2) + x \cdot h(x^2). \tag{4}$$

The above formula is called the folding formula. Let $y = x^2$, then for y, the degrees of polynomials $g(y)$ and $h(y)$ are $d/2$.

For example: Given a polynomial $f(x)$ of degree $d = 2^3$

$$f(x) = a_0 + a_1x + a_2x^2 + a_3x^3 + a_4x^4 + a_5x^5 + a_6x^6 + a_7x^7 + a_8x^8. \tag{5}$$

The even and odd terms of $f(x)$ are $f(x) = g(x^2) + x \cdot h(x^2)$

$$\begin{aligned} g(x^2) &= a_0 + a_2x^2 + a_4x^4 + a_6x^6 + a_8x^8 \\ xh(x^2) &= a_1x + a_3x^3 + a_5x^5 + a_7x^7 + 0x^9 \end{aligned} \tag{6}$$

Let $y = x^2$, then

$$\begin{aligned} g(y) &= a_0 + a_2y + a_4y^2 + a_6y^3 + a_8y^4 \\ h(y) &= a_1 + a_3y + a_5y^2 + a_7y^3 + 0y^4 \end{aligned} \tag{7}$$

Rewriting as

$$\begin{aligned} g(x) &= a_0 + a_2x + a_4x^2 + a_6x^3 + a_8x^4 \\ h(x) &= a_1 + a_3x + a_5x^2 + a_7x^3 + 0x^4 \end{aligned} \tag{8}$$

Therefore, the degrees of $g(x)$ and $h(x)$ are both $d/2$. The original polynomial of degree d is equivalent to the sum of odd and even terms, with degree decreased to $d/2$. Folding half each time, that is, 2, can generalize the folding 2^n, thus improving the folding efficiency.

2.4 FRI

FRI is used to verify polynomial degree is less than d. By combining the above direct test, linear combination test, and polynomial Folding Lemma, it is possible to test the degree of a polynomial $f_0(x)$ of degree $d = 2^n$ in only $O(\log d)$ rounds.

`Prover:` In step 1,
(1) Commit to the degree d polynomial $f_1(x)$ by computing its Merkle root $root(f_1)$.
(2) Compute the hash of the Merkle root to get a random number z:

$$z := \mathsf{hash}\,(root(f_1)) \tag{9}$$

(3) Computing polynomial values $f_1(z), f_1(-z)$.
`Send:` $\{root(f_1), f_1(z), f_1(-z), path(f_1(z)), path(f_1(-z))\}$.

`Verifier:`
(1) Use the Merkle root $root(f_1)$ and path $path(f_1(z))$ to verify the correctness of $f_1(z)$:

$$root(f_1) = \mathsf{Merkle}\left(f_1(z), path(f_1(z))\right) \tag{10}$$

(2) Similarly, verify $f_1(-z)$ using its path. If both checks pass, accept. Otherwise, reject.

`Prover:` In step 2,
(1) According to the folding formula, the degree d polynomial $f_1(x)$ can be expressed as:

$$f_1(x) = g_1(x^2) + x \cdot h_1(x^2) \tag{11}$$

Let this be Formula 1.
(2) Compute a hash based on the random number z to obtain a new random number α_2:

$$\alpha_2 := \mathsf{hash}(z) = \mathsf{hash}\left(\mathsf{hash}\left(root(f_1)\right)\right) \tag{12}$$

(3) Construct a linear combination polynomial:

$$f_2(x) = g_1(x) + \alpha_2 \cdot h_1(x) \tag{13}$$

(4) Commit to the polynomial $f_2(x)$ by computing its Merkle root $root(f_2)$. Here $f_2(x)$ has degree $d/2$.
(5) Computing polynomial values $f_2(-z^2)$.
`Send:` $\{root(f_2), f_2(-z^2), path(f_2(z^2)), path(f_2(-z^2))\}$.
Note that there is no need to send the value $f_2(z^2)$ since it can be computed directly.
`Verifier:`
(1) Hash the Merkle root $root(f_1)$ to obtain a random number z:

$$z := \mathsf{hash}\left(root(f_1)\right). \tag{14}$$

(2) Using the data $f_1(z), f_1(-z)$ from Step 1, the random number z, and the folding formula (Formula 1), obtain:

$$\begin{aligned} f_1(z) &= g_1(z^2) + z \cdot h_1(z^2) \\ f_1(-z) &= g_1(z^2) - z \cdot h_1(z^2) \end{aligned} \tag{15}$$

This allows solving equations to obtain $g_1(z^2), h_1(z^2)$.
(3) Hash z and $root(f_2)$ to obtain another random number α_2:

$$\alpha_2 := \mathsf{hash}(z, root(f_2)) \tag{16}$$

(4) Using $g_1(z^2), h_1(z^2)$, the random α_2, and the linear combination formula, compute:

$$f_2(z^2) := g_1(z^2) + \alpha_2 \cdot h_1(z^2) \tag{17}$$

to obtain $f_2(z^2)$.
(5) Use the received $root(f_2), path(f_2(z^2))$ to verify $f_2(z^2)$:

$$root(f_2) == \mathsf{Merkle}\left(f_2(z^2), path(f_2(z^2))\right). \tag{18}$$

(6) Similarly verify $f_2(-z^2)$. Accept if both checks pass, otherwise reject.

`Prover:` In step 3,
(1) According to the polynomial folding formula, the degree $d/2$ polynomial $f_2(x)$ can be expressed as:

$$f_2(x) = g_2(x^2) + x \cdot h_2(x^2) \tag{19}$$

Let this be Formula 2.
(2) Hash z and $root(f_2)$ to obtain a random number α_3:

$$\alpha_3 := \mathsf{hash}(z, root(f_2)) \tag{20}$$

(3) Construct a linear combination polynomial:

$$f_3(x) = g_2(x) + \alpha_3 \cdot h_2(x) \tag{21}$$

(4) Commit to $f_3(x)$ by computing its Merkle root $root(f_3)$. Here $f_3(x)$ has degree $d/2^2$.
(5) Computing polynomial values $f_3(-z^4)$.
`Send:` $\{root(f_3), f_3(-z^4), path(f_3(z^4)), path(f_3(-z^4))\}$.
Note that there is no need to send $f_3(z^4)$ since it can be directly computed.
`Verifier:`
(1) Using the verified value $f_2(z^2)$ from Step 1, z^2, and the folding formula (Formula 2), obtain:

$$\begin{aligned} f_2(z^2) &= g_2(z^4) + z^2 \cdot h_2(z^4) \\ f_2(-z^2) &= g_2(-z^4) - z^2 \cdot h_2(-z^4) \end{aligned} \tag{22}$$

This allows solving equations to obtain $g_2(z^4), h_2(z^4)$. (2) Hash z and $root(f_2)$ to obtain the random number α_3:

$$\alpha_3 := \mathsf{hash}(z, root(f_2)). \tag{23}$$

(3) Using $g_2(z^4), h_2(z^4), \alpha_3$, and the linear combination formula, compute:

$$f_3(z^4) := g_2(z^4) + \alpha_3 \cdot h_2(z^4) \tag{24}$$

to obtain$f_3(z^4)$.
(4) Use $root(f_3), path(f_3(z^4))$ to verify $f_3(z^4)$:

$$root(f_3) == \mathsf{Merkle}\left(f_3(z^4), path(f_3(z^4))\right). \tag{25}$$

(5) Similarly verify $f_3(-z^4)$. Accept if both checks pass, otherwise reject.

`Prover:` In step $\log(d)$,
`Send:` $\{root(f_{\log(d)}), f_{\log(d)}(-z^n), path(f_{\log(d)}(z^n)), path(f_{\log(d)}(-z^n))\}$
`Verifier:`
(1) Compute the polynomial value $f_{\log(d)}(z^n)$.
(2) Based on the polynomial folding and linear combination formulas, obtain a polynomial:

$$f_{\log(d)}(x) = g_{\log(d)}(x) + \alpha_{\log(d)} \cdot h_{\log(d)}(x) \tag{26}$$

This polynomial has degree $d/2^{\log(d)} = 1$, so it is a constant polynomial.
(3) The verifier uses direct testing to test the degree of the constant polynomial.

Note that:
(1) For the value range in the above protocol, it is necessary that for each value z in the range L, $-z$ is also in the range L.
(2) The commitment to the polynomial $f_1(x)$ is not over the range L, but rather over the range $L^2 = \{x^2 : x \in L\}$.

2.5 Probability Analysis

For the polynomial values $f_0(z), f_0(-z)$, assume the probability that $f_0(z) = f_0(-z) = 0$ is ε, where $0 < \varepsilon < 1$. Then the probability of other cases is $1-\varepsilon$, where $0 < 1-\varepsilon < 1$.

$$f(x) = g(x^2) + x \cdot h(x^2) \tag{27}$$

If the $1-\varepsilon$ probability case occurs, i.e. $f_0(z) = f_0(-z) \neq 0$, then there exists a unique α_1 that makes the linear combination zero:

$$f_1(z^2) = g_0(z^2) + \alpha_1 \cdot h_0(z^2) = 0 \tag{28}$$

However, α_1 is a random number computed via Fiat-Shamir, so the probability of manipulating α_1 to a desired value is equal to the difficulty of POW. Therefore, the probability that $f_1(z^2) \neq 0$ approaches 1. If the prover cheats on $f_1(z^2)$, the Merkle consistency check will fail. Thus, if the $1-\varepsilon$ probability case occurs, the verifier will reject it.

Conversely, the probability the prover successfully cheats in a round is ε. Over $\log(d)$ rounds, the probability the prover succeeds in every round is $\varepsilon^{\log(d)}$, which

decays exponentially. Therefore, the probability the prover can successfully cheat is negligible. In summary, soundness holds except with negligible probability ε.

2.6 Batch FRI

In the Batched FRI Protocol, the prover aims to demonstrate the closeness of a set of functions $f_0, f_1, ..., f_N : H \leftarrow F$ to low-degree polynomials all at once. It is possible to run individual FRI protocols for each function f_i in parallel. Specifically, the prover directly applies the FRI protocol to a random linear combination of each function f_i. After committing to the functions and receiving a uniformly sampled value ε from the verifier, the prover calculates a function

$$f(X) := f_0(X) + \sum_{i=1}^{N} \varepsilon^i f_i(X), \tag{29}$$

and applies the FRI protocol to it.

The function f is computed as $f_0(X) + \sum_{i=1}^{N} \varepsilon^i f_i(X)$, where a distinct random value $\varepsilon_i \in K$ is used for each function f_i rather than relying on powers of a single value ε. Although this alternative is secure, the soundness bound increases linearly with the number of functions N. Therefore, it's reasonable to assume that N is sublinear in K to maintain protocol security.

In the batched version, the verifier needs to confirm the correct relationship between functions $f_0, f_1, ..., f_N$ and the initial FRI polynomial $p_0 = f$. The verifier uses evaluations of p_0 received from the prover during each FRI query phase. To facilitate the verification

$$\begin{aligned} p_0(r) &:= f_0(r) + \sum_{i=1}^{N} \varepsilon^i f_i(r) \\ p_0(-r) &:= f_0(-r) + \sum_{i=1}^{N} \varepsilon^i f_i(-r), \end{aligned} \tag{30}$$

the prover sends evaluations of functions $f_0, f_1, ..., f_N$ at both r and $-r$, allowing the verifier to perform a local consistency check between $f_0, f_1, ..., f_N$ and p_0. These evaluations are accompanied by their respective Merkle tree paths.

3 Fibonacci Sequence Stark

3.1 zero knowledge proof

For an NP problem F, demonstrate publicly the input value X and the output value Z without revealing the secret input Y. Prove that you know an input value Y that satisfies the computational relationship

$$F(X, Y) = Z \tag{31}$$

such that the verifier accepts this fact.

3.2 Fibonacci's sequence

A prover needs to prove that the 1000th Fibonacci number Z starts from a public value X and a secret value Y, such that $F(X, Y) = Z$. This process is equivalent to proving

$$F_0 := X, F_1 := Y \tag{32}$$
$$F_i := F_{i-2} + F_{i-1} \tag{33}$$
$$Z := F_{1000}. \tag{34}$$

The computation requires n steps and w registers. The trace T is a $n \times w$ table, where $n = 1000, w = 2$. For example, if $X = 3, Y = 4$, the Table 1 can be constructed as follows: The above algorithm can be encoded as transition constraints and boundary

Table 1 Fibonacci's sequence.

n	$T_{n,0}$	$T_{n,1}$
0	3	4
1	4	7
2	7	11
3	11	18
4	18	29
...	...	...
999	F_{999}	F_{1000}

constraints as follows:

$$\begin{cases} T_{i+1,0} = T_{i,1}, \\ T_{i+1,1} = T_{i,0} + T_{i,1}. \end{cases} \tag{35}$$

$$\begin{cases} T_{0,0} = X, \\ T_{999,1} = Z. \end{cases} \tag{36}$$

Therefore, the following inference can be made:

Inference 1. *The prover proving that he knows the secret value Y satisfying the relation $F(X, Y) = Z$ is equivalent to the prover proving that it knows the trace T satisfying the transition constraints transition_constraints and the boundary constraints boundary_constraints.*

3.3 Trace Polynomial

Let $n = 0, ..., 999$ be the x-coordinates of a polynomial, and let the traces stored in register $T_{n,0}$ and register $T_{n,1}$ be the polynomial values, then the value expression of the trace polynomial can be constructed as follows:

$$P_0(i) = T_{i,0}, i = 0, ..., 999. \tag{37}$$
$$P_1(i) = T_{i,1}, i = 0, ..., 999. \tag{38}$$

Therefore, coefficient polynomials $P_0(x), P_1(x)$ of degree 999 can be constructed to express the traces.

The value expression of the trace polynomial can be equivalently transformed into the coefficient expression of the trace polynomial, so there is an equivalent transformation:

Fibonacci sequence ADD triple: $a_n = a_{n-1} + a_{n-2}$
Trace value expression: $T_{i+1,1} = T_{i,0} + T_{i,1}, i = 0, ..., 999$
Polynomial coefficient expression: $P_1(i+1) = P_0(i) + P_1(i), i = 0, ..., 999$
Polynomial coefficient expression: $P_1(i+1) - (P_0(i) + P_1(i)) = 0, i = 0, ..., 999$
Polynomial coefficient expression: $Q(x) := P_1(x+1) - (P_0(x) + P_1(x)) = 0, x = 0, ..., 999$

Note that the degree of $Q(x)$ is 1000. We construct a public target polynomial:

$$R(x) = (x-0)(x-1)(x-2)...(x-999) \tag{39}$$

If $x = 0, ..., 999$, then the target polynomial $R(x) = 0$. There are other solutions x' to the equation $Q(x) = 0$. Thus, we can compute a quotient polynomial

$$C(x) = \frac{Q(x)}{R(x)}. \tag{40}$$

Note that the degree of the quotient polynomial is 1.

Similarly, we can construct a multiplication tuple.

Fibonacci's sequence MUL triple: $a_n = a_{n-1} * a_{n-2}$
Trace value expression: $T_{i+1,1} = T_{i,0} \cdot T_{i,1}, i = 0, ..., 999$
Polynomial coefficient expression: $P_1(i+1) = P_0(i) \cdot P_1(i), i = 0, ..., 999$
Polynomial coefficient expression: $P_1(i+1) - (P_0(i) \cdot P_1(i)) = 0, i = 0, ..., 999$
Polynomial coefficient expression: $Q(x) := P_1(x+1) - (P_0(x) \cdot P_1(x)) = 0, x = 0, ..., 999$

Note that the degree of $Q(x)$ is 2000. The public target polynomial is still:

$$R(x) = (x-0)(x-1)(x-2)...(x-999) \tag{41}$$

If $x = 0, ..., 999$, then the target polynomial $R(x) = 0$. There are other solutions x' to the equation $Q(x) = 0$. Thus, we can compute a quotient polynomial

$$C(x) = \frac{Q(x)}{R(x)}. \tag{42}$$

Note that the degree of the quotient polynomial is 1000.

Similarly, we can construct a multiplication quadruple.

Fibonacci's sequence MUL quadruple: $a_n = a_{n-1} * a_{n-2} * a_{n-3}$
Trace value expression: $T_{i+1,1} = T_{i,0} \cdot T_{i,1} \cdot T_{i,2}, i = 0, ..., 999$
Polynomial coefficient expression: $P_1(i+1) = P_0(i) \cdot P_1(i) \cdot P_2(i), i = 0, ..., 999$
Polynomial coefficient expression: $P_1(i+1) - (P_0(i) \cdot P_1(i) \cdot P_2(i)) = 0, i = 0, ..., 999$
Polynomial coefficient expression: $Q(x) := P_1(x+1) - (P_0(x) \cdot P_1(x) \cdot P_2(x)) = 0, x = 0, ..., 999$

Note that the degree of $Q(x)$ is 3000. The public target polynomial is still:

$$R(x) = (x-0)(x-1)(x-2)...(x-999) \tag{43}$$

If $x = 0, ..., 999$, then the target polynomial $R(x) = 0$. There are other solutions x' to the equation $Q(x) = 0$. Thus, we can compute a quotient polynomial

$$C(x) = \frac{Q(x)}{R(x)}. \tag{44}$$

Note that the degree of the quotient polynomial is 2000.

Therefore, the following inference can be made:

Inference 2. *Algorithms often involve multiple multiplications, so the degree of the quotient polynomial $C(x)$ is usually higher than the degree of the trace polynomial $T(x)$.*

Now we back to Fibonacci's sequence ADD triple, the transition constraints *transition_constraints* can be equivalently transformed into the following equations

$$T_{i+1,1} = T_{i,0} + T_{i,1} \Leftrightarrow C_0 = \frac{P_1(x+1) - (P_0(x) + P_1(x))}{\prod^{i}_{[0,...,998]} (x-i)} \tag{45}$$

$$T_{i+1,0} = T_{i,1} \Leftrightarrow C_1 = \frac{P_0(x+1) - P_1(x)}{\prod^{i}_{[0,...,998]} (x-i)} \tag{46}$$

And the boundary constraints *boundary_constraints* can be equivalently transformed into the following

$$T_{0,0} = X \Leftrightarrow C_2(x) = \frac{P_0(x) - X}{x-0} \tag{47}$$

$$T_{999,1} = Z \Leftrightarrow C_3(x) = \frac{P_1(x) - Z}{x-999} \tag{48}$$

Therefore, the following inference can be made:

Inference 3. *The prover proving that he knows the trace T satisfying the transition constraints transition_constraints and the boundary constraints boundary_constraints is equivalent to the prover proving that he knows the polynomials $P_0(x), P_1(x)$ such that $C_0(x), C_1(x), C_2(x), C_3(x)$ are degree D polynomials.*

On the contrary, if the verifier accepts that $C_0(x), C_1(x), C_2(x), C_3(x)$ are degree D polynomials, then the transition constraints and boundary constraints hold. Therefore,

the verifier accepts the prover's proof that he knows the secret value Y satisfying the relation $F(X, Y) = Z$.

3.4 Stark Workflow

Using the Fiat-Shamir transformation, calculate the random numbers $\alpha_0, \alpha_1, \alpha_2, \alpha_3$. Transform the linear combination constraint polynomials $C_0(x), C_1(x)$ and boundary constraint polynomials $C_2(x), C_3(x)$

$$C(x) = \alpha_0 C_0(x) + \alpha_1 C_1(x) + \alpha_2 C_2(x) + \alpha_3 C_3(x) \tag{49}$$

Using the polynomial halving lemma,

1. The prover provides the Merkle root *Root* of the coefficients of the polynomial $C(x)$ and proves that its degree is 2^n;
2. Equivalently, the prover uses the random number β_0 from the Fiat-Shamir transformation to prove the Merkle root $Root_0$ of the coefficients of the polynomial $C^0(x)$ and proves that its degree is 2^{n-1};
3. Equivalently, the prover uses the random number β_1 from the Fiat-Shamir transformation to prove the Merkle root $Root_1$ of the coefficients of the polynomial $C^1(x)$ and proves that its degree is 2^{n-2};
4. And so on, ...
5. Equivalently, the prover uses the random number β_n from the Fiat-Shamir transformation to prove the Merkle root $Root_n$ of the coefficients of the polynomial $C^n(x)$ and proves that its degree is 2^0;
6. Finally, the verifier verifies the above halving process, verifies the Merkle root $Root_n$ of the coefficients of the polynomial $C^n(x)$, and verifies that the degree of the polynomial $C^n(x)$ is 1. This is equivalent to verifying the Merkle root *Root* of the coefficients of the polynomial $C(x)$ and proving that its degree is 1024. This, in turn, is equivalent to verifying that the trace T satisfies the transition constraints and boundary constraints, which is equivalent to verifying $F(X, Y) = Z$ without knowing the secret Y.