STARK – A Proof System Unlocking Blockchain Scalability

What Does The ZK-STARK Do to Achieve Scalability? (For Dummies)

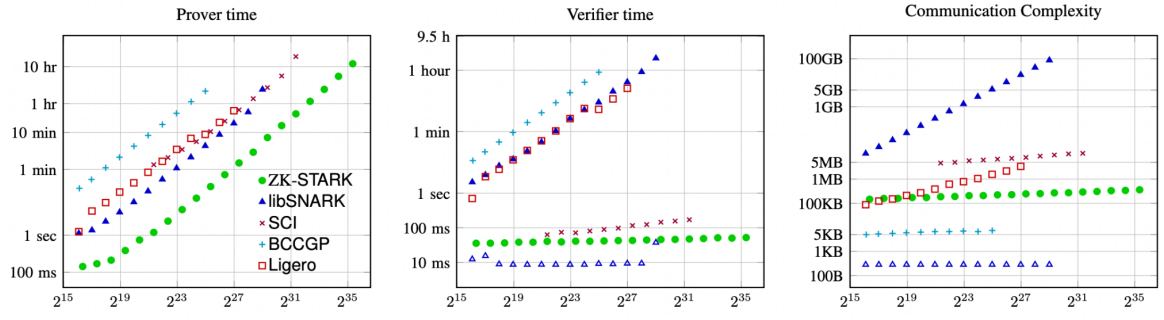
By: Phurinut Rungrojkitiyos, *StarkNet House Research Fellow*

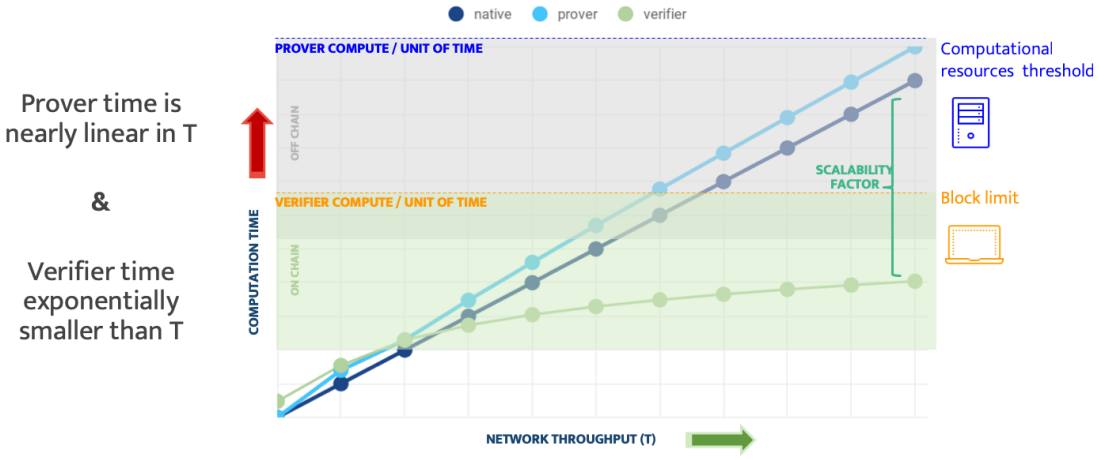
Integrity

Computational Integrity (CI) is one of the goals and the reason for humanity to use blockchain technologies. In the traditional world, the integrity of an activity was proven by some “trustable” people, devices, or trustable institutions like banks, governments, and others. Noticeably, the traditional-world integrity comes along with the centralization problem and probability of corruption. As the blockchain has come, the common mechanism to prove the integrity has been shifted from using the trustable third party to check to using all nodes on the network to re-run the computation of all transactions and gain the public trust.

Scalability

The first letter of the acronym, STARK, stands for "Scalability". What this term tries to express is the ability of the new proof system to scale a blockchain system. But, why do we want to scale the blockchain system at first? The answer is that we want to increase the complexity and the number of activities on blockchain while the cost decreases. Unfortunately, we cannot just simply upgrade people's computers into more powerful machines. Therefore, the light at the end of this tunnel is the new proof system, called Zero-Knowledge STARK (ZK-STARK). This proof system allows computational integrity to be proven and succinctly verified dramatically faster than re-running the computation in the traditional way.





The pictures show the performance of ZK-STARK and compare with others

The Top View of How Does This Proof System Work?

The process of this proof system is about the talk between a prover and the verifier. What makes this conversation that seems very plain able to prove the computational integrity is the objective of the prover and the verifier that is set oppositely. For the prover, their job is to convince a verifier of the integrity of a certain computation by answering and giving some information that is queried by the verifier. In contrast, the public will entrust the verifier to play the role of a suspicious gatekeeper charged with determining which claims are true and which are false.

Amazingly, these two people, the verifier and the prover, do not use human languages to talk to each other. Instead, they use polynomials. *In order to make the verifier decide to trust a certain computational integrity statement, the prover has to convince the verifier that the polynomial is indeed of low degree. The verifier will be convinced that the polynomial is of low degree if and only if the original computation is correct.* At this point, it is quite clear about the goal of the proof system and the roles of the prover and the verifier in this proof system. However, there is an important step that has to be done before they can start talking to each other to prove the integrity statement. It is called Arithmetization,

and it is about converting or translating the computational integrity statement into a polynomial.

Arithmetization

Step 1: Convert The CI Statement into formal algebraic language – prepare the ingredients

The first step of arithmetization is like preparing ingredients for the chef to cook (to translate everything into a polynomial) in the second step. And, what has to be prepared here are the execution trace and the polynomial constraints. Firstly, the execution trace can be visualized with a table as shown in the example below; however, every stage of the computation must be shown in this table, while each row corresponds to a different stage. Essentially, the last characteristic of this table is that each row also has to be able to be verified with the row next to it.

item	price
Avocado	$4.98
Apple	$7.98
Milk	$3.45
Bread	$2.65
Brown Sugar	$1.40
total	**$20.46**

item	price	running total
Avocado	$4.98	$0.00
Apple	$7.98	$4.98
Milk	$3.45	$12.96
Bread	$2.65	$16.41
Brown Sugar	$1.40	$19.06
total	**$20.46**	$20.46

The picture on the left shows the ordinary table, and on the right shows the table that will be used as an execution trace.

From the table on the right hand side, we can see that we can check the computational integrity by using the constraint that

$$running\ total_{afer} = running\ total_{before} + price$$

And, this can apply to every pair of rows in the table, so this is why we can achieve the succinct constraint.

The next item that we have to prepare for the chef is polynomial constraints. Indeed, polynomial constraints are the set of rules that all of the elements in the trace are satisfied if and only if the trace represents a valid computation. The equation, $running\ total_{after} = running\ total_{before} + price$, written above, is one of the constraints from the set of them.

In order to continue to use this great visualizable example from STARK and to express the values easily, we will define A_{ij} as the value of the i[th] row and j[th] column item in the receipt. Therefore, now we can fully write the set of constraints as follows.

1) $A_{0,2} = 0$ // We start the running total from 0.
2) $\forall\ 1 \le i \le 5 : A_{i,2} - A_{i-1,2} - A_{i-1,1} = 0$ // Each row's running total is correct.
3) $A_{5,1} - A_{5,2} = 0$ // The last running total is the total sum.

As we can see now, we have already got all the important ingredients, execution trace and the polynomial constraints that hold if and only if the sum in the receipt is correct, to be used in the second step of the arithmetization. Nevertheless, this example only shows a case of the constraint for this trace; the appearance of constraints depends upon the trace, which can be even more complex.

Step 2: transforming the two ingredients into a single low-degree polynomial

This step aims to combine the execution trace and the polynomial constraints that we built in the previous step into building a single low-degree polynomial such that the polynomial is of low degree if and only if the execution trace satisfies the polynomial constraints.

Before going deeper into the process of combining the execution trace and the polynomial constraints into a polynomial, the goal of doing all of these should be reminded again that we want to build a proof system that can prove and verify faster. Therefore, we don't want the verifier to ask the prover too many questions (without this second step of arithmetization the verifier might have to go through every computation on the trace and ask the prover). This is possible because in the second step of arithmetization we applied the technique, called "Error Correction Codes" to our execution trace.

We can think of the Error Correction Codes as an amplifier; it is used to amplify the errors in the trace if they exist. To use this, we have to transform the execution trace into a polynomial first. Then, extend the domain of that polynomial; this is where the errors and incorrect execution traces are amplified (this amplification is so high that the small errors can be distinguished easily). Lastly, we use the polynomial we got and the polynomial constraints to generate another polynomial that is guaranteed to be of low degree if and only if the execution trace is valid. Thus, this last polynomial is the last product and the key to proving the computational integrity that we are always looking for.

Conclusion

ZK-STARK is able to achieve scalability because this proof system allows the verifier to probabilistically prove the computational integrity statement of the prover. In other words, the verifier can randomly check some of the computations; however, it is still able to notice incorrectness in the computations with high performance (the chance of soundness is very small). These advantages of the ZK-STARK will allow people to do more extreme computation on the chain while the cost is reduced.

References

[1] Segal, & Peled. (2021, December 7). *Arithmetization I - StarkWare*. Medium. Retrieved September 16, 2022, from https://medium.com/starkware/arithmetization-i-15c046390862

[2] Peled. (2021, December 8). *Arithmetization II - StarkWare*. Medium. Retrieved September 16, 2022, from https://medium.com/starkware/arithmetization-ii-403c3b3f4355

[3] Ben-Sasson, E. (2017, September 8). *ECCC - TR17-134*. Retrieved September 16, 2022, from https://eccc.weizmann.ac.il/report/2017/134/

[4] Wikipedia contributors. (2022b, April 14). *Interactive proof system*. Wikipedia. Retrieved September 16, 2022, from https://en.wikipedia.org/wiki/Interactive_proof_system

[5] Wikipedia contributors. (2022a, March 18). *Probabilistically checkable proof*. Wikipedia. Retrieved September 16, 2022, from https://en.wikipedia.org/wiki/Probabilistically_checkable_proof

[6] Wikipedia contributors. (2022c, September 15). *Reed–Solomon error correction*. Wikipedia. Retrieved September 16, 2022, from https://en.wikipedia.org/wiki/Reed%E2%80%93Solomon_error_correction

[7] Ben-Sasson, E. (2018). *Scalable, transparent, and post-quantum secure computational integrity*. Retrieved September 16, 2022, from https://eprint.iacr.org/2018/046

[8] Riabzev, & Ben-Sasson. (2021, December 7). *STARK Math: The Journey Begins - StarkWare*. Medium. Retrieved September 16, 2022, from https://medium.com/starkware/stark-math-the-journey-begins-51bd2b063c71

[9] *STARKs, Part 3: Into the Weeds*. (2018, July 21). Retrieved September 16, 2022, from https://vitalik.ca/general/2018/07/21/starks_part_3.html

[10] *STARKs, Part I: Proofs with Polynomials*. (2017, November 9). Retrieved September 16, 2022, from https://vitalik.ca/general/2017/11/09/starks_part_1.html

[11] *STARKs, Part II: Thank Goodness It's FRI-day*. (2017, November 22). Retrieved September 16, 2022, from https://vitalik.ca/general/2017/11/22/starks_part_2.html