Zero-Knowledge Proofs

The geek equivalent of asking your maths teacher to give you full marks for just writing 'Hence Proved'

Bharat Kumar Ramesh and Swapnika Nag
Jul 6

ZK proofs have been all the rage recently, in large part due to the potential it has as a Layer-2 scaling solution for Ethereum. *(You probably have come across the term ZK-SNARK),* which is touted as a solid form of zero knowledge proofs used by chains such as zkSync, Polygon Hermez, etc.

In this post, we delve deeper into some of the fundamental questions around this, starting with what are zero knowledge proofs and why are they useful.

We'll then move to a high-level overview of how ZK-SNARKs work, and the likely impact on the Eth ecosystem. Let's dive in.

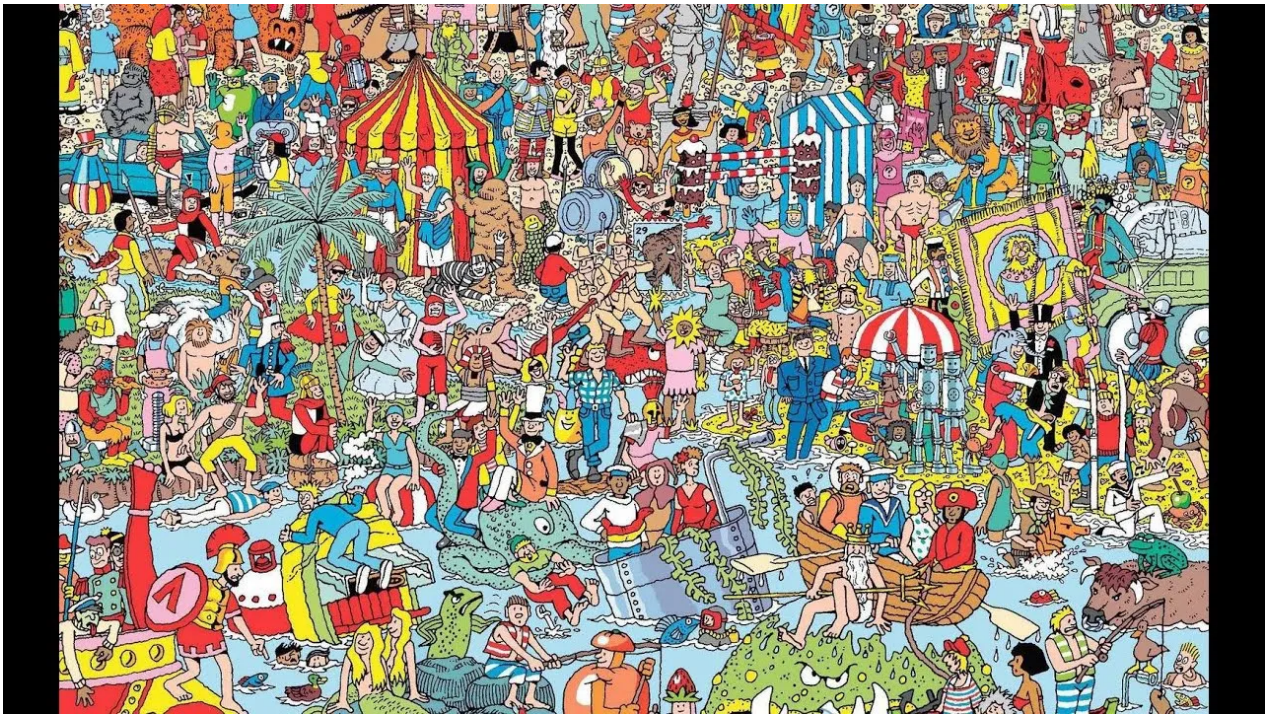
What is a zero-knowledge proof?

Let's start with the basic premise we want to solve for.

Say there is a statement or fact that I hold. Can I (the prover) prove to you (the verifier) that the statement is indeed true, without revealing any additional information.

Put differently, the verifier should only know if it is indeed true or false, and nothing more.

Let's understand this better with a simple example of finding waldo. Try and find waldo in this image:



Now let us say I come to you and say that **I have a powerful computer vision algorithm that can detect waldo in less than 2 secs**. But I don't want to show you that algorithm or send it to a third-party for verification before the sale.

And being the pragmatic buyer that you are, you don't want to buy it without any diligence that it actually is legitimate.

So to resolve this, we devise an experiment. You pick a series of images of your choice (any number n), and show me the images one-by-one. If my algorithm works, I should be able to find waldo within that time for each and every instance.

Once this has been successfully done many times, you (the verifier) can eliminate any luck involved, and are therefore now confident that my algorithm indeed is capable of 'finding waldo'. And yet, you've gained no more knowledge about the workings of the algorithm.

This is a simple form of a zero-knowledge proof. There is zero knowledge passed to the verifier in the proof, and yet they can determine if it is true or false.

So why is it relevant?

We exchange proofs all the time (under the hood) in our daily interactions.

For example, every time you access an application, you are proving to the server that you are indeed the legitimate owner of that account. This is is done today through the use of a password - *This is a secret that is known only to you, so when you input that into your browser*

or app, it converts the password string into an encrypted form, and sends it to the server. The server compares the encrypted text against what is stored, and if it matches, authenticates the user.

While this is the common practice, it still involves sending across the password over a server. And that's a potential point of failure.

What would be ideal is if we could construct a zero-knowledge proof, such that the server is convinced that the user is indeed the owner of the account, without the need to transmit any more information

Here's another example. Say you need to prove to a lessor that you are financially sound to be a tenant. Sending across a bank statement is a method of verifying it. But ideally, we should be able to send a proof to the landlord confirming that we satisfy the condition for tenancy without having to disclose any more information

How does it work?

There are multiple types of zero-knowledge proofs, but a broad classification can be based on the interactivity. There are **interactive proofs** (where the prover and verifier engage synchronously to prove it) and **non-interactive proofs** (where the prover and verifier engage async)

Captcha is a form of interactive proof. When you perform a random task, you prove that you are not a bot, without revealing any more personal information about yourself to the verifier (application). The actions of the proof happens synchronously, and hence is interactive.

A **non-interactive proof** is one where the prover generates a proof, which can then be independently verified by a verifier (or many). This has the advantage of being composable, and persistent. Naturally, this is far more advantageous than an interactive proof. (And consequently, more expensive as well)

At the base of it, a non-interactive proof system requires the prover to construct some form of a proof that has a few implicit properties:

- Easy to construct if the underlying fact is true; **Impossibly expensive to construct otherwise**
- Easily verifiable by the verifier (independently)
- Reveals no more information except whether the fact is true or false

As an interesting thought exercise, think about how you would construct a non-interactive zero-knowledge proof for the 'Find Waldo' algorithm.

One rudimentary answer is to shoot a video, where people picked at random, select a 'find waldo' image from the internet, and the algorithm detects waldo within the 2s time period. Ofcourse, this satisfies condition 2 and 3, but not 1, as it is not really hard to construct this proof even if the algorithm does not work, as you could game the audience

And what does this have to do with rollups and SNARKs?

Now that we know what's a ZK proof, let's come to an often-read term in the context of Ethereum scaling called rollups and SNARKs

ZK-based scaling has been touted as the most promising scaling solution for Ethereum, and a great solve to the problem of high gas fees in the ecosystem.

At a high level, here's what a rollup does. Say 3 transactions have occured since the last block has been mined. For simplicity, let us say that 3 linear transfers occur between 4 wallet addresses (A → B → C → D)

There are two ways to compute the new state of the Ethereum ecosystem

- **Status quo (Base Layer) -** I add all the 3 transactions to a block that is on the base Ethereum layer (also known as Layer 1 or L1), *i.e. A → B is added as a transaction on the chain, and likewise, so are the other two.*

  Anyone watching the chain can now compute the new state by taking the original state of the chain, and adjusting it based on each transaction.

  This is expensive to scale since I need to have all the transactions broadcast to all nodes, and stored on-chain. And ofcourse, this has some privacy limitations
- **Rollup - Alternatively,** In a rollup, a node called the relayer takes these 3 transactions and the current state of the Ethereum chain.

  It stores the transactions off-chain, and computes the end-state of the Ethereum ecosystem, *(which is basically, decrease A by an amount, and increase D by the same amount).*

  These set of transactions is computed into a single rollup transaction, which is

recorded on the chain. Because the transactions are stored off-chain, it allows for incredible scaling of the transaction throughput.

There is a catch here though - the system breaks down if the other nodes cannot be certain that the transactions within a rollup are valid.

And to solve precisely this problem, rollups look to construct a zero-knowledge proof such that any other nodes can verify it, and be certain that the transactions in the rollup are all valid, **without requiring any more information about the transactions)**

One of the methodologies to generate this proof is called a ZK-SNARK. This stands for Zero Knowledge Succinct Non-Interactive Argument of Knowledge

- Zero Knowledge - *the verifier gets no more information about the rollup beyond whether it is valid or not*
- Succinct - *the proof itself is much smaller than the transactions, so easy to store on chain*
- Non-Interactive - *the proof can be generated by the prover, and verified independently by many verifiers at any point of time*
- Argument of Knowledge - *a way to make the acryonym sound cooler than ZK-SNIP (Succint Non-Interactive Proof)*

How does a ZK-SNARK work?

Consensys has a great article on it, and I'm just going to leave that here
- https://consensys.net/blog/developers/introduction-to-zk-snarks/

Essentially, there is a program that has a proof generator function and a verification function. And there is a publicly known key pair of a 'prover key' and a 'verification key'. These are specifcially generated one-time for the program *(generation of this key-pair is a critical point of failure, so orgs use elaborate ceremonies to ensure that no single entity is involved in corrupting this process. Checkout the massive precautions that ZCash took here*
- https://z.cash/technology/paramgen/)

The prover does the following:

- Takes a list of transactions and adds to the rollup - *this is the private input* ***priv_t***
- Computes the new state of the chain from these transactions - *this is a public value* ***pub_x***
- Takes the **prover_key**

- Runs the prover function in the program with these parameters (priv_t, pub_x, prover_key) and generates a **proof**
- Broadcasts the **proof** and the new state of the chain **pub_x** to all verifiers

Each verifier can now do the following:

- Takes the public output or the new state of the chain i.e. **pub_x**
- Receives the **proof** (from the prover)
- Takes the **verification_key**
- Runs the verification function in the program with these parameters (pub_x, proof, verification_key)
- The program returns true or false

The program uses cryptographic methods such that it is possible for the prover to compute a true proof if they have a valid **priv_t.** But computationally infeasible to do so, if they do not have a valid **priv_t**

And correspondinly, it is easy for a verifier to verify the same. Therefore, if the proof is computed to true, the verifier can be confident that the private input is valid, without knowing anything more about the input

Therefore, by storing just the proof, along with the new state of the chain on the base layer of Ethereum, one can abstract away a lot of the transactions away from L1, and store them off-chain. This significantly increases the scaling potential of the ecosystem, and reduces cost, without compromising on the core underlying security.

We're already seeing strong signs of this. Multiple projects are in the race to create the dominant rollup solution including Aztec, zkSync, Polygon Hermez, StarkNet, Arbitrum, etc. **Watch this space - we're in for some exciting times ahead**