Zero-Knowledge Proof

Rosenblatt, Santiago

University of Oxford, February, 2020

Abstract

Nowadays, most of the people is using applications that capture or store their personal data (e.g. contact information). But is there any way we can still utilize services or ask for health insurance without sharing our health information? The answer is, it might be! So, this is why we are going to talk about Zero-Knowledge Proofs.

As we keep evolving and almost everything tends to involve technology, being able to prove you have some information or meet certain criteria, without revealing the information, is key. This is the reason, through this paper, we are going to talk about zero-knowledge proof, explaining what it is, giving some examples, and finally presenting the design of a simple TCP server, that assists two people who want to know if they are working with the same secret.

1. Introduction

Privacy in the current era is key, and having the ability to prove you know something without sharing the information, is extremely useful and allows working in a more secure environment.

Zero-Knowledge Proofs exist since the 1980s with the purpose of assisting in the above scenario.

2. Zero-Knowledge Proof

Zero - Knowledge Proofs allow data to be verified without revealing exactly what it is.

In this protocol, each transaction has a *verifier* and a *prover*. The prover's purpose is to prove to the verifier that he meets with some criteria or has information, but without telling the verifier anything about that.

After finishing the transaction, the verifier should know whether the prover possesses what he claims to.

2.1 Properties

A Zero-Knowledge Proof needs to meet three properties or criteria: *Completeness*, *Soundness* and *Zero-Knowledge*.

Completeness, implies that if something is true, it can be proven.

Soundness gives security to the verifier. It ensures that the output will be true, only, and only if the prover meets what the verifier wants to prove.

Finally, Zero-Knowledge, ensures that only the statement being proven is revealed but nothing more, neither the secret. If the prover and verifier are not working with the same secret, the verifier must not get to know the real secret the prover possesses.

2.2 Types

When working with Zero-Knowledge Proofs, we can find two types: interactive and non-interactive.

2.2.1 Interactive

Interactive is the basic Zero-Knowledge Proof one can find. In this case, the prover needs to perform a series of actions to finally prove to the verifier that has what he claims to.

The verifier asks a lot of questions, and the prover needs to answer all of them correctly.

2.2.2 Non-Interactive

As it can be deducted from its name, the non-interactive type, does not rely in an interactive process between the verifier and prover.

This type relies hardly in mathematics and uses a *key generator function*, a *prover function* and a *verifier function*.

The generator function, receives a secret parameter and generates a pair of keys, a prover key and a verification key.

The prover function, receives as an input the prover key, the common secret and a witness. With all of this, it outputs a proof.

Finally, the verify function, receives the verifier key, the common secret and the proof. As an output, this function tells if there is a match between the secrets or not.

2.3 Use Cases

Today there are some applications being used, and specifically in blockchain, non-interactive zero-knowledge proofs become extremely useful.

We will briefly mention the Zcash non-interactive example and a simple illustrative example for the interactive case.

2.3.1 Two Billiard Balls (Interactive)

In this case, there are two balls (red and green) and two friends, Alice and Bob.

Alice wants to prove Bob, who is color-blinded, that the two balls have different colors. To do so, Alice tells bob to switch (or not) the balls behind his back, multiple times, and that she will tell if he has switched them.

She tells him to do it 30 times and is accurate in all of them. Given the probability is having being randomly that accurate is extremely low (2^{-30}), Bob now believes what Alice claimed in the beginning.

2.3.2 Zcash (Non-Interactive)

Using zk-SNARK[1], zCash validates the data of each transaction without validating information of the amount and the parties involves.

In the flow, the sender of a transaction creates a proof that shows with high probability:

- The input values sum.
- He possesses the private spending keys for the input notes.
- The private spending keys of the input notes are cryptographically linked to a signature over the entire transaction.

3. Designing a Zero Knowledge Proof Protocol

For the design of our protocol, we took into account both the zero-knowledge proof properties, as well as other properties that ensure it is safer. Specially, to solve some of the last group properties, we thought about designing a TCP server that is assumed to be run by a trusted third party.

3.1 Satisfied Properties

Complying with the Zero-knowledge proof properties is essentially what makes a Zero-Knowledge Protocol. However, in order to ensure more security, other things could be taken into account. All this this will be answered in the following sub sections.

3.1.1 Soundness and Completeness

Our server receives a secret submitted by the prover, and encrypts it using bcrypt with a cost of 10, saving it only then.

After the above, it returns a uuid and a pair of keys to use by the prover and the verifier.

The verifier can then use the uuid, verifier key, and its secret, to find if the secrets match.

[1] Ben-Sasson, Eli; Chiesa, Alessandro; Tromer, Eran; Virza, Madars, 2019. *Succint Not-Interactive Zero Knowledge for a von Neumann Architecture* - https://eprint.iacr.org/2013/879.pdf.

As the probability of colliding is extremely low ($½^{64}$), both soundness and completeness are solved.

3.1.2 Zero-Knowledge

The closest to the secret our server stores, is the hashed secret, but doesn't keep any information that can be linked to the submitter. Other than that, the server only tells the verifier if the secrets match.

3.1.3 Disallow Dishonesty

When the prover tries to verify the secrets, using the verifier key, the secret and the uuid, if they match, the server stores there is a match, or in the opposite case that they do not.

The verifier has five attempts per hour to verify against a certain uuid, and if there is a match, it is not allowed to change it anymore.

With this measure, the prover can then access the server using other operation, and check the current state of the secret (waiting for verifier, match or no match), being sure that as the server is run by a trusted third-party and not by the verifier, the later will not be able to intercept the communications and tell that the secrets do not match when they actually do.

3.1.4 Privacy Preserving

As mentioned before, the only information the server has about the prover and the verifier are the keys it generated and the hashed secret. There is anything stored that can link the hashed secret to the interested parties.

3.1.5 Zero Information Leakage

When secrets are unequal, that is the only thing the parties will know, as that is what state will reflect, but the server will never output as the result of an operation the secret.

3.1.6 Protection Against Mathematical Attacks

On a first instance, the possibility of finding a secret by using brute forcing, is remote.

Besides having to know the verifier key and uuid of where the hashed secret is located, there is a velocity check in place, that limits the attempts of verifying a secret to only 5 times per hour.

3.2 Attack Vectors and Counter Measures

Different attack vectors can be found when working with Zero-Knowledge Proofs. Some of them, are mentioned below, together with the corresponding counter measure.

3.2.1 Impersonation

In this attack, an attacker pretends to be the other entity. By combining a uuid with a different key for the prover and verifier, this attack tends to not be feasible.

3.2.2 Modifying the Response Provided

A verifier could try to modify its response after having known the secrets match.

When secrets match, the server disallows future modifications to the state, so the prover could know the secrets are matching as well.

3.2.3 Man in the Middle Attack

This wide know attack, could be countered if both prover and verifier agree to perform some homomorphic encryption to the secret before submitting and verifying.

3.2.4 Forced Delay

In this attack, the server could be running on the verifier computer and each request will be stopped and analyzed by a proxy. With this power, the verifier could get knowledge of the secret the prover possesses, as well as returning an untrusted response, such as telling that the secrets do not match, when they actually do.

This is countered when having the server running as a trusted third party in another place.

3.2.5 DDOS to the Server

Although it was not the purpose to evaluate attacks towards the server, given it was developed using Golang and outputs a binary, it can be executed without problem as a docker container which could be deployed using Kubernetes and auto-scale based on minimum CPU requirements.

If it is run in the cloud, resources to scale horizontally would be illimited (when money is not a problem).

3.3 Commands

As we are using a TCP server, to performs the operations needed, some commands were created.

To connect to the server, netcat can be used in the form of *nc host_addr port* (default port is 4321).

The commands created are simple to use and have a format corresponding to *command_name:::arg1:::arg2*, separating commands and arguments with *:::*.

3.3.1 Help

The help command returns the list of available commands with a brief description and example for each of them.

3.3.2 Submit Secret

The submit secret command, received a secret from the prover, and returns the uuid corresponding to the row of the table where it was stored, a prover key and a verifier key.

An example of the command is: *submitsecret:::secret*.

3.3.3 Verify Secret

The verify secret command, receives the secret of the verifier, the uuid and the verifier key. It returns the state of the secret, which can be a match or no match for this case. An example of the command is: *verifysecret:::secret:::uuid:::verifier_key*.

3.3.4 Get Secret State

The get secret state command, is used by the prover to get the secret state and know whether the one he possesses matches the one of the verifier.

This command receives a uuid and a prover key, and would be similar to this: *getsecretstate:::uuid:::prover_key*.

4. Conclusion

Throughout this short paper, we have gone through the basics of Zero-Knowledge Proofs, but should be enough to give the reader an understanding of its purpose, the different types and some use cases. We also went through the design of a Zero-Knowledge Proof system, which's implementation can be found in https://github.com/tuckyapps/zero-knowledge-proof.

A lot more can be found and learnt about Zero-Knowledge Proof, and the reader is encouraged to do so. Some really good papers to follow with the topic are *Succinct Non-Interactive Zero Knowledge for a von Neumann Architecture*[2] and *Non-Interactive Zero-Knowledge Proofs for Composite Statements*[3].

[2] Ben-Sasson, Eli; Chiesa, Alessandro; Tromer, Eran; Virza, Madars, 2019. *Succint Not-Interactive Zero Knowledge for a von Neumann Architecture* - https://eprint.iacr.org/2013/879.pdf.

[3] Agrawal, Shashank; Ganesh, Chaya; Mohassel, Payman, 2018. *Non-Interactive Zero-Knowledge Proofs for Composite Statements* - https://eprint.iacr.org/2018/557.pdf.

References

Agrawal, Shashank; Ganesh, Chaya; Mohassel, Payman, 2018. *Non-Interactive Zero-Knowledge Proofs for Composite Statements* - https://eprint.iacr.org/2018/557.pdf.

Anwar, Hasib, 2018. *What is ZKP? A Complete Guide to Zero Knowledge Proof* - https://101blockchains.com/zero-knowledge-proof.

Ben-Sasson, Eli; Chiesa, Alessandro; Tromer, Eran; Virza, Madars, 2019. *Succint Not-Interactive Zero Knowledge for a von Neumann Architecture* - https://eprint.iacr.org/2013/879.pdf.

Lavrenov, Dmitry, 2019. *Securing a Blockchain with a Noninteractive Zero-Knowledge Proof* - https://www.altoros.com/blog/securing-a-blockchain-with-a-noninteractive-zero-knowledge-proof.

Luciano, Adam, 2018. *ZK-STARKs – Create Verifiable Trust, even against Quantum Computers* - https://medium.com/coinmonks/zk-starks-create-verifiable-trust-even-against-quantum-computers-dd9c6a2bb13d.

Ray, Saan, 2019. *What are Zero Knowledge Proofs?* - https://towardsdatascience.com/what-are-zero-knowledge-proofs-7ef6aab955fc.

W, Oscar, 2019. *WTF is Zero-Knowledge Proof* - https://hackernoon.com/wtf-is-zero-knowledge-proof-be5b49735f27.

Zcash. *What are zk-SNARKs?* - https://z.cash/technology/zksnarks.

Zhu, Nicole, 2019. *Understanding Zero-knowledge proofs through illustrated examples* - https://blog.goodaudience.com/understanding-zero-knowledge-proofs-through-simple-examples-df673f796d99.