Zero Knowledge Proofs

Improve Tor's Reputation System

Daniel Orihuela Rodríguez

MASTER'S DEGREE FINAL PROJECT UPC

Cybersecurity Management

UPC School Year 2020-2021

DIRECTOR

Humbert Costas Pujol

SOC Data Specialist, Nestlé

THESIS SUPERVISOR

Xavier Salleras Soler

PhD Student in Applied Cryptography, ICT Department at UPF

If you think cryptography will solve your problem, either you don't understand cryptography, or you don't understand your problem.

—Roger Needham and Butler Lampson

Acknowledgements

Thanks to every person that directly or indirectly helped me engage, push and finish this thesis. My family, for supporting me with my projects and studies. Humbert Costas, for helping me throughout the project. Xavier Salleras, for bringing me the opportunity to work on that problem. He was always there to discuss my doubts and design proposals, supporting me in every moment. Without him, I would not have worked on this problem. This thesis would not exist.

Abstract

In this thesis, we study the different Zero-Knowledge Proofs types, Peer to Peer reputations systems and Tor. We did some initial research on the application of Zero-Knowledge Proofs in Tor to improve its reputation system and reduce the number of malicious nodes that want to disturb the network or deanonymize the clients. We propose an initial design that tries to solve some problems. Also, we discuss its advantages, disadvantages and problems.

Resumen

En esta tesis, estudiamos los diferentes tipos de Pruebas de Conocimiento cero, los sistemas de reputación "Peer to Peer" y Tor. Hemos hecho una investigación inicial en la aplicación de las Pruebas de conocimiento cero en el sistema de reputación de Tor, para reducir el número de nodos maliciosos que quieren empeorar el servicio o desanonimizar a otros usuarios. Proponemos un diseño inicial que intentan resolver estos problemas y discutimos sus ventajas, desventajas y problemas.

Preface

My first programming experience was a couple of years before reaching university. After building my first computer program, I knew that I wanted to do this for the rest of my life.

I went to university to learn Computer Science. During my years in college, I discovered Edward Snowden and Aaron Swartz. Since that moment, I am obsessed with my privacy on the internet. That lead me to start a master's degree in cybersecurity. Later, for my master's degree final project, I decided to contact Xavier Salleras, PhD student on Applied Cryptography. I told him that I was interested in helping with a research project. Something that could improve users privacy or that could give them back control of their data. From where this thesis was born.

Contents

List of Figures

1 Introduction

1.1 Motivation

Our preferred area of knowledge in the cybersecurity field is cryptography. Cryptography studies encryption and decryption techniques used to alter the linguistic representation of messages. The goal is that no eavesdropper can understand the content of the message.

We wanted to do some research in the Cryptography area, specifically around privacy. We contacted Xavier Salleras, PhD Student in Applied Cryptography in the UPF, to know if we could contribute to a project related to those problems. He proposed that we start investigating the use of Zero-Knowledge Proofs to improve Tor's reputation system, potentially collaborating in a scientific article with him.

This work is the fruit of the desire to improve people privacy.

1.2 Goals and Scope

In this thesis, we want to make a start of the art of various pieces of knowledge. We have Zero-Knowledge Proofs, Peer Peer reputation systems and Tor. The end goal is to improve the Tor reputation system, specifically, reduce the number of malicious exit nodes with the knowledge we have gained. Due to time constraints, we will focus on how these systems work at a high level without getting deeper into the technical aspect. Otherwise, all these systems are complex and take a considerable amount of time to understand at the low level.

The main goal is to propose a design that improves Tors' reputation system. Due to time constraints, we will focus on an initial design to mark a direction that looks promising.

Finally, we want to build a demonstration of some part of the design. That way, it will be easier to understand.

1.3 Main Problem

Tor's reputation system present some difficulties in various areas.

Identify sophisticated malicious exit nodes. Not all malicious exit nodes present signs that can be seen from the outside or replicated with different users. For example, a node that tries to deanonymize a user and nothing else cannot be detected. The inputs and outputs of that exit node are virtually equivalent to an honest exit node.

Avoid attackers from repeatedly creating malicious nodes after getting them banned. Due to the anonymity of Tor, there is no way that Tor can avoid attackers from creating new malicious exit nodes after banning their previous exit nodes.

Avoid attackers from creating unlimited malicious exit nodes. The anonymity that Tor provides makes it difficult to detect that a single attacker is creating multiple exit nodes to perform an attack.

In our proposed design, we try to solve the problems explained in the second and third paragraph.

2 State of the Art

2.1 Zero-Knowledge Proofs

Zero-Knowledge Proof (ZKP) is a cryptographic primitive that allows users to demonstrate that a statement is true while keeping some information secret. Shafi Goldwasser, Silvio Micali and Charles Rackoff conceived the first ZKPs in 1985 [1]. They involve two parties in the process:

- **Prover**: Party which wants to demonstrate that it knows a secret. But does not want to disclose it.
- **Verifier**: Party which wants to know if the prover knows the secret.

Furthermore, ZKPs must satisfy three properties:

- **Completeness**: For every valid assertion, there is a prover strategy that will make the verifier accept with high probability [2]. A prover's strategy is the set of steps that they follow to create proof.
- **Soundness**: For every invalid assertion, the verifier will reject with high probability, no matter what strategy the prover follows [2].
- **Zero-knowledge**: The proof does not convey any information to the verifier, only that the statement is true.

These properties ensure that a verifier can verify that the prover knows a witness (i.e. the secret or statement). It requires the prover and the verifier to interact several times. There is a well-known example called **Ali Baba Cave** [3], which can help visualize how a ZKP works at a high level.

There is a cave with two paths: A and B, which lead to the same door. This door can only be opened with a secret password, and the prover wants to demonstrate that he knows it to the verifier. We could follow these steps:

1. The prover selects one path and reaches the door.
2. The prover calls the verifier, who enters into the cave.
3. The verifier tells the prover to exit the cave through path A or B. If the prover knows the password, he can get out both ways.
4. Repeat steps 1-3 as many times as required. Reducing the probability that the prover was lucky, and reached the door by the path the verifier asked him to exit.

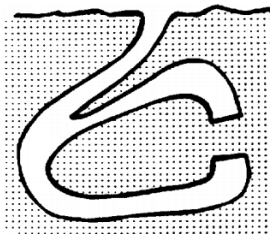

Figure 1: Ali Baba cave [3]

As we can see, the verifier can easily detect if a prover knows the witness in this example. However, challenging the prover once is not good enough. Doing it once means that the prover has a 50% chance to enter the cave by one path. Then be asked to get out by the same path, without the need to use the door. If the verifier asks the prover to take the challenge a second time, the probability of being lucky two times gets reduced to 25% $(1/2 \cdot 1/2)$. Each time the verifier asks the prover to take the challenge, the probability is halved. The probability that the prover is lucky n times is $P(n) = (1/2)^n$. For that reason, the prover and the verifier need to interact.

We can see a general ZKP flow in the figure 2.

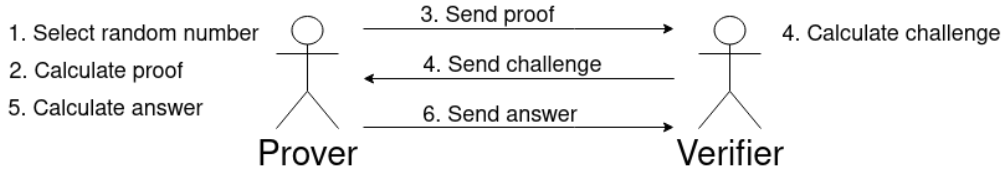

Figure 2: ZKP high level flow diagram

With the basic notions of how ZKP works, we can look at a more mathematical example. Feige, Fiat and Shamir designed in 1987 a scheme that extended zero-knowledge proofs, which at that time revealed one bit of knowledge, to truly zero-knowledge proofs [4].

1. The prover picks a random R, and sends $X = \pm R^2 \pmod{n}$.
2. The verifier sends a random boolean vector $(E_1, ..., E_\mathrm{k})$.
3. The prover sends the value $Y = R \cdot \prod_{ej=1} S_\mathrm{j} \pmod{n}$.
4. The verifier verifies that $X = \pm Y^2 \cdot \prod_{ej=1} I_\mathrm{j} \pmod{n}$.
5. Repeat steps 1 to 4 [4].

Where S_j is the secret key, and I_j is the public key. Both generated in a previous step with a key generation protocol. There exist alternatives protocols to Fiat-Shamir. E.g., Guillou-Quisquater or Schnorr's.

The mathematical formulas that we can find in different ZKP protocols, like the previous one, are called circuits. We already mentioned that a prover wants to demonstrate its knowledge of a witness. Circuits support the generation and verification of the required proofs and are composed of various components: wires, gates and constraints. Wires indicate the inputs and outputs of the different gates. A gate represents an operation, and the constraints are each of the equations resulting from the inputs, gates and outputs [5]. For example, figure 3 has 10 wires, 4 gates and 4 constraints.

The witness of the prover must comply with the circuit to generate valid proofs. Then, the verifier can check if a given proof complies with a specific circuit.

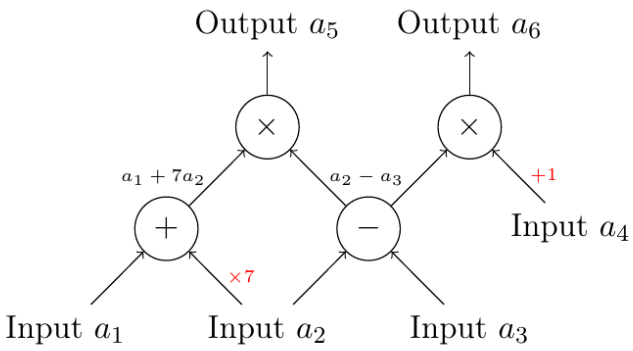

Figure 3: Example of an arithmetic circuit [6].

ZKP proofs have multiple use cases. We can prove statements on private user data (e.g., Alice has enough money to buy the house). We can perform anonymous authorization (e.g., login to a web page without revealing the password). We can make anonymous payments (e.g., pay for a good or service without disclosing identifiable information, like in Zcash). We can outsource computation (e.g., outsource a heavy computation to a third party and verify the result without redoing the execution [7].

ZKP proofs have some properties that differentiate them from other types of proofs [8]:

- **Interaction**: The prover and the verifier talk back and forth.
- **Hidden Randomization**: The verifier tosses coins that are hidden from the prover and thus unpredictable to him.
- **Computational Difficulty**: The prover embeds in his proofs the computational difficulty of some other problem (i.e. trapdoors).

In general, ZKP proofs are good to prove secrets without disclosing them. However, their main disadvantage is that the prover and the verifier must be available during the interaction. Otherwise, the verifier cannot verify the proof. That is not always desirable or acceptable. To solve this shortcoming, Non-Interactive ZKP (NIZKP) appeared in 1988 [8].

2.2 Non-Interactive Zero-Knowledge Proofs

A Non-Interactive Zero-Knowledge Proof (NIZKP) is a ZKP that does not require interactivity between the prover and the verifier. It encompasses three algorithms:

- **Key Generator**: Given a secret, generates the proving key and the verification key.
- **Prover**: Given a proving key, a common reference string and the statement to prove. Return a proof.
- **Verifier**: Given a proof, a common reference string and the verifying key. Return whether the proof is true or false.

The secret allows the prover to see in advance the challenges that the verifier will ever propose [8]. Hence, the prover can simulate the challenges of the prover. Nevertheless, this predictability has its own set of problems. Moreover, the prover and the verifier need access to a parameter called Common Reference String (CRS). This piece allows them to create and verify their proofs.

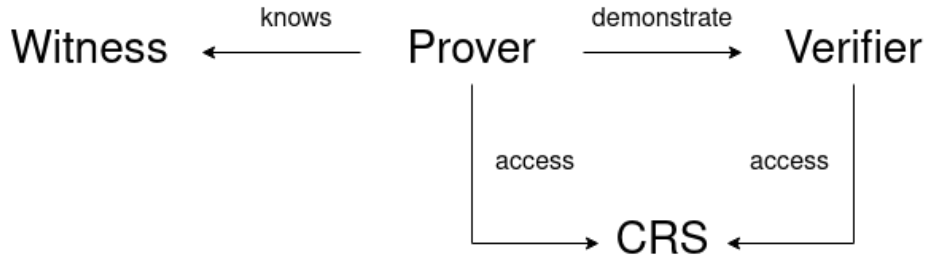

Figure 4: NIZKP high level relations scheme

To better understand NIZKP, we are going to modify the Ali Baba cave example.

First modification. We place a speaker in the cave to reproduce one of the two following sentences: "Leave the cave through path A" or "Leave the cave through path B". Each

one with a probability of 0.5 of being selected. We give the prover the remote controller of the speaker. This way, the prover has a medium to simulate the challenges.

Second modification. We place some cameras in the cave to record the actions performed by the prover. The path used to enter the cave, the path used to leave the cave, and the audio of the simulated challenge. This way, the prover can demonstrate to the verifier that he passed all the challenges.

The specific steps of this example would be:

1. The prover selects one path and reaches the door.
2. The prover pushes the button in the remote control.
3. The speaker reproduces one of the recordings. The prover has to get out of the cave through path A or B. If he knows the password, he can get out both ways.
4. Repeat steps 1-3 as many times as required.
5. Send the proof to the verifier.

We can generalize this flow into the one we can see in figure 5.

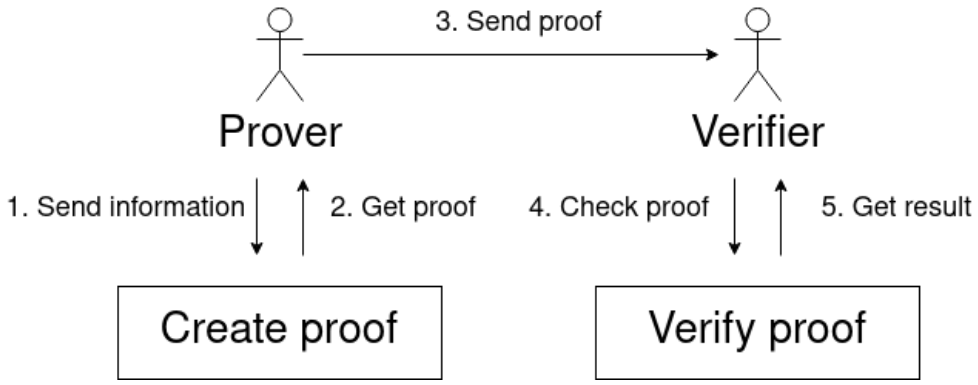

Figure 5: NIZKP high level flow diagram

Now, we can see how a full-fledged system works. Normally, NIZKP requires a trusted setup. Commonly, we do this through a secure Multi-Party Computation (MPC) [5], which helps with the generation of a secure CRS for real-world usage [9]. MPC is also in charge

of generating the proving and verifying keys. This phase is offline and only done once. On the other hand, creating and verifying the proof is done online since the prover must send it.

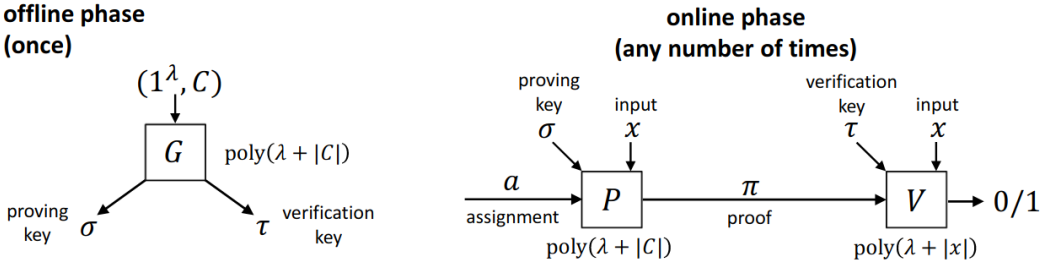

Figure 6: Key generator, prover and verifier [10]

First, we have the setup phase. In this step, we generate the proving and verifying keys with an MPC, and throw away the secret used to generate them. That way, an attacker cannot get it and generate unlimited valid proofs.

Second, we have the proving phase. The prover generates a proof with the proving key. For this example, we want to evaluate if sha3(password + salt) == hash. In that case, the prover will generate a proof with the proving key, the password, the salt, the CRS and the resulting hash. Recall that thanks to the CRS, the prover can see in advance the challenges of the verifier.

Finally, the verification phase. Any verifier can check the proof with the verification key, the hash and the CRS.

Despite the improvements in interactivity, NIZKP still had high costs (e.g., expensive key generation or hashing [11]). Hence, they were not popular. Nonetheless, NIZKP recovered its popularity in 2013 with the appearance of zkSNARKs [11].

2.2.1 zk-SNARKs

zk-SNARKs are an instance of non-interactive proofs. The letters are an acronym from:

- **Succinct**: Proofs are small (constant size), and verification is fast.
- **Non-Interactive**: Interaction between the prover and the verifier is not needed.
- **Argument of Knowledge**: The soundness property of ZKP holds against polynomially bounded provers.

Generating keys is slow. However, creating and verifying proofs is fast. zk-SNARKs can create proofs in linear time and can verify them in constant time.

zk-SNARKs are good in use cases where the speed is required, and the interaction adds too much overhead. A good example is blockchains, which have a slow communication mechanism. However, there exist other types of NIZKP apart from zk-SNARKs. E.g. Bulletproofs or zk-STARKs.

2.2.2 Bulletproofs

Bulletproofs is another instance of NIZKP which appeared in 2017 [12].

It is well suited for range proofs on committed values [12]. Thus, it allows verifying that a value is inside a range. For example, a person could prove that he has the solvency to buy a house. If the house has a price of 400.000€, they could demonstrate that they have x amount of money where 600.000€ $< x <$ 1.000.000€. Both the proof generation and verification times are linear.

Another benefit of Bulletproofs is that they do not require a trusted setup or CRS.

which are good for proving that a value is within a range. There exist other types of NIZKP like Bulletproofs [12].

2.2.3 zk-STARKs

zk-STARKs is another instance of NIZKP that appeared in 2018 [13]. Its letters stand for:

- **Succinct (Scalable)**: Proofs are small, and verification is fast. The verifier can check the proof in less time (exponentially less) than the time needed to execute the verification function. The prover needs more time to construct the proof (quasi-linearly) than the time required to verify it [13].
- **Transparent**: No trusted setup is required.
- **Argument of Knowledge**: The soundness property of ZKP holds against polynomially bounded provers.

zk-STARKs were created as an alternative for zk-SNARKs. They are more efficient and solve zk-SNARKs scalability[1] and transparency problems. Furthermore, they are post-quantum secure. However, it presents a drawback regarding zk-SNARKs. The size of the proofs is bigger, which could possess a limitation in some systems (e.g., blockchain).

zk-STARKs is the "successor" of zk-SNARKs. We can apply it to the same use cases. However, proof sizes are bigger and reducing the size is still under investigation.

2.2.4 zk-SNORKs

zk-SNORKs are another NIZKP instance, 2018 [14]. It stands for:

- **Succinct**: Proofs are small, and verification is fast.
- **Non-Interactive**: Interaction between the prover and the verifier is not needed.
- **Oecumenical (Universal)**: Any application can use the same parameters (e.g., an open-source library could include the global parameters).
- **Argument of Knowledge**: The soundness property of ZKP holds against polynomially bounded provers.

[1] zk-SNARKs vs zk-STARKs benchmark https://medium.com/coinmonks/zk-starks-create-verifiable-trust-even-against-quantum-computers-dd9c6a2bb13d

As with zk-STARKs, zk-SNORKs are an improvement over zk-SNARKs. Its main goal is to reduce the concerns about the trusted setup and its honesty due to the following problems. First, zk-SNARKs embeds a trapdoor in a relation-dependent CRS. The weakness is that if an attacker knows the trapdoor, he can use it to subvert the system's security. Second, to use zk-SNARKs the CRS needs to be computed every time the relation is changed [14]. zk-SNORKs solve these problems with updatable and universal CRS schemes.

- **Universal**: Any application can use the same parameters
- **Updatable**: Any user at any time can update the CRS.

These schemes use a relaxed CRS model in which the adversary can fully generate or contribute to the generation of the CRS. Nonetheless, if at least one of the participants is honest, the adversary will not be able to manipulate the system. However, these schemes add a time overhead[2] over the proof creation that makes them slow.

In summary, zk-SNORKs are an improvement over zk-SNARKs that alleviate some of the individuals' worries about the trusted party at the spent of larger times in the creation of proofs.

2.2.5 Multi-Party Computation

Some of the NIZKP we have talked about require a Multi-Party Computation to work. In this section, we will see its goal and how it works.

Multi-Party Computation is a subfield of cryptography that studies how different parties can compute a function over their private inputs without leaking them to the other parties. It allows computing the final result without an external party. Furthermore, it aims to ensure the following basic properties:

[2] Universal Snarks https://medium.com/aztec-protocol/the-hunting-of-the-snark-3-3-c0a6e17c6d92

- **Input privacy**: We cannot infer the private data from the messages the parties exchange to calculate the final result.
- **Correctness**: No subset of colluding adversaries can force honest parties into outputting incorrect results.

This tool helps with the creation of CRS in NIZKP protocols. Thus, alleviating concerns about single entities generating the CRS on its own. When a CRS is created, some private parameters called "toxic waste" are generated. We must destroy them for the scheme to be secure. Otherwise, an attacker with access to the toxic waste can forge valid proofs. In the NIZKP case, we chose some third parties. Each party will generate its secret and maintain the security and availability of its parameters that allows creating the CRS.

The disadvantage is that if any participants leave, we must restart the process. Therefore, the selection of participants is critical. Due to MPC limitations, the number of participants is limited and must remain online through the entire process. This leads to a bigger surface area of attacks and practical problems for the parties that must maintain custody of the hardware [9].

MPC is not a perfect solution and it has some problems. However, it fits the purposes of some NIZKP protocols.

2.2.6 NIZKP implementations comparison

In the previous sections, we discussed the different advantages and disadvantages of each NIZKP protocol, apart from their time complexities.

The following image works as a summary of the comparison between different NIZKP protocols implementations[3] (2019).

[3]Comparison of different zk-SNARKs https://zhuanlan.zhihu.com/p/40245832

ZKP name	Implementation/library	Prover Runtime	Verifier Runtime	CRS/SRS size	Proof size	Post Quantum Secure?	Universal?	Trusted Setup?	Updatable?	Crypto Assumptions
SNARKs	libsnark(in C++)	nlogn	l	n	1	NO	NO	YES	NO	Knowledge of Exponent (q-type)
Groth 2016	bellman (in Rust)	nlogn	l	n	1 (only 3 group elements)	NO	NO	YES	NO	Knowledge of Exponent (q-type)
Hyrax 2017	hyraxZK (in C++)	d(hc+clogc)+w	l+d(h+log(hc))	sqrt(w)	dlog(hc)+sqrt(w)	NO	YES	NO	NO	Discrete Log
ZK vSQL 2017	N/A	nlog(c)	l+d*polylog(n)	log(n)	d*log(c)	NO	YES	NO	YES	Knowledge of Exponent (q-type)
Ligero 2017	libiop (in C++)	nlog(n)	c*log(c)+h*log*(h)	N/A	sqrt(n)	possible, no security proof	YES	NO	NO	hash function
Bulletproofs 2017	dalek (in Rust)	nlogn	nlogn	n	logn	NO	YES	NO	NO	Discrete Log
BCG+2017	N/A	n	n	N/A	sqrt(n)	possible, no security proof	YES	NO	NO	hash function
BBC+2018	N/A	nlogn	n	sqrt(n)	sqrt(nlogn)	possible, no security proof	YES	NO	NO	SIS
STARKs 2018	libSTARK (in C++)	n*polylog(n)	polylog(n)	N/A	(logn)^2	possible, no security proof	YES	NO	NO	hash function
Aurora 2018	libiop (in C++)	nlog(n)	n	sqrt(n)	(logn)^2	possible, no security proof	YES	NO	NO	hash function
GKM+2018	N/A (the previous version of Sonic)	nlogn	l	n^2	1	NO	YES	NO	YES	Knowledge of Exponent (q-type)
Sonic 2019	sonic (in Rust)	nlogn	l+logn	n	1	NO	YES	NO	YES	AGM (algebraic group model)
Fractal 19	libiop (in C++)	nlogn	l+logn							
Libra 2019 (not facebook libra)	N/A (has implementation but not open-source)	n	dlogn	n	dlogn	NO	YES	YES	NO	Knowledge of Exponent (q-type)
PLONK 2019 (based on Sonic)	plonk (not implemented by paper authors)	nlogn	l	n	1	NO	YES	NO	YES	AGM (algebraic group model)
MARLIN 2019 (Concurrent Work of PLONK)	marlin (in Rust)	nlogn	l+logn	n	1	NO	YES	NO	YES	AGM (algebraic group model)

Figure 7: Comparison of different NIZKP implementations [3]

The table shows visually that each NIZKP protocol has its advantages and disadvantages. For that reason, it is necessary to study well the constraints of the use case we want to solve and apply the correct NIZKP. Even the specific implementation can have a huge impact.

2.3 ZKP problems

All ZKP protocols have problems. Now, we are going to summarize them.

In the previous section, we have talked about general-purpose ZKP protocols. "By general-purpose, we mean protocol design techniques that apply to arbitrary computations" [15], i.e .the ZKP protocol can create proofs for any arithmetic circuit. They require one of the following [16]:

- The proof size that is linear or super-linear in the size of the computation verifying a witness
- Prover or verifier must perform work that is super-linear in the time to verify a witness
- Complex parameter setup to be done by a trusted third party
- Rely on non-standard cryptographic assumptions
- Very high concrete overheads

At a higher level, these problems are about: space, time or trust. It is interesting to

detect which problem has each of the ZKP protocols since they limit the specific use cases or limits where they can be used effectively.

2.4 P2P reputation systems

Peer to peer (p2p) is a decentralized network architecture where all the nodes behave the same (e.g. BitTorrent). One of the main problems is identifying which nodes are good and which are malicious. P2P reputation systems are the solution.

A reputation system aims at helping unknown agents to select reliable peers to make a transaction (e.g., download a piece of a file) and score the peers in the network. It tries to provide a service to the users of p2p imposing them a minimal cost at the expense of reducing malicious nodes.

2.4.1 Reputation system design factors

A reputation system design to mitigate misbehaviours is driven by: expected behaviour of good nodes, goals of malicious nodes and technical limitations of the environment [17].

2.4.1.1 User Behaviour

The system designer must build a system that makes the users' life easier without making it an unpleasant experience. There are several user behaviours and requirements that can affect the design [17]:

- **Node churn**: Rate at which peers enter and leave the network, as well as how gracefully. Higher levels of churn require increased data replication, redundant routing paths and topology repair protocol.
- **Reliability**: Users needs some guarantees on reliability or the availability of the service.
- **Privacy**: Users may expect the data to have a certain level of security. For example, users could expect data to be encrypted and not accessible by unauthorized users.

However, some operations may require data to be unencrypted.

- **Anonymity**: Users may do not require anonymity. They may want to hide behind a pseudonym. Maybe they want to have the agent's actions be disconnected from other actions and their real person. A reputation system would be impossible under the last requirement.

We must take into account all these characteristics around users to design the reputation system. The most important ones from a security point of view are privacy and anonymity.

2.4.1.2 Threat model

Different reputation systems aim to reduce specific types of adversaries. *"Most of the existing research does not claim to handle malicious peers that bring to bear all these attacks at once. Much of the work focuses solely on independent selfish peers"* [17]. In this section, we will see different types and the techniques they use.

There exist two types of adversaries in P2P networks, selfish peers and malicious peers [17].

- **Selfish**: Peers aim to use the network and its services without contributing or using the least amount of resources possible.
- **Malicious**: Peers who want to cause harm to the system or its users. They do not care about spending large amounts of resources.

Furthermore, we can further classify adversaries depending on the techniques used to restart or improve their reputation [17].

- **Traitors**: Peers that behave correctly to build a high reputation and begin defecting. These are effective against systems that give additional privileges to users with higher reputation.

- **Collusion**: Group of peers acting with malicious purposes as a group. These are effective against systems where covert affiliations are untraceable, and the opinions of unknown peers impact one's decisions.
- **Front peers/Moles**: Peers that cooperate with other peers to increase their reputation. Then, they promote malicious peers. These are effective against systems where there is no pre-existing trust relationships and peers have only the word and actions of others in guiding their interactions.
- **Whitewhasers**: Peers that leave and join the system to lose their bad reputation.
- **Denial of service**: Peers that disrupt the network.

2.4.1.3 Environment limitations

P2P architectures, as well as their alternatives, have some limitations that we must understand.

Decentralized architecture may share the power between peers, but they are hard to design and implement. Furthermore, it may slow down the service, or it may be impossible to introduce a specific feature. On the other hand, certain functionalities are easier to implement and manage on a centralized server. However, it has some drawbacks. People may not trust the only entity in the power of its data. Also, it can become a bottleneck and a single point of failure. The third option is hybrid architectures. These are flexible and allow certain features to work in a decentralized manner while others work centralized.

To improve the design of a system, and therefore of its reputation system. We need to know the advantages and disadvantages of the different architectures.

2.4.2 Reputation system building blocks

Reputation systems require a complex design that involves several building blocks. We are going to see each of these pieces in detail.

There are three main parts [17]:

- **Information Gathering:** Collect information on the behaviour of the node.
- **Scoring and Ranking:** Score peers based on expected reliability.
- **Response:** Reward contributors and take action against malicious nodes.

2.4.2.1 Information Gathering

This component is in charge of collecting the information of the users to determine their honesty [17].

The collected information (e.g. history of actions) must be linked to a user through persistent identification. Hence, the type of identities used by peers is a concern. An identity scheme can have several properties:

- **Anonymity**: Actions cannot be traced back to a real person. Most p2p networks use user-generated random pseudonyms. Tor, for example, uses a redirection scheme called the onion routing.
- **Spoof-resistant**: Avoids adversaries from impersonating other users of the system. We can solve this with asymmetric encryption and the use of nonces for replay attacks.
- **Unforgeable**: Identities with these characteristics avoids whitewashers and Sybil attacks. In centralized versions, a trusted system entity forges them. For decentralized solutions, identifiers can be hard to produce, making it slower to whitewash or generate identities.

The information of the different peers is used to calculate its reputation, which can be done individually by each peer or by all peers sharing their experiences. We are trying to answer the question: "How can we trust a random peer?". There are various strategies to collect the required information:

- **Personal experience**: Use local information to know whether or not the peer is good (e.g., check if we already made a transaction with him and if the transaction was successful).
- **External trusted sources**: Ask users outside the network (e.g., friends, coworkers or business relationships).
- **One-hop trusted peers**: Ask trusted users (in the network) the peer has met before.
- **Multi-hop trusted peers**: Also known as *transitive trust*, is a variant of the previous approach. We ask trusted peers (e.g., neighbours), as well as trusted peers of trusted peers, building a chain of trust.
- **Global system**: (Centralized). The system collects information about all peers from all peers.

Regardless of the strategy used to collect information, we cannot enforce honest and accurate reporting on transactions outcomes by all peers. For this reason, most reputation systems assume the majority of the users are honest and that collecting information from a large number of peers will result in a relatively accurate assessment of a peer's behaviour.

Dealing with strangers is another key situation. When new peers join the network, we do not have a transaction history of that node to reason about their intentions. Therefore, a *stranger policy* is required to handle these situations. Two simple strategies are: optimistically trust all strangers and pessimistically refuse to interact with strangers. Other more complex strategies exist, for example, the "stranger adaptive" strategy, which merges the information of all the first transaction ever done in the network. That, along with a "generosity" metric based on recent stranger interactions, allow peers to calculate the probability of being cheated by a stranger [18] [19] [20].

2.4.2.2 Scoring and Ranking

The second component of a reputation system calculates the reputation of the users and

builds a ranking [17].

The first component collected information from a user. This information provides us with various statistics to use as inputs. Preferably, we would use the amount a peer co-operates and defects. Nonetheless, adversaries may not openly defect. We can get this information from malicious nodes that provide a bad service, but we cannot get this information from selfish peers since they defect silently. For example, in file-sharing networks, free riders will not listen to queries or share files. That way, peers cannot know how often other peers ignore request on purpose. We could solve this by looking at the rate a peer contributes to the network. Moreover, we should give more weight to defection information. That is a clear indication of malicious nodes. More than contributions to the network are an indication of good nodes.

As a result of the previous inputs, we obtain the reputation, which can be in different formats. A few examples could be a binary value, a scaled integer or a continuous scale. The format will depend on the application. A practice that some p2p networks follow is to maintain multiple reputations for each user to tackle the different types of adversaries. Thus, making it more flexible. For example, TRELLIS uses two ratings. One for the likelihood that a given peer will cooperate on a transaction, and the other for the probability that they recommend a malicious peer.

The last step is to select the correct peer. We will choose the peer at our disposal with the highest-ranking score. That could lead us to a malicious peer if our pool selection is composed only of malicious nodes. We can solve this using a threshold to reject all peers with a lower reputation.

2.4.2.3 Response

The last part of the design is the incentives and punishments that reputation systems

can use to motivate the good behaviour of nodes [17].

Incentive models aim to stop selfish peers and make them contribute, compensating their effort with some benefit. These can also mitigate malicious nodes if access to services requires the peer to contribute first. Examples of incentives could be:

- **Speed**: Faster download speeds or reduced response latency.
- **Quality**: Provide content at different levels of quality.
- **Quantity**: Give the user more access to resources (e.g. more number of permitted downloads).
- **Money**: Micropayments in exchange for contributing to the network.

On the other hand, punishment models reduce the number of malicious peers. Depending on the number of times a given peer is detected to misbehave and the strictness level we want to apply, we can retaliate in several ways:

- Warn other users
- Disconnect neighbours from the adversary
- Eject the adversary from the network
- Kick the adversary from the network for some time

Reputation systems are a cornerstone of P2P networks. They must be thorough designed to reduce the number of adversaries of certain types; without losing sight of the service goal.

2.5 Tor

Tor is an anonymity network designed to give us privacy while searching on the internet. It defends us from, for example, surveillance, fingerprinting and censorship.

2.5.1 Tor Design

The Tor network is complex and has a lot of low-level details that make it an anonymous network. However, in this section, we will present the high-level ideas that make it work.

2.5.1.1 Virtual Circuits

Tor uses the concept of Virtual Circuit to reference groups of nodes in the network that allow two machines to communicate through an encrypted tunnel. These circuits are made of three types of nodes [21]:

- **Entry nodes**: Communicate clients with the circuit. It knows the origin.
- **Relays**: Connects the entry and the exit node. It cannot see any sensitive information.
- **Exit nodes**: Links the circuit with the destination. It knows the receiver and the message.

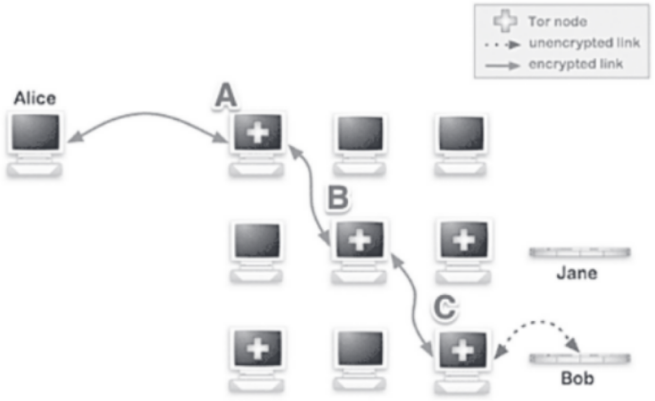

Figure 8: Tor Virtual Circuit [22]

To build a circuit, the user negotiates a symmetric key with each router following the Diffie-Hellman key exchange protocol. Figure 9 shows how a two-hop circuit is build. First, Alice negotiates a symmetric key with OR1. Then, Alice negotiates a symmetric key with OR2 using OR1 as a proxy or anonymizer.

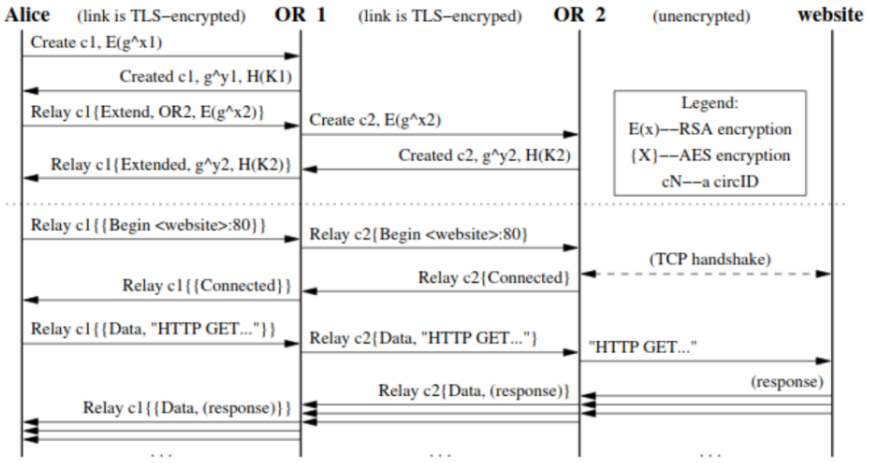

Figure 9: Comparison of different NIZKP implementations [21]

2.5.1.2 Directory Authorities

These are well-known nodes from a small group in the Tor network. They collect information from other nodes to know the state of the network. Clients can at any moment fetch network information from them, apart from the list of onion routers.

Tor has nine nodes of this type that are in the power of third trusted parties. These machines are critical. All the Tor network clients will build their circuits and know the state of the network from the directories. For that reason, they use a consensus protocol to agree on the different information they provide. This consensus is executed every hour and needs to be accepted by more than half of the directory servers to be valid.

2.5.1.3 Onion Services

Onion services, also known as hidden services, are anonymous network services (e.g. website) that only can be accessed through the Tor network.

These services provide end to end encryption and end to end authentication, besides hiding the IP of the server and the persons' identity behind it. People can use them for

different purposes. For example, write blogs with controversial opinions in a censored state without fear of punishment, or a black market like Silk Road [4].

Tor uses two types of nodes to establish the communication between a client and an onion service, Introductory Points and Rendezvous Nodes. When a person creates an onion service, it will build virtual circuits to random nodes that will work as introductory points. The job of these nodes is to listen to client requests. So, when a client wants to access an onion service, it will connect to an introductory point and send him the direction to rendezvous points. This node acts as an intermediary and will maintain the connection between the client and the onion service. For that reason, the IP of the onion service is kept hidden. Furthermore, this avoids Denial of Service attacks against onion services.

2.5.2 Reputation system

Tor's reputation system is straightforward. As we said, it has some nodes called Directory Servers, which must reach a consensus. In that step, they decide which routers must go off the list and which ones can remain.

Decreasing or detecting bad exit nodes is complex and an open area of research. However, Tor has a couple of mechanisms to ease that problem [5].

First, Tor uses exitmap [6]. This scanner for exit relays executes some tasks (e.g. fetching a web page, uploading a file, connecting through SSH) to monitor their reliability and honesty. Tor runs this tool periodically, and they will use that information on the consensus.

Second, Tor users can report [7] malicious exit nodes. Then, a subset of vigilant Tor developers will try to reproduce the problem and try to solve it with the exit node operator.

[4] SilkRoad black market https://en.wikipedia.org/wiki/Silk_Road_(marketplace)

[5] Bad exit flag on exit node https://tor.stackexchange.com/questions/253/what-specifically-causes-an-exit-to-get-a-bad-exit-flag

[6] Scanner for Tor exit relays https://github.com/NullHypothesis/exitmap

[7] Report exit node https://blog.torproject.org/how-report-bad-relays

If the operator does not answer or help, they will mark the exit node with a "BadExit" flag, preventing clients from using it in the future. In severe cases, they can remove it from the consensus.

These methods help reduce malicious exit nodes but cannot detect sophisticated attackers. In the next section, we will see some of these attacks.

2.5.3 Attacks

Tor can suffer various attacks [8]. In this thesis, we will focus on the attacks from an exit node point of view [21].

2.5.3.1 End-to-end confirmation attack

An attacker can deanonymize a user if they control the entry node and the exit node to the server. In that situation, an attacker can correlate data entering a circuit with the data that is going out. Hence, knowing which user is behind the communication. Attackers can use different pieces of information, even though the most used methods take into account the timing of the packets, the number of packets send and received, or the sizes of the packets [23].

2.5.3.2 Man in the middle attack (unauthenticated protocols)

An attacker can impersonate a server if the client is not communicating over a protocol with end-to-end authentication (e.g. HTTP). In that case, the exit node would be able to intercept all the information in plain text and send whatever response they decide to the client. For example, this attack was used to spy on European governmental agencies [9] and was not discovered until one year later.

[8] Tor attacks summary https://github.com/Attacks-on-Tor/Attacks-on-Tor

[9] Onionduke malware https://blogs.salleurl.edu/es/networking-and-internet-technologies/onionduke-el-apt-a-traves-de-tor

2.5.3.3 Denial of Service attack

An attacker could modify the source code of the exit node to drop packets. For example, they could drop all the traffic, even though this is easily detected, and Tor will remove this exit node from their router list. However, there are more subtle ways of Denial of Service attacks. They could, for example, drop all traffic to a specific web, server or port.

Sometimes this attack is a side-effect of botnets using Tor [24, 22]. If a botnet using Tor has enough victims, they can potentially slow and bottleneck the traffic in the Tor network.

2.5.3.4 Sybil attack

An attacker can create an undefined number of Tor relays to gain increase its influence in the network. With a high enough number of relays into its power, some attacks are easier to execute. E.g. correlation attacks, since having more relays increases the probability that a client builds circuits with an entry and exit node under the attacker's control. Furthermore, these nodes can degrade the user experience and impair their anonymity.

These attack may seem highly improbable to occur, nonetheless in may of this year, an article [10] was posted on the internet of an attacker that had over 25% of the exit nodes in its power. In that specific case, the attacker was performing an SSL strip to downgrade HTTPS to HTTP to change the receipt address of bitcoin transactions with his own.

[10] Over 25% of Tor exit relays spied users https://thehackernews.com/2021/05/over-25-of-tor-exit-relays-are-spying.html

3 Solution

To solve some of the problems and attacks mentioned in sections 1.3 and 2.5.3, we propose an initial design that is far from finished but shows a direction that may be promising despite some unsolved problems that it has.

3.1 Summary

Our design involves mainly two components: a decentralized reputation system and NIZKPs. Optionally, we will explain how enclave technologies could improve the design.

First, we have a decentralized reputation system based on [25]. This solution will help us, among other things, to control how many exit nodes a given person can have, besides preventing them from creating nodes without restrictions.

Second, we have NIZKPs. These proofs will help clients verify that a given exit node is honest. Therefore, increasing their confidence in the fact that the circuit is not compromised.

Finally, we can use enclave technologies (e.g. IntelSGX) in various situations to make sure that the code has not been modified. For example, the client could check the exit node code has not been altered or the other way around.

In the following section, we will discuss the design piece by piece and see the advantages and disadvantages.

3.2 Proposed design

The core of our design is made of two components, as we explained in the summary. A decentralized reputation system and NIZKPs.

3.2.1 Decentralized reputation system

We based this piece of the system on [25]. We will call that system "Tassos Decentralization". In this section, we will explain it.

Tassos Decentralization is a privacy-preserving reputation scheme for collaborative systems (e.g. P2P) where peers can represent themselves with different pseudonyms. It introduces the "Registrar", which is in charge of deduplicating identities. That is, detecting if two given pseudonyms represent the same unique identifier of a person.

The registrar is the central piece of the decentralization system. We need it to register new users, avoiding the creation of multiple credentials for the same person. An implementation called Candid [26] exists, which is a set of decentralized nodes that handles decentralized identities. CanDID is not a registrar, but a registrar with extra functionalities. CanDID allows importing identities securely from existing systems. Hence, Alice can generate new credentials based on his Social Security Administration profile, attesting to her Social Security Number. Furthermore, it facilitates the generation of pseudonyms. Alice can create unlimited pseudonyms with her credentials, all of them linked to the Social Security Number. That allows her to perform transactions as if she was different persons but allowing the system to deduplicate identities when needed.

Regarding Tor, we can use the Tassos Decentralization to prevent Sybil attacks and whitewashers. To get these advantages, we need to modify the Tor protocol.

Add a new exit node:

1. User creates new credentials if needed.
2. User creates an exit node under a pseudonym.
3. This is put into a queue.
4. In the following consensus, the directory authorities perform some steps.

(a) Deduplicates identities and check if the user surpassed the limit of exit nodes. If true, we reject the exit node.

(b) Check if the reputation is above the threshold.

(c) The exit node is accepted and will get published in the following consensus if directory authorities agree.

With this changes, we can make sure that a user never has a high percentage of the exit nodes of the network. Therefore, we are preventing Sybil attacks. The limit of exit nodes could be a static value (e.g. threshold = 10), or it could be a dynamic value based on the current number of exit nodes (e.g. limit = 5% of exit nodes of the network).

As for the Tor reputation system, we keep using exitmap plus user reports to decide if we remove an exit node. However, we add a new step. When an exit node is known to be malicious, it will harm the reputation of its owner. Then, while executing the consensus, directory authorities will need to check if any of the exit nodes owner reputations have fallen under the threshold and remove them from the router list. Also, the user will not be able to publish new exit nodes in the future. With that, we avoid whitewashers.

However, this piece is far from perfect and has some disadvantages and limitations.

We are not preventing governmental organizations from creating an unlimited number of exit nodes. This kind of organization has enough resources to create fake persons to create new exit nodes. Furthermore, criminal organizations with enough resources or power to create fake persons or use stolen user data will have no problem creating exit nodes without limit.

Our design does not contemplate the possibility of resetting a reputation to 0. That could be interesting if, for example, a criminal organization created an exit node with your data. Earned a bad reputation. And now you want to build legit exit nodes.

Furthermore, the consensus needs to check if any of the exit nodes have an owner with

a bad reputation. That fact could make the consensus protocol slow and infeasible. We did not have time enough how this impacts the time of the protocol. However, any person willing to keep researching in this area needs to take this into account.

3.2.2 NIZKP

Now that we avoid Sybil and whitewashers, we can discuss how to build circuits with honest exit nodes.

The idea here is to use NIZKP so that clients can verify the honesty of a given exit node when building a circuit. That way, the client will never create a circuit with a malicious exit node. To do that, we will use zk-SNARKs due to their fast verification time. The directory authorities will perform some steps after accepting a new exit node.

1. Directory authorities accept new exit node.
2. Create proving and verifying keys.
3. Send the proving key to the exit node.
4. Publish the verifier (this can be in a blockchain like Ethereum or by the directory authorities).

In our design, the verifiers will be published in the directory authorities. That means that these machines will execute the code necessary to verify proofs. That could be a problem if we did not trust the directory authorities since they could trick the client into thinking that a given proof is valid when it is not. However, since we already trust these machines there is no problem.

The flow of creating a circuit will require some changes too. See figure 10.

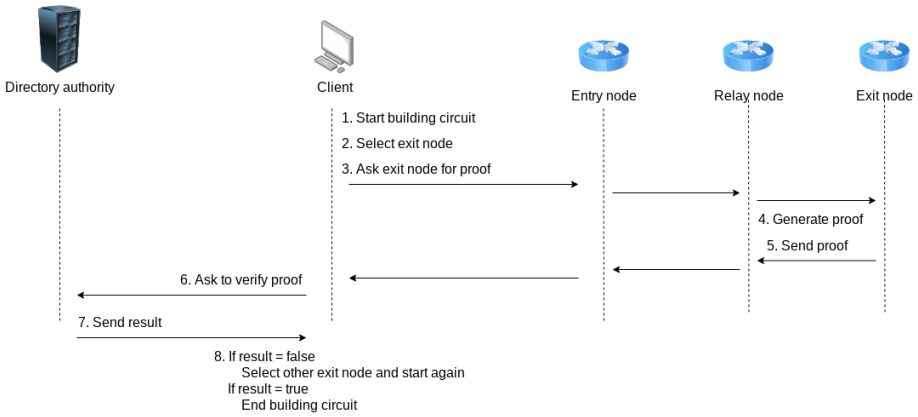

Figure 10: Circuit creation

That is the new general flow that we propose to create virtual circuits. Nonetheless, we need to understand how the exit node generates the proof (see figure 11 and 12). To make sure that the exit node cannot manipulate it and trick the user.

When generating a proof, the exit node needs to get some information from the directory authority. The exit node will first send its pseudonym to a directory authority, who will forward it to the registrar. As we know, the registrar is only in charge of deduplicating identities. However, for our design, it also needs to be able to retrieve the reputation. At that point, the directory authority will receive the reputation. Finally, it will send to the exit node the data to create the proof. The reputation (r), the minimum reputation (r_{min}), an expiration timestamp (t_{exp}), a signature of the reputation and the expiration timestamp concatenated (s), and the public key (pk). With this information, the exit node can generate a valid proof. The signature prevents the modification of the reputation or the expiration timestamp. Furthermore, the verifier uses the expiration time to check that the proof is still valid. Otherwise, the exit node could use the highest reputation he got since its creation. Once the proof is created, the exit node will send it to the client, along with t_{exp}. Then, the client will forward both pieces to the directory authority for verification.

The next image shows the circuit we designed to create and verify the proofs. It has

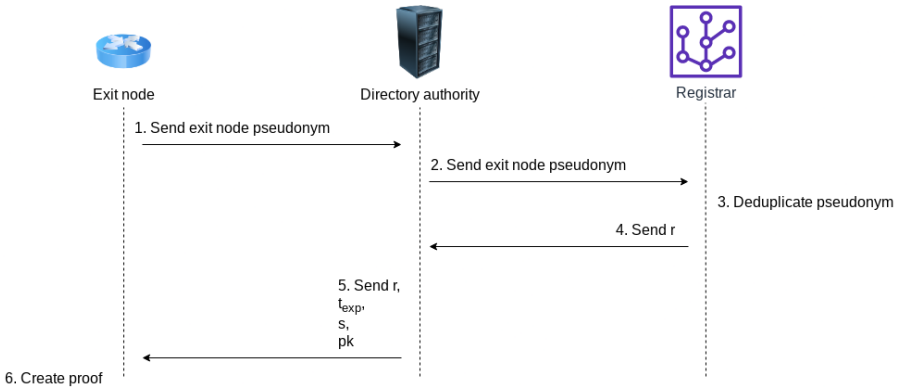

Figure 11: Proof generation

some additional parameters to the ones we saw in the previous flow. The message (msg), which is the reputation and expiration timestamp concatenated (e.g. reputation = 5, t_{exp} = 1234, message = 51234); and the t_{length}, the length of the timestamp. As private parameters, we have r, s and msg. The others are public. That way, the exit node can prove the knowledge of its reputation, the message and its signature.

The circuit verifies a couple of things.

1. msg contains r and t_{exp}. The signed message must have the correct reputation and expiration timestamp. This is done checking the following expression: $message - t_{exp} - (reputation * 10^{t_{length}}) = 0$. This only works if the reputation is an integer.
2. $r > r_{min}$. The reputation is above the minimum specified, which is 0 and hardcoded in the circuit.
3. $t_{exp} > t_{now}$. The proof did not expire.
4. s is the signed msg and can be verified with pk. The signature must be verified with the public key, and it must match the introduced message.

The output of the circuit is an AND of all the previous checks. If all of them are okay, the created proof is valid, false otherwise.

The verifier will check a proof against that same circuit. To do this, it will need the t_{length}, t_{exp}, t_{now}, and pk. The t_{exp} will be received from the client along with the proof, the t_{length} can be calculated from t_{exp}, the t_{now} can also be calculated and the pk is the public key of the consensus (i.e. the directory authority has access to it). If the exit node did not modify any piece of data, it will return 1 to the client, 0 otherwise.

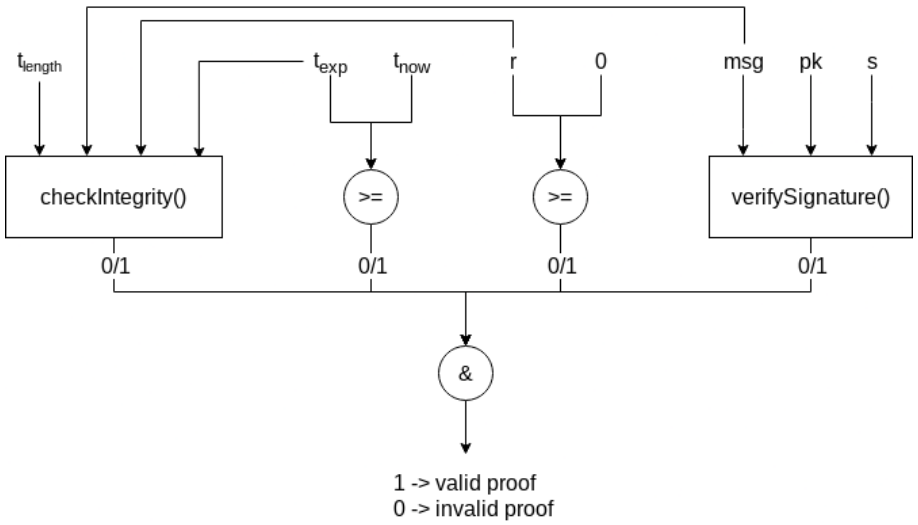

Figure 12: ZKP circuit

For the signature verification, we use EdDSA Poseidon. A zk-SNARK friendly hash function that requires fewer constraints and thus is faster compared to others like the EdDSA Pedersen hash. Specifically, our design has 4238 constraints. This impacts the time required to generate a proof, which is around 0.660s. On the other hand, the time to verify the proof is not tied to the number of constraints. It is done in constant time and needs around 0.245s. Therefore, we are not adding an excessive overload to the verifiers, which are the directory authorities. Even though further research on a realistic environment is required.

With these changes, any client can create circuits without fear of using a malicious exit node.

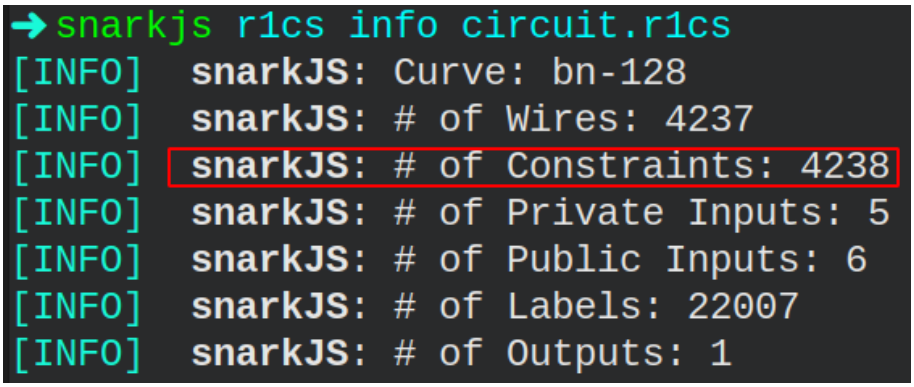

Figure 13: Number of constraints

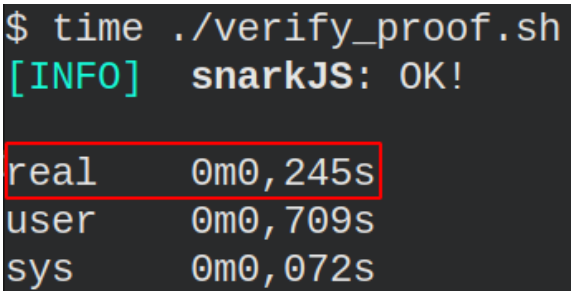

Figure 14: Verification time

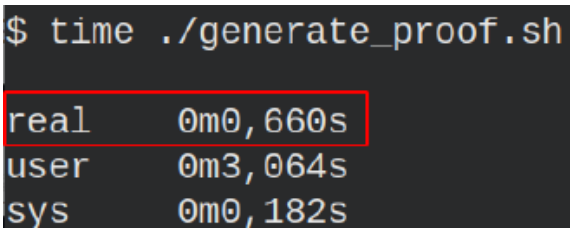

Figure 15: Proof generation time

The main disadvantage in this part is that any client can copy a valid proof sent by an exit node. That means that anyone could create a malicious exit node that sends valid proof. We believe this should not be a problem. An attacker with a bad reputation has no way of introducing a new exit node in the network. Thus, there is no situation in which an attacker could take advantage of that vulnerability. That is an important fact that should be taken into consideration in future research since it could be a real problem.

3.2.3 Trusted execution environment

Optionally, we could improve the security of the Tor network using SGX-Tor [27].

SGX-Tor is a trusted execution environment that uses IntelSGX technology to improve the overall security of Tor. Our design is compatible with SGX-Tor, and we could use it to allow the client to verify that a given exit node source code has not been modified. Also, a relay of the Tor network could verify if the code of a client has been changed. In both cases, this would reduce the probability that a client is using Tor over a compromised software. Therefore, improving their chances of being anonymous.

The disadvantage is that SGX-Tor requires IntelSGX to work. That is inconvenient for all the clients that do not have the proper hardware. For that reason, this part is optional. However, it would be interesting to use the SGX-Tor only on the exit node.

4 Results

We proposed a solution to improve Tor's reputation system that includes a Decentralized reputation system and Non-Interactive Zero-Knowledge Proofs, which could be improved with SGX-Tor. It has its disadvantages, and we did not have time to make a full implementation and test its viability. For that reason, we cannot provide technical results.

Further research is needed to implement a minimum viability mechanism for each part and check how they fit together and their feasibility. However, we believe that we set a good direction.

5 Future Research

There are multiple lines to keep researching on:

- How to solve the disadvantages and flaws of the proposed design.
- Make a minimum implementation involving all the pieces.
- Use of other NIZKP.
- Find new vulnerabilities and attacks that affect the design.
- Use of blockchain and zk-rollups for the verification of proofs.
- Use new pieces of information to calculate the reputation of users.

6 Conclusion

In this work, we have presented a design to improve Tor's reputation system using Zero-Knowledge Proofs. Although we did not have time to implement and tests its feasibility, we think it is a small step in a good direction. Our design adds resistance against Sybil attacks and whitewashers.

References

[1] S. Goldwasser, S. Micali, and C. Rackoff, "The knowledge complexity of interactive proof-systems," in *Proceedings of the Seventeenth Annual ACM Symposium on Theory of Computing*, STOC '85, (New York, NY, USA), p. 291–304, Association for Computing Machinery, 1985.

[2] S. P. Vadhan, "A study of statistical zero-knowledge proofs," 1999.

[3] J.-J. Quisquater, M. Quisquater, M. Quisquater, M. Quisquater, L. Guillou, M. Guillou, G. Guillou, A. Guillou, G. Guillou, S. Guillou, and T. Berson, "How to explain zero-knowledge protocols to your children," pp. 628–631, 08 1989.

[4] U. Feige, A. Fiat, and A. Shamir, "Zero knowledge proofs of identity," pp. 210–217, 01 1987.

[5] X. Salleras and V. Daza, "Sans: Self-sovereign authentication for network slices," *Security and Communication Networks*, vol. 2020, p. 1–8, Nov 2020.

[6] H. Mayer, "zk-snark explained: Basic principles," 2016. https://blog.coinfabrik.com/wp-content/uploads/2018/12/zkSNARKexplained.pdf.

[7] M. Petkus, "Why and how zk-snark works," *CoRR*, vol. abs/1906.07221, 2019.

[8] P. F. Manuel Blum and S. Micali, "Non-interactive zero-knowledge and its applications," *In Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing, STOC '88, New York, NY, USA. ACM*, pp. 103–112, 1988.

[9] S. Bowe, A. Gabizon, and I. Miers, "Scalable multi-party computation for zk-snark parameters in the random beacon model." Cryptology ePrint Archive, Report 2017/1050, 2017. https://eprint.iacr.org/2017/1050.

[10] E. Ben-Sasson, A. Chiesa, D. Genkin, E. Tromer, and M. Virza, "Snarks for c: Verifying program executions succinctly and in zero knowledge (extended version)," in

Advances in Cryptology – CRYPTO 2013 (R. Canetti and J. A. Garay, eds.), (Berlin, Heidelberg), pp. 90–108, Springer Berlin Heidelberg, 2013.

[11] E. Ben-Sasson, A. Chiesa, E. Tromer, and M. Virza, "Succinct non-interactive zero knowledge for a von neumann architecture." Cryptology ePrint Archive, Report 2013/879, 2013. https://eprint.iacr.org/2013/879.

[12] B. Bünz, J. Bootle, D. Boneh, A. Poelstra, P. Wuille, and G. Maxwell, "Bulletproofs: Short proofs for confidential transactions and more." Cryptology ePrint Archive, Report 2017/1066, 2017. https://eprint.iacr.org/2017/1066.

[13] E. Ben-Sasson, I. Bentov, Y. Horesh, and M. Riabzev, "Scalable, transparent, and post-quantum secure computational integrity." Cryptology ePrint Archive, Report 2018/046, 2018. https://eprint.iacr.org/2018/046.

[14] J. Groth, M. Kohlweiss, M. Maller, S. Meiklejohn, and I. Miers, "Updatable and universal common reference strings with applications to zk-snarks." Cryptology ePrint Archive, Report 2018/280, 2018. https://eprint.iacr.org/2018/280.

[15] J. Thaler, "Proofs, arguments, and zero-knowledge," 2021. http://people.cs.georgetown.edu/jthaler/ProofsArgsAndZK.pdf.

[16] R. S. Wahby, I. Tzialla, abhi shelat, J. Thaler, and M. Walfish, "Doubly-efficient zksnarks without trusted setup." Cryptology ePrint Archive, Report 2017/1132, 2017. https://eprint.iacr.org/2017/1132.

[17] S. Marti and H. Garcia-Molina, "Taxonomy of trust: Categorizing p2p reputation systems," *Computer Networks*, vol. 50, no. 4, pp. 472–484, 2006. Management in Peer-to-Peer Systems.

[18] K. Lai, M. Feldman, I. Stoica, and J. Chuang, "Incentives for cooperation in peer-to-peer networks," 06 2003.

[19] M. Feldman, K. Lai, I. Stoica, and J. Chuang, "Robust incentive techniques for peer-to-peer networks," in *Proceedings of the 5th ACM Conference on Electronic Commerce*, EC '04, (New York, NY, USA), p. 102–111, Association for Computing Machinery, 2004.

[20] M. Feldman, C. Papadimitriou, J. Chuang, and I. Stoica, "Free-riding and whitewashing in peer-to-peer systems," *Selected Areas in Communications, IEEE Journal on*, vol. 24, pp. 1010 – 1019, 06 2006.

[21] R. Dingledine, N. Mathewson, and P. Syverson, "Tor: The second-generation onion router," in *Proceedings of the 13th Conference on USENIX Security Symposium - Volume 13*, SSYM'04, (USA), p. 21, USENIX Association, 2004.

[22] M. Casenove and A. Miraglia, "Botnet over tor: The illusion of hiding," *6th International Conference on Cyber Conflict*, 2014.

[23] J. Fajfer, *Correlation Attacks on TOR*. Bachelor's thesis, Czech Technical University in Prague, Faculty of Information Technology, 2018.

[24] D. Orihuela, *Malware and spy systems hybrid botnet over Tor*. Bachelor's thesis, UPF's Polytechnic School, Information and Communications Technology, 2020.

[25] T. Dimitriou, "Decentralized reputation." Cryptology ePrint Archive, Report 2020/761, 2020. https://eprint.iacr.org/2020/761.

[26] D. Maram, H. Malvai, F. Zhang, N. Jean-Louis, A. Frolov, T. Kell, T. Lobban, C. Moy, A. Juels, and A. Miller, "Candid: Can-do decentralized identity with legacy compatibility, sybil-resistance, and accountability." Cryptology ePrint Archive, Report 2020/934, 2020. https://eprint.iacr.org/2020/934.

[27] S. Kim, J. Han, J. Ha, T. Kim, and D. Han, "Enhancing security and privacy of tor's ecosystem by using trusted execution environments," in *14th USENIX Symposium on*

Networked Systems Design and Implementation (NSDI 17), (Boston, MA), pp. 145–161, USENIX Association, Mar. 2017.