Merkle Tree Based Zero Knowledge Proofs

Swara Lawande, Prasad Hadkar, Srikanth Rao Parcha

University of Florida, Gainesville

ABSTRACT

With the explosion of the internet, more of the once done offline activities have transitioned to online spaces. One of the most ubiquitous applications of the internet is Blockchain. It is touted as one of the most revolutionary inventions of recent history and finds its applications in a gamut of domains such as online banking, e-voting, and file-sharing. However, it comes with some glaringly apparent disadvantages. One such problem is the set membership verification or authentication problem - this problem aims to check if the given entity is a part of a set or not. Using Zero-Knowledge Proofs (ZKP) is one of the more novel and upcoming ways to achieve set membership verification. In this paper, we have developed a proof-of-concept implementation of Zero-Knowledge proofs using Merkle trees in Java programming language. Our implementation solves the set membership verification problem while ensuring the confidentiality of user identity. We have presented the explanation of a myriad of algorithms used in our implementation, followed by results of running various benchmarked tests on the parameters involved in the construction of ZKPs. We have provided our deductions from the obtained results and the performance of our implementation. We finally conclude this paper by summarizing our implementation and gathering our opinions on any future implementations.

BACKGROUND

1) Cryptographic hashing function: A cryptographic hashing function is a mathematical function that alphanumeric value into a seemingly random compressed value. A hashing function takes an arbitrary length input and produces a fixed-length output. The three main characteristics of a cryptographic hashing function are preimage resistance, collision resistance, and collision resistance.

2) Merkle tree: A Merkle tree is a hash tree-based data structure. In a Merkle tree, the main aim is to find a route from a selected node to the root, also called the Merkle root. A Merkle tree is constructed in a top-down manner. A Merkle tree is constructed by hashing sibling nodes together. This process continues until there is a single node remains. This node is called the Merkle root. Every non-leaf node in the Merkle tree is a hash of the nodes that came before it, in other words, its children.

3) Zero-knowledge proofs: Zero-knowledge proofs allow the user to 'prove' another user/party that a claim/statement is true without conveying any additional information apart from the fact that the claim/statement is, in fact, accurate. In a zero-knowledge proof, roving a claim/statement requires the proof to be in possession of some secret information, then no one apart from the user with the secret information, i.e., the prover, can prove the claim/statement.

SYSTEM ARCHITECTURE

The three main elements of our system are as follows:

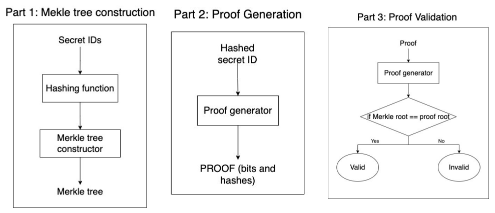
1. Merkle Tree Construction:

The first phase of the architecture is the construction of the Merkle tree. As the name suggests, this phase is involved in creating the Merkle tree. Here, the secret IDs of the participants are fed to a hashing function (example-SHA256). The hashed version of the IDs are then passed onto the Merkle tree constructor which then generates the Merkle tree.

2. Generation of Proof of Inclusion: This is the second phase of our implementation. In this phase, a participant generates a unique proof of inclusion. This proof allows the participant to convince the "validator" that the participant is indeed a valid member of the set. In this phase, the hashed secret IDs of the participants are fed into the proof generator. The proof generator generates a unique proof for the user. This proof consists of bits and hashes.

3. Proof Validation:

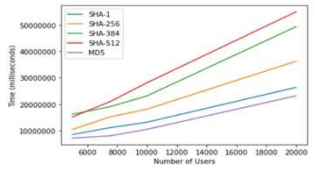
The final phase of our implementation is proof validation. As the name suggests, the validator validates the proof generated by the participant. As mentioned earlier, this proof aims to prove the membership validity of a participant. In this phase, the main aim to match the Merkle root generated by hashed secret ID of the participant and the root generated by the authority/validator. If they match, set membership verification is achieved.



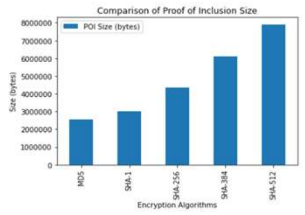
The system architecture is represented by the above flowcharts. We developed a proof-of-concept implementation based on the above system using Java. Below image represents the concept of Zero-Knowledge Proof.

EXPERIMENTS AND RESULTS

We conducted experiments to test the performance of our proof-of-concept implementation of Zero-Knowledge-Proof.

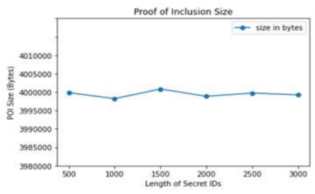


From the above graph we observe that as the number of users (secret hashed ids) increases, the execution time of our system also increases. Additionally, the encryption time is positively correlated with the hash output bit size. Consequently, MD-5 takes least amount of time, whereas,SHA-512 takes most time. Similarly, MD-5 consumes the least amount of memory and SHA-512 needs the most in terms of bytes.



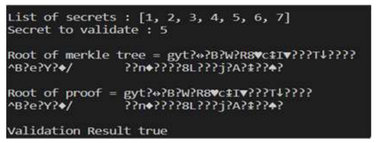
We conclude that when the application security is critical, one can go for a more secure algorithm such as SHA-512 and compromise performance, whereas when performance is priority, one can go for a less secure algorithm such as SHA-256.

From the below image, that increasing the length of secret id does not affect the size of the proof of inclusion. This is because, the proof consists of list of bits and hashes which will consume approximately the same amount of space(in terms of bytes) for the same encryption algorithm.



Further, we conducted experiments on the different scenarios for validation of Proof of Inclusion as follows:

1. Valid Proof : hashed secret present in Merkle Tree



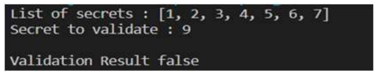
```
List of secrets : [1, 2, 3, 4, 5, 6, 7]
Secret to validate : 5

Root of merkle tree = gyt?↔?B?W?R8♥c‡I▼???T↓????
^B?e?Y?◆/        ??n◆????8L???j?A?‡??♠?

Root of proof = gyt?↔?B?W?R8♥c‡I▼???T↓????
^B?e?Y?◆/        ??n◆????8L???j?A?‡??♠?

Validation Result true
```

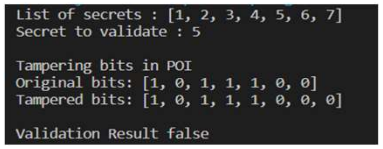
2. Invalid Proof : hashed secret not in Merkle Tree.



```
List of secrets : [1, 2, 3, 4, 5, 6, 7]
Secret to validate : 9

Validation Result false
```

3. Invalid Proof : Adversary generates invalid proof of inclusion (for this we tampered with a valid POI)



```
List of secrets : [1, 2, 3, 4, 5, 6, 7]
Secret to validate : 5

Tampering bits in POI
Original bits: [1, 0, 1, 1, 1, 0, 0]
Tampered bits: [1, 0, 1, 1, 1, 0, 0, 0]

Validation Result false
```

APPLICATION DOMAIN

- E-voting
- Authentication
- Finance
- Data-Sharing
- Medical Data
- Scientific Research

CONCLUSIONS

In this paper, we provided an implementation of Zero-Knowledge Proof to solve the set-membership or authentication problem in immutable distributed systems. We devised algorithms and presented a system design to generate and validate ZKPs using the Merkle Tree data structure. Further, we presented our proof-of-concept implementation of the system in Java. We benchmark the performance of this implementation by performing experiments based on various parameters of the ZKP. Finally, we presented our deductions and conclusion based on the results obtained from these experiments. We conclude that Zero-Knowledge Proofs is a novel and practical approach for authenticating users while preserving their privacy and confidentiality of their identity.

REFERENCES

- Truc Nguyen and My T. Thai (2022). zVote: A Blockchain-based Privacy-preserving Platform for Remote E-voting, University of Florida
- Harry Alford (2018), What Is Zero-Knowledge Proof And Why You Should Care, medium.com
- I. Miers, C. Garman, M. Green and A. D. Rubin, "Zerocoin: Anonymous Distributed E-Cash from Bitcoin," 2013 IEEE Symposium on Security and Privacy, 2013, pp. 397-411, doi: 10.1109/SP.2013.34.