Zero-Knowledge Inclusion Proofs

Student

Roman Bögli

Advisor
Prof. Dr. Mitra Purandare

Subject Area
Computer Science

Project Partner
IBM Research, Rüschlikon, Zürich

Introduction: The fields of computer science and cryptography opened the door to the most recent type of medium of exchange, namely cryptocurrencies which elevates the quality of money traits even more. These traits include unforgeability, verifiability, portability, divisibility, fungibility, durability, and — exclusively in the case of digital money — programmability. Prominent cryptocurrencies, such as Bitcoin for example, additionally introduce the trait of being censorship-resistant through the use of Distributed Ledger Technology (DLT) powered by proof-of-work consensus mechanisms.

Problem: A disadvantage of DLT systems concerns the risk of loss of user privacy. This stems from the requirement for transactions to remain auditable for all, resulting in their transparent recording within publicly available blockchain data structures. Consequently, observers may track individual token movements, negatively impacting user privacy.

Also, the security model of digital money usually breaks in the scenario of compromised, stolen, or lost keying material used for digital signatures. Although there are strategies to minimize this risk, there is no such thing as absolute security. This becomes especially delicate in systems where a money-issuing authority uses keying material to sign or mint new tokens, allowing a successful attacker to mimic minting and thereby counterfeit money.

Lastly, the concern of being quantum-resistant urges adopting new algorithms grounded on Post-Quantum Cryptography (PQC). Therefore, a future-oriented digital money solution must account for this development and upgrade the employed cryptographic primitives to quantum-safe versions.

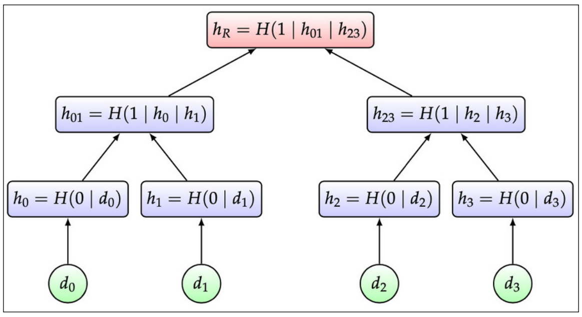
In summary, the problem encompasses the question of how to verifiably prove a specific token's validity in a trustless and quantum-resistant setting without revealing which specific token it concerns in order to preserve user privacy.

Approach / Technology: This work describes a solution to this problem that employs Zero-Knowledge Proof (ZKP) protocols to convince a verifying party of a token's inclusion in a public set of valid tokens, encoded as Merkle tree data structure, without revealing the token specifically.

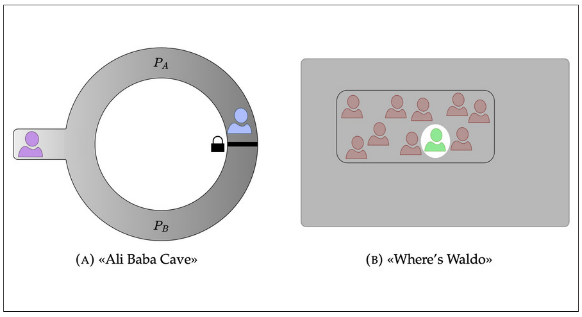
Preliminary, the fundamental concepts of cryptographic hash functions, commitment schemes, and Merkle trees are explained, followed by a general introduction to ZKP protocols.

In order to contribute to the prospective goal of implementing a minimum viable product of such a proof system, this work additionally provides an overview of identified implementation facilitating automation frameworks, which forms the basis for future endeavors in this topic.

Merkle Tree
Own presentment



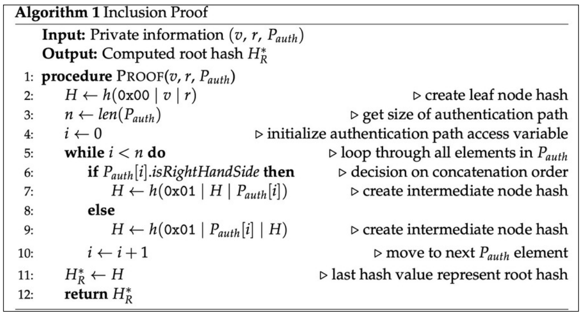
Interactive (A) and Non-Interactive (B) ZKP Analogy
Own presentment



(A) «Ali Baba Cave»

(B) «Where's Waldo»

Inclusion Proof Algorithm
Own presentment



Algorithm 1 Inclusion Proof

```
    Input: Private information $(v, r, P_{auth})$
    Output: Computed root hash $H_R^*$
procedure PROOF$(v, r, P_{auth})$
    $H \leftarrow h(\texttt{0x00} \mid v \mid r)$                        ▷ create leaf node hash
    $n \leftarrow len(P_{auth})$                        ▷ get size of authentication path
    $i \leftarrow 0$                        ▷ initialize authentication path access variable
    while $i < n$ do                        ▷ loop through all elements in $P_{auth}$
        if $P_{auth}[i].isRightHandSide$ then                        ▷ decision on concatenation order
            $H \leftarrow h(\texttt{0x01} \mid H \mid P_{auth}[i])$                        ▷ create intermediate node hash
        else
            $H \leftarrow h(\texttt{0x01} \mid P_{auth}[i] \mid H)$                        ▷ create intermediate node hash
        $i \leftarrow i + 1$                        ▷ move to next $P_{auth}$ element
    $H_R^* \leftarrow H$                        ▷ last hash value represent root hash
    return $H_R^*$
```