Private Airdrop System with Zero-Knowledge

Tal Derei

Zero-Knowledge University (ZKU)

April 2022

1 Motivation

To motivate this construction, we must map out the current landscape of public crypto airdrops. To encourage participation in their protocol, blockchain-based projects typically distribute free ERC-20 tokens to early adopters and active participants in their community. This system of allocating ERC-20 tokens for governance purposes (i.e. token-voting mechanisms) is flawed. Participating in these airdrop events requires revealing your public wallet address (i.e. public key), and thereby doxxing your financial history associated with your public identity. Linking your public metamask address to a web3 platform reveals everything about yourself, since your public address is a gateway to publicly available transaction history on the blockchain. Consider, for example, if a protocol decides to blacklist your address from receiving an airdrop based on your DAO voting history? This warranted a system enabling users to claim airdrops completely anonymously, without revealing their public keys.

2 Technical Specifications

2.1 Merkle-Tree Construction

1. Construct a merkle tree of commitments, *commitment* = hash(key, secret), eligible for the airdrop. These commitments are stored in a MongoDB database.
2. Recursively hash the merkle tree and publish merkle root hash on-chain (~ vector commitment).
3. Everytime a user enters an input (i.e. the key/secret pair), it generates a witness that serves as an input to the proof generation algorithm.
4. User submits merkle path proof for verification and claims airdrop, without revealing which commitment is associated to their public key.

2.2 Circuits: Proof Generation and Verification

The circuits (*withdraw.circom* and *merkleTree.circom*) are referenced from a simplified version of the Tornado Cash: https://github.com/chnejohnson/simple-tornado. The circuits (2^16 constraints) will be compiled and proven with Groth16, requiring a trusted setup (i.e. Power of Tau MPC Ceremony). A verifier contract handling the proof verification on-chain is then automatically generated by circom/SnarkyJS.

2.3 On-Chain Solidity Contract

The airdrop contract is responsible for **[1]** calling the compiled verification contract to verify the proof, **[2]** checking against the nullifier set to prevent double-withdrawals, **[3]** redeeming the airdrop to address.

2.4 Long-Term Roadmap and Use-Cases

The system can be extended, for example, to NFT merchandise drops where you prove you own an NFT without revealing your public identity/address. Another extension is incorporating a simple reputation service meeting certain on-chain requirements (i.e. checking whether users have more than **10** ONE in their wallet). This requires a ZK-Oracle or some external trusted third-party entity verifying the validity of the on-chain information before creating the proof.

2.5 Competitive Landscape

A competing project includes a more complicated infrastructure that requires two-wallets and ECDSA signature schemes: https://github.com/nalinbhardwaj/stealthdrop. Additionally, they lack reputation services. My construction would be a simpler implementation from scratch w/ reputation service long-term.

2.5 Product Roadmap

Version-1: Allow users to verify they're in a merkle-tree of commitments and retrieve an airdrop anonymously. ***Version-2:*** Layering a reputation service described above in section *2.4* is a longer-term goal that will take time.

2.7 Tools and Resources

- Circom 2 (ZK-SNARK Compiler)
- Snarkjs (Typescript/Javascript framework for zk-SNARKs)
- Tornado Cash

- MongoDB
- React / Next.js / Ether.js / Hardhat