Zero-Knowledge University (ZKU):

Assignment 5

Tal Derei

04 April 2022

Course Registration Email: Tad222@lehigh.edu

Discord Username: Margulus#4273

Github (Personal Repistory): https://github.com/TalDerei

Github (ZK Research): https://github.com/TalDerei/Zero-Knowledge

Q1. Shielding the TX

1. Ethereum-to-Harmony Bridge:

Cross-asset private bridging involves the use of mixer-like mechanisms and light-clients. Ethereum and Harmony are smart-contract-based protocols, A and B respectively, and a private bridge model for transfering 100,000 UST tokens from A to B involves maintaining a bridge-contract S on both, S_A and S_B respectively. These bridge contracts, S_A and S_B, have both **[1]** bridging functionality (i.e. mechanisms to SETUP/DEPOSIT/WITHDRAW algorithms / APIs), and **[2]** light-client functionality (SETUP/ADD-HEADER/BRIDGE-STATE algorithms / APIs). Light-clients allow both the bridge contract S and external

users to verify proofs of transactions in order to track the consensus state. In that regard, we can break up the state maintained by the bridge contract S into: 1. bridge-related state and 2. light-client related state.

[**1**]Bridge-related state:

merkle tree I_N

[**2**] Light-client related state:

merkle tree headers B[]

And these states are changed by running algorithms that change state (i.e. like verifying a proof of proof of proof-of-work α for new block header B_i, or updating merkle tree).

Then the overall protocol can be constructed (facilitated by light-clients) as follows:

1. circuit C for merkle tree membership verification **2.** S deployed on both A and B, S_A and S_B respectively, with initial state

3. user interacts with S_A by calling DEPOSIT(), which appends merkle leaf z into merkle tree I_A

4. user generates proof π_{I_A} of deposit by calling MT.PROVE()

5. user submits proof to withdraw tokens by calling MT.VERIFY-SNARK()

6. user withdraws on S_B

Storage concerns dominant every facet of blockchain technology, including cross-chain bridging. The storage complexity of the bridge contract is on the order of the light-client storage complexity. And since the merkle headers stored by the client are larger than

the merkle roots and nullifiers stored by the bridge contract, the storage complexity of the bridge is bounded above by the storage complexity of the light client (i.e. O(S) = O(light-client) respectively). Some solutions include blockchain optimizations to reduce the storage requirements of the client (i.e. stateless clients).

The proposal above references "Trustless, privacy-preserving blockchain bridges" by Drew Stone. I'll propose three **alternative**, practical solutions below:

> [**1**] Maintaining a public cross-chain bridge connected by two mixers, where the privacy is preserved in both deposit/withdrawal phases.
>
> [**2**] Shielded notes (i.e. Azect's zk.money protocol) using L2 zk-rollup contract to bridge funds from L1 to L2 anonymously.
>
> [**3**] Aztec Connect: the first private bridge allowing anyone to anonymously interact with DeFi protocols on Layer 1 from Layer 2 (enabling private DeFi). The "Bridge Contract" is a simple 50-100 line interface allowing Aztec's L2 zkRollup to interact with a Layer 1 smart contract with 100x cost savings. This would enable private swaps, private staking for yield, private lending, private voting in DAOs, etc.

2. Wormhole Bridge:

The $320m wormhole hack revealed the fragility of many of these cross-chain bridges. There are two attack vectors we need to consider: [1] fake withdrawals, and [2] double withdrawals. In regards to [1], due to the soundness requirement of our system (if verifier V accepts a proof π generated by prover P, P knows the witness of circuit with probability 1 - negl(probability)), we can assume proof forgery has a nelgible probabilty. In regards to [2], double withdrawals by implementing a nullifier set. If a user with-

draws, the nullifier set will be updated and indicate the user cannot withdraw again. Specifically in regards to wormhole's vunerability, I would increase the number of validators (i.e. Guardian accounts) needed to sign off in order to approve a transaction.

Q2. Aztec

1. Aztec Notes:

Aztec is an open source zero-knowledge protocol for building privacy on Ethereum using PLONK, a recursive zk-SNARK proof system powering their zk-zk-rollup. Aztec is focusing on a generic solution to achieve confidential transactions and cross-asset settlements using [1] aztec notes, and [2] zero-knowledge proofs. Let's break these into seperate componenets:

[1] Aztec notes follow a UTXO (i.e. unspent transactions) model similiar to that of Bitcoin. In the same way an account has a balance, a note has an owner. And a user's balance of an asset is the sum of all valid notes their address owns. Aztec notes are represented by ERC-1723 standard (creating a confidential asset standard with common programming interface), and have public (on-chain) components and private components:

On-Chain Data

1. ETH address of owner
2. Aztec note public key
3. Aztec metadata

Private Data

1. Note value
2. Viewing key (construct zk proofs and decrypt notes)
3. Spending key (authorize transfers)

At a high level, you submit ETH to the Aztec Rollup contract on L1, and then "shield" the assets on L2 by creating notes to make them private. These shielded assets are the Aztec notes referenced above.

Aztec has a family of zero-knowledge proofs, sharing a common reference string. These modular proofs can perform different business logic (i.e. interest / dividend proof), and prove something about Aztec notes. The first example is a ***"Join-Split Proof"*** which allows a set of input notes to be joined or split into a set of output notes. The smart contract destroys input notes in it's state registers and creates output notes. A Join-Split Proof then essentially proves that the sum of the input notes = the sum of the output notes (even though the notes are encrypted). For example, let's say you wanted to build a zk-dapp, and perform a private swap/transfer between your zk-dapp and another zk-dapp (e.g. loan originator, or exchange). A ***"Swap Proof"*** (Bilateral Swap) is an atomic swap of notes between two encrypted digital assets (e.g. zk-dapp and an exchange). A ***"Send Proof"*** generates a proof that verifies a private transaction was executed. A ***"Mint / Burn Proof"*** generates a proof that verifies the creation or destruction of some asset. A ***"Dividend Proof"*** generates a proof that verifies an Aztec note is a percentage of another Aztec note.

Aztec further pioneered the concept of ***"Efficient Range Proofs"***, combining existing range proofs with polynomial commitments. The previous zk-proofs are efficient to verify because of these range proofs underlying them. Perform homomorphic encryption being able to evaluate mathematical statements over encrypted variables. One of the problems, the arithmetic your evaluating is over integers that are modulus a large prime number → Need to prove that the Aztec note does and wrap around the elliptic curve modulus Proof no notes have a negative value which would allow you to print in-

finite money because you can construct a proof we're you have input notes of -2 until now there's some to zero. This zero knowledge proof proves the integer in the note is constrained by a certain range.

Additionally, Aztec has a Cryptography Engine, ACE (ERC-1723) - shared suite of zk validator smart contracts for zk proofs. ACE accept zk proofs and spits out transfer instructions.

Q3. Webb Protocol

1. and 2. Anchor and VAnchor Contracts:

Mixer commitment = nullifier, secret

Anchor commitment = chainID, nullifier, secret

Vachnor commitment = chainID, amount, public key, blinding

The Anchor commitment is an extension of a "mixer" commitment, with the additional chainID field to specify the chain. The Vanchor commitment maintains a blinding term to add entropy to the poseidon hash in order to avoid leaking public key. A new commitment scheme might be commitment = hash(key, secret), allowing you to identify your commitment is part of a merkle tree without revealing which commitment corresponds with your public key. This would eliminate the need for extra noise / entropy parameters.

3. VAnchor UTXO:

UTXO structure in the VAnchor:

```
{
    chainID
    amount
    public key
    blinding
}
```

VAnchor utilizes a join-split transaction model with two inputs and outputs, respectively. Then compute the commitment and nullifier, and then verify the correctness of those values.

Q4. Thinking In ZK

Aztec: Given your privacy-preserving solution, it's fair to say that it's not practical to make private versions of DeFi protocols, but instead it's more practical to privately interact with DeFi protocols using layer-2. Do you think that will always be the case / paradigm? Additionally, what Ethereum Virtual Machine inefficiencies would you change in order to make your zk-zk-rollup more efficient and performant?

Q5. Final Project Proposal

[SEE NEXT PAGE]

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