Zero Knowledge University (Assignment-5)

This is the fifth assignment report in the fifth week of Zero-Knowledge University (ZKU) cohort. In the fifth week of the cohort, ZKU teaches the student about Shielding the TX, AZTEC and Webb protocol. The assignment consist of 5 (five) questions which are answered in this report.

Based on my attempt to complete the assignment, I write this report by grouping the question into 5 (five) headings below. All of my work can be found at **https://github.com/Rizary/zero-knowlegde-university-report/tree/zku-assignment-5** branch.

I. Shielding the TX

You have been asked to present a mechanism that will allow a user to privately and securely bridge 100,000 UST tokens from Ethereum to Harmony. Draft a write-up explaining the protocol to be built to cater for this need, highlighting the challenges to be faced and potential resolution to them.

A. Private and secure bridge tokens protocol for cross-chain

Securing bridge token protocol for cross-chain blockchain is needed so that user is allowed to privately and securely bridge their token. The example for that is if user wants to bridge 100,000 UST tokens from Ethereum to Harmony, they want to make sure it is private and secure.

In this case, I came up with the design as follows:

1. We will have the Smart Contract (S) in blockchain A (S_A) and blockchain B (S_B) that have Setup, Add_Header, Add_Bridge_State, Deposit, and Withdraw functionality.

2. This smart contract will have two state that is related to bridge activity. First, it will have a merkle tree instance with fixed height. Second, it will have a type of arrays which will be used to store the serial numbers (sn) from when a user can verifies a zero-knowledge proof on S_A and S_B.

3. A user of blockchain A (called User1) decides to transfer some token $T to blockchain B.

4. User then generates random value and interact with S_A and deposits a fixed sized deposit into S_A after that, S appends a merkle leaf in the merkle tree instance on blockchain A using ADD function.

5. User then generate proof of their deposit using function PROVE. Then after some delay, user can then withdraw their tokens.

There will be some challenges such as double withdraw from the system. The solution is to add some delay D to the process on withdrawals. Another challenges will be on storage complexity. The solution is merkle-izing the set of nullifiers and verifiably tracking the latest root of all reported nullifiers.

The next challenges will be a Vampire Attack, which is to suck liquidity onto new blockchains. The solution is to give user a reward for providing liquidity on each side of the bridge in order to facilitate private transfers. Users are got rewarded to lock up tokens on both sides, causing liquidity to distribute across the bridge.

B. Key mechanism to preventing crypto bridge hack

Once a user u deposits value into the tree, it is imperative that they cannot double-withdraw from the system, a problem similar to the double spending problem. There are some solutions to that such as:

1. A user u can only double spend if the nullifiers sn used have not been relayed to the bridged blockchain.

2. When a user submits the same nullifier to both instances of S, the system will find out about the respective action in time D, which is enough to cancel the withdrawal when detected.

3. When a relayer updates the nullifier set, we require that each new nullifier added is checked for existence on the contract already to find these duplicates mentioned above.

C. Smart contract for user request and circuits for privacy

II. AZTEC Protocol

A. AZTEC Note concept and how it is used to privately represent various assets on a blockchain.

AZTEC is an open source layer 2 network that uses zkSNARK proofs to provide privacy and scaling via zkRollup service. Like any other zkRollup network, AZTEC submit rollups every 1 minute. The protocol, designed for use within blockchain protocols that support Turing-complete general-purpose computation.

The protocol is utilizes a commitment scheme that enables the efficient verification of range proofs. It can also be used to create confidential representations of existing digital assets. One of the feature in AZTEC is called the **AZTEC NOTE**. Note is an encrypted representation of abstract value. It is an output of the AZTEC commitment function and is comprised of a tuple of elliptic curve commitments and three scalars: a viewing key, a spending key, and a message.

There are various proofs utilized by the AZTEC protocol such as:

1. *Range proofs*

This is used to prove that an AZTEC note is greater than another AZTEC note or vice versa. There are two type of range proofs, **private range** and **public range**. The private range proofs is useful like public but also adding the ability to build identity and group membership schemes. The public range proof is useful for proving that ownership of an asset post trade is below a regulatory maximum.

2. *Swap proofs*

Swap proofs is a bilateral swap proof that allows an atomic swap of two notes to take place. A validated proof, proves that the makers bid note is equal to the takers ask note and the makers ask note is equal to the takers bid note.

3. Dividen proofs

Dividen proofs is a proof that allows the prover to prove that the input note is equal to an output note multiplied by a ratio. This is useful for paying interest from an asset.

4. Join-split proofs

The Join Split proof is a proof that allows a set of input notes to be joined or split into a set of output notes. This proof is used best to combine note values into a larger note, or split a note into multiple notes with different owners. This proof ensures that the sum of the input notes is equal to the sum of the output notes

B. How AZTEC can be used to create private loan application on the blockchain (benefit and challenges)

Please see my comment in my gists here: https://gist.github.com/Rizary/5a84ae086744fed6157d16ec66e35c4f

III. Webb protocol

A. Different between commitments made to Anchor and VAnchor contracts

The commitment to Anchor is handled differently with VAnchor. In **Anchor**, The system requires users to both deposit a fixed denomination of ERC20 assets into the smart contract and insert a commitment into the underlying merkle tree of the form: commitment = Poseidon(destinationChainId, nullifier, secret). Commitments adhering to different hash functions and formats will invalidate any attempt at withdrawal. Using the preimage of the commitment, users can generate a zkSNARK proof that the deposit is located in one-of-many anchor merkle trees and that the commitment's destination chain id matches the underlying chain id of the anchor where the withdrawal is taking place. The chain id opcode is leveraged to prevent any tampering of this data.

In **VAnchor**, The system requires users to create and deposit UTXOs for the supported ERC20 asset into the smart contract and insert a commitment into the underlying merkle tree of the form: commitment = Poseidon(chainID, amount, pubKey, blinding). The hash input is the UTXO data. All deposits/withdrawals are unified under a common `transact` function which requires a zkSNARK proof that the UTXO commitments are well-formed (i.e. that the deposit amount matches the sum of new UTXOs' amounts). It is in the form of:

1. UTXO = { destinationChainID, amount, pubkey, blinding }
2. commitment = Poseidon(destinationChainID, amount, pubKey, blinding)
3. nullifier = Poseidon(commitment, merklePath, sign(privKey, commitment, merklePath))

B. UTXO scheme works on the VAnchor contract

The UTXO scheme in VAnchor is made from destinationChainID, amount, public key, and blinding. By using the UTXO, users can generate a zkSNARK proof that the UTXO is located in one-of-many VAnchor merkle trees and that the commitment's destination chain id matches the underlying chain id of the VAnchor where the transaction is taking place.

C. How the relayer works for deposit part of the tornado contract

The relayer for deposit of the tornado contract happened in https://github.com/webb-tools/relayer/blob/main/src/events_watcher/tornado_leaves_watcher.rs#L77-L122. The function works as follow:

1. The relayer implements the EventWatcher.
2. It first set up some middleware (http), contract (using tornado wrapper contract), events and store (using sled store).
3. It contains handle_event function that is matching the event whether it is **DepositFilter** or **WithdrawalFilter**.
4. If it is a **DepositFilter**, then it sets the commitment, lead_index, value of pair leaf_index and commitment hash. It also set the chain_id by getting tho contract chain id. Last, the function insert to the store (which is a tree structure) both leaves (using chain_id, contract address, and list of value) and last deposit block number (using chaind_id, contract address and the log of the block number).
5. If it is a **WithdrawalFilter**, it doesn’t do anything and just return the function.

IV. Thinking in ZK

The thought on questions for AZTEC and Webb

A. AZTEC

1. Is there any possibility to implement Decentralized Identity (DID) in the AZTEC workflow?

B. Webb

1. What is the reason to choose SNARK over STARK?
2. Is there any possibility to implement Decentralized Identity (DID) in the Webb workflow?