ZKU Assignment 5

Name: Shyam Patel
Discord: L_guy#4297
Email: shyampkira@gmail.com
GitHub Repo: [Link]

Question 1.Shielding the TX

In week 2 we learnt about one of the three use cases of ZK, Privacy, and how zero-knowledge can be used to shield the identity of both parties in a transaction. This week we will be focusing on privacy within the context of bridges. This is especially useful as we move into a cross-chain era in blockchain and users require the same level of anonymity when bridging their tokens across different chains

1. 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.

There are two aspects to deal with: **privacy** and **security**

- **Privacy:** This can be achieved with a mixer: the user sends their funds to a mixer before withdrawing them to an anonymous wallet, where bridging can be enacted from. Mixers are generally well audited and battle tested at this point, so security here is a given.
- **Security:** The main issue for a bridge would be security since most unidirectional-bridge are an off-chain system, it inherits none of the native security a blockchain achieves. Other bridges like Wormhole and Multichain are bidirectional, or two-way, meaning you can freely convert assets to and from blockchains.

Flow for kind-of secure bridge would be something like this:

- A user deposits 100k UST tokens on the Ethereum blockchain onto a bridge. This bridge shall contain following components:
  - A smart contract on Ethereum that allows users to deposit 100K UST.
  - A smart contract on Harmony that users can withdraw funds by giving a proof.
  - A circuit that can generate proof of a deposit and probably withdraw.
  - Web-app:
    - Retrieve merkle roots stored in contract on Ethereum and submit to Harmony Contract
    - Move fund from Ethereum to Harmony
- WebApp allows the user to construct a zk-proof which enables users to prove that the deposit is actually made. This can be achieved through a state transition proof, just like validating a block on a zk-rollup, or stateless clients.

- User takes their proof and posts it on the destination blockchain to claim their bridged tokens.
- Verifier contract checks the proof from the user and returns a bool based on the input. This amounts to the Verifier contract being a stateless client for the origin blockchain.

2. In February 2022, a hack on a popular crypto bridge led to the second biggest crypto heist where $320m was lost. Following the technical details behind the hack, it is very clear that bridge smart contracts need to be airtight to prevent scrupulous individuals from taking advantage of them. Briefly explain key mechanisms you will put in place in your interoperable private bridge (specifically the withdrawal methods) to prevent a similar attack (Double withdrawal and fake withdrawal).

The concern for the above mechanism is cross-chain double spending. Performing L1-to-L1 transfers, as one L1 may fork while the bridge request is taking place, thus essentially creating "double spending" of money.
The other risks such as tamper information or collusion (in case of guardian nodes) are abstracted away by zero-knowledge proofs. Smart contract security is a big challenge so it's better to get a thorough security audit and also have it peer reviewed. Other than that, learning from the past attacks is important, in order to prevent it.
The wormhole hack can be described by a logic fault where the code was written keeping in mind == and && are interchangable, however `false == false` evaluates to `true`, but `false && false` evaluates to `false`. Thus, make sure that checks are written properly because using checks this way is erroneous. Another key aspect to a secure bridge, as shown by the axie attack, was that trustlessness in validators is hard to achieve. It seems that for now, unless you are a zk bridge, you should use KYC for this to avoid malicious actors.

Q2 Aztec

AZTEC protocol utilizes a set of zero-knowledge proofs to define a confidential transaction protocol, to shield both native assets and assets that conform with certain standards (e.g. ERC20) on a Turing-complete general-purpose computation.

1. Briefly explain the concept of AZTEC Note and how the notes are used to privately represent various assets on a blockchain. There are various proofs utilized by the AZTEC protocol including range proofs, swap proofs, dividend proofs and join-split proofs. Highlight the features of these proofs and the roles they play in creating confidential transactions on the blockchain

Aztec follows a UTXO model like Bitcoin. The core of any Aztec transaction is a Note. An Aztec note is an encrypted representation of abstract value, basically encrypted values, i.e. cypher messages. How this abstract representation maps to real quantities is a higher-level detail of digital assets that utilize the protocol.
Aztec Note consists of a tuple of elliptic curve commitments and three scalars:

- a viewing key
- a spending key, and
- a message.

Knowledge of the viewing key allows the note to be decrypted, revealing the message.
Knowledge of the viewing key can be used to create valid join-split zero knowledge proofs.
These proofs are then signed by the spending key.
The state of notes is managed by a Note Registry for any given asset.

Currently AZTEC supports various proofs. Some of them are:

- **Join Split (Transfer):** The Join Split proof allows a set of input notes to be joined or split into a set of output notes. Usually this is used 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.
- **Bilateral Swap (Trade):** The bilateral swap proof allows an atomic swap of two notes to take place. This is useful for trading two assets e.g., fiat and a loan/bond/security. A validated proof proves that the maker’s bid note is equal to the taker’s ask note and the maker's ask note is equal to the taker's bid note.
- **Dividend Proof:** This proof 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.
- **Private Range:** This is used to prove that an AZTEC note is greater than another AZTEC note or vice versa. This is useful for proving that ownership of an asset post trade is below a regulatory maximum. It can also be used to build identity and group membership schemes.

- **Public Range:** Like the private range proof. This is used to prove that an AZTEC note is greater than a public integer or vice versa. This is useful for proving that ownership of an asset post trade is below a regulatory maximum.

2. Using the loan application as a reference point, briefly explain how AZTEC can be used to create a private loan application on the blockchain highlighting the benefits and challenges. In the Loan Application, explain the Loan.sol and LoanDapp.sol file (comment inline)

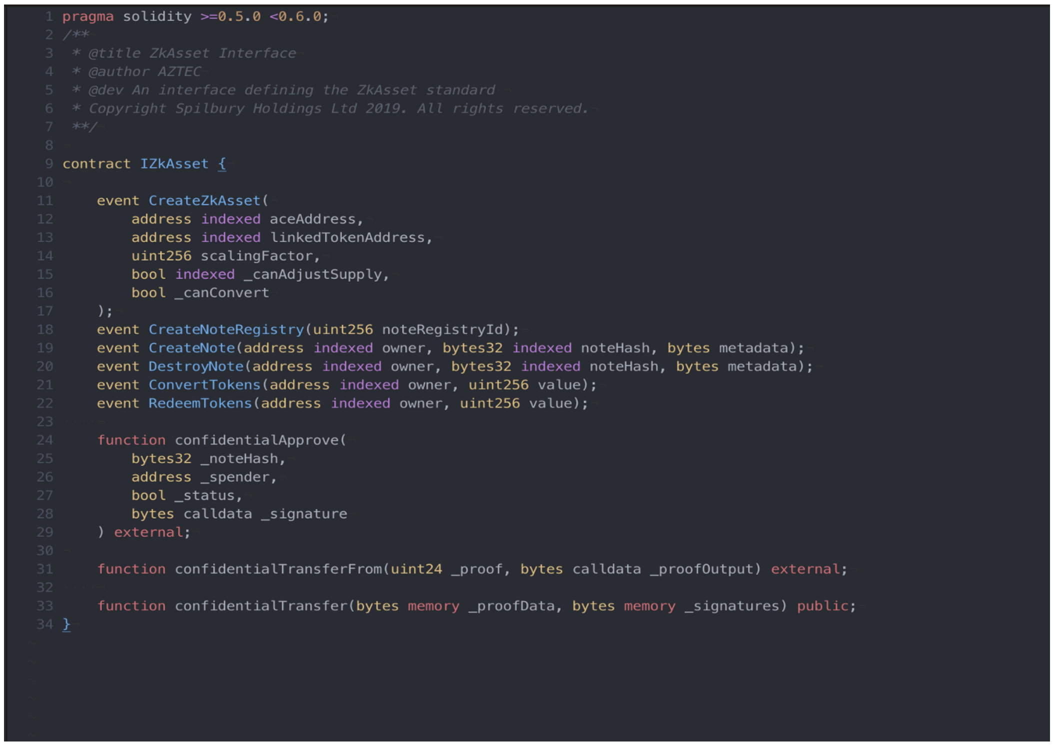
Privacy, Anonymity and Confidentiality are often used when discussing zero-knowledge systems. It is important to define the meaning of each and address how AZTEC handles them.

- **Privacy:** all aspects of a transaction remain hidden from the public or third parties.
- **Confidentiality:** the inputs and outputs of a transaction are hidden from the public but the transaction parties remain public.
- **Anonymity:** the inputs and outputs of a transaction are public but the transaction graph is obscured from one transaction to the next, preventing the identification of the transaction parties.

AZTEC enables confidential transactions out of the box. The inputs and outputs of any transactions are represented as encrypted numbers and the value hidden from public view.

Creating Confidential Assets:

EIP1724 aims to standardize the interface for interacting with confidential assets that conform to UTXO based models. AZTEC has reference implementations of EIP1724 for the supported asset types in the `@aztec/protocol` NPM package.



```
pragma solidity >=0.5.0 <0.6.0;
/**
 * @title ZkAsset Interface
 * @author AZTEC
 * @dev An interface defining the ZkAsset standard
 * Copyright Spilbury Holdings Ltd 2019. All rights reserved.
 **/

contract IZkAsset {

    event CreateZkAsset(
        address indexed aceAddress,
        address indexed linkedTokenAddress,
        uint256 scalingFactor,
        bool indexed _canAdjustSupply,
        bool _canConvert
    );
    event CreateNoteRegistry(uint256 noteRegistryId);
    event CreateNote(address indexed owner, bytes32 indexed noteHash, bytes metadata);
    event DestroyNote(address indexed owner, bytes32 indexed noteHash, bytes metadata);
    event ConvertTokens(address indexed owner, uint256 value);
    event RedeemTokens(address indexed owner, uint256 value);

    function confidentialApprove(
        bytes32 _noteHash,
        address _spender,
        bool _status,
        bytes calldata _signature
    ) external;

    function confidentialTransferFrom(uint24 _proof, bytes calldata _proofOutput) external;

    function confidentialTransfer(bytes memory _proofData, bytes memory _signatures) public;
}
```

The EIP 1724 ZkAsset Standard

Private Loan Application:

Built on top of AZTEC, the loan application provides for the following functionalities:

- A borrower can create a loan request with a confidential loan notional.
- A lender can request access to see the value of the loan notional.
- A lender can settle a loan request by transferring the notional to the borrower, the transfer notional should be confidential. The blockchain should verify that the notional amount and the settlement amount are equal.
- The borrower should be able to pay interest into an account that the lender can withdraw from. Any payments to the interest account should be confidential.
- The lender should be able to withdraw interest from the interest account as it accrues up to the last block time. The blockchain should verify the amount of interest the lender is withdrawing is correct, and the withdrawal amount and the balance of the account should remain confidential.
- The lender should be able to mark a loan as defaulting if the interest account does not contain sufficient interest. The blockchain should validate that this is the case whilst keeping the total interest paid, the account balance and the loan's notional confidential.
- The borrower should be able to repay the loan and any outstanding accrued interest at maturity. Both the interest and the notional repayment should remain confidential.

Combination of two confidential assets:

- Loan ZkAsset (ZkAssetMintable): represent a loan
- Settlement ZkAsset (ZK-Asset): stores notes that are used for value transfer: primary settlement, interest payments and repayment.

Proofs that would be used are:

- ***Mint Proof:*** to mint a note that represents the value of Loan ZkAsset.
- ***Join Split Proof:*** to validate fund deposit, withdrawal, and transfer.
- ***Bilateral Swap Proof:*** to settle a loan by swap notes between loan asset and settlement asset.
- ***Dividend Proof:*** to validate the accrued interest is calculated correctly per loan setting.
- ***Private Range Proof:*** to validate that the accrued interest is greater than the available balance inside the interest account.

By leveraging AZTEC, developers can build fully anonymous DApps while ignoring the detailed implementation of ZKP. This version of AZTEC is totally running on current blockchain infrastructure (ETH), and thus is constrained by the scalability and gas fee (of ETH). The new version of AZTEC is built on its own L2 solution.
Compared with ZK-SNARK or ZK-STARK, Sigma-protocol is weak in terms of expressing ZK logic. The new version implements ZKP based on PLONK ZK-SNARK. These AZTEC proofs are hard to understand and are not very developer friendly.

Link to commented codes for [Loan.sol] and [LoanDapp.sol]

Q3 Webb

Webb protocol is tornado cash with a bridge built on top of it. EVM code, relayer code, substrate code (in development). Webb is not live yet, it's only on testnet. Code is not yet complete so be aware of that while reading it.

1.What is the difference between commitments made to the Anchor and VAnchor contracts? Can you think of a new commitment structure that could allow for a new feature? (eg: depositing one token and withdrawing another?) If so, what would the commitment look like?

Commitments to Anchor have poseidon hash consisting of ChainID, nullifier and secret, whereas VAnchor has ChainID, amount, nullifier and so-called "blinding" (which is the secret from Anchor commitment).

Anchor:

```
Commitment = Poseidon(chainID, amount, pubKey, blinding)
```

VAnchor:

```
Commitment = Poseidon(chainId, nullifier, secret)
```

New commitment structure, allowing for token swap:

```
Commitment = Poseidon(chainID, amount, tokenAddressOne,
tokenAddressTwo, pubKey, blinding)
```

3. Describe how the UTXO scheme works on the VAnchor contract.

- UTXO stands for Unspent Transaction (TX) Output. Basically, A UTXO is the amount of digital currency remaining after a cryptocurrency transaction is executed.
- UTXOs are processed continuously and are responsible for beginning and ending each transaction. Implementing UTXOs greatly simplifies the accounting methods of the blockchain. Instead of having to track and store every single transaction, in order no less, we just need to track the unspent coins, also known as the UTXOs.
- When a transaction is completed, any unspent outputs are deposited back into a database as inputs which can be used at a later date for a new transaction.
- Using the preimage / UTXO of the commitment, 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. The chain id opcode is leveraged to prevent any tampering of this data.

4. Explain how the relayer works for the deposit part of the tornado contract

Deposit transactions won't go through relayer, only transfer and withdrawal (this doesn't for now) do. Relayer listens to deposit event emitted by contract `FixedDepositAnchor` and save the event with following info:

- chain_id, contract.address, deposit.commitment, deposit.leaf_index
- block_number

Question 4: Thinking in ZK

1. If you have a chance to meet with the people who built the above protocols what questions would you ask them?

Webb: Why did you pick substrate as your primary inter-chain mechanism over Cosmos?