ZKU Assignment Week 5

- My choose was Stream B, which is a `infrastructure track`.

Question 1

> In week 2 you 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.

Question 1.1

> 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.

TL;DR

- It uses both a mixer and a cross-chain bridge. Users will be able to deposit tokens on Ethereum, and establish a zero-knowledge proof of ownership of the fund on the other Harmony protocol. After depositing funds, the smart contract will update the Merkle Tree to commit the new Merkle root.
- This new Merkle root should then be transmitted from Ethereum to the Harmony, so that the mixer's state is synced throughout both chains. The user will then be able to verify to the protocol that they hold a given quantity of tokens in the protocol using the zero knowledge proof created during deposit.

Requirements

- User would like to deposit 100K UST tokens from Ethereum to Harmony. This action allows users to withdraw same amount of UST tokens from Harmony network.

Component

1. `contract_one` : A smart contract deployed on Harmony network to withdraw funds if and only if an appropriate proof is given.
2. `contract_eth` : A smart contract deployed on Ethereum network that user has 100K UST tokens in the mapping.
3. `circuit_eth` : A circuit contract deployed on Ethereum network that user generates proof of depositing and withdrawing assets.

Deposit process

> `Web Browser`

1. When depositing assets from User through `webb.tools` in a browser, user generates two secrets. Let us call them `S_1`, `S_2`.
2. The `commitment`, or `C`, later used in a ZK proof for withdrawal is generated upon a user's deposit. The commitment is the `PoseidonHash(DestinationChainID + nullifier + secret)` when using Webb Protocol. It adds the commitment to its Merkle tree and emits `event Deposit(_commitment,`

`insertedIndex, block.timestamp)`.

3. The `nullifier` is created with `N = hash(S_1)`. It is a public input that is sent on-chain to be checked with the smart contract & the Merkle tree data.
4. User submits a hashed `commitment` made in the above step to the Anchor's merkle tree for insertion, when calling deposit function. The commitment contains the chainID of the chain they wish to withdraw on as well as some secret data.

> `contract_eth`

1. The contract verifies the amount of funds.
2. A relayer picks this up and opens proposal to update the state. The contract inserts the `commitment` parameter to the deposit merkle tree. This proposal is then voted on by the relayers. If a proposal is passed, It calculates a newly created deposit tree root. The root is stored to a cache of Anchor 1's last 30 historical Merkle roots.
3. It returns merkle root and merkle path to the user. The parameters are saved to local storage of browser, which is able to be called with `ethereum` object on Chrome console. By this mean, Chain 2 can now be submitted a valid zero-knowledge proof to withdraw by other users.

Withdrawal (lower privacy level involved)

- User must show the Harmony contract that her deposit commitment is a leaf on a Merkle tree with root in one of these caches in order to withdraw assets.
- Each anchor keeps a cache of the last 30 historical Merkle roots of all the anchors with which it has an edge. Alice must show the anchor 2 that her deposit commitment is a leaf on a Merkle tree with root in one of these caches in order to withdraw on chain 2. A proof like this is known as a `one-of-many` Merkle proof.
- However, if Alice exposes the secret and nullifier, her transfer of cash from chain 1 to chain 2 loses a lot of privacy.

> `contract_one`

1. User generates a proof that one knows two secrets, `S_1`, `S_2` and merkle path.
2. User can compute the merkle root and nullifier rendered from `S_1`, `S_2`, and `merkle path`. As stated above, it is called as a `one-of-many` Merkle proof.
3. The anchor on chain 2 which receives the following parameter - `proof`, `merkle root`, `nullifier` and a `withdraw recipient address` can then validate the Merkle proof as one of many, checks nullifier is not in `nullifier_history` to avoid double spending, verifying proof whether it is created by public input - merkle path and nullifier - and release tokens to Alice's chain 2 account.

Withdrawal (higher privacy level involved)

- To involve a higher privacy level, one can exploit `Withdraw` circuit in which the circuit's constraints are met if and only if Alice may withdraw legitimately on chain 2.
    - The length and levels parameters are used in the `Withdraw` circuit. The number of anchors connected across the bridge by an edge is called length. The height of the anchor Merkle trees is measured in levels.
    - The following parameter is private.
        - `secret`, `nullifier`

        - `pathElements[levels]` : an array of the Merkle proof elements.
        - `pathIndices[levels]` : a binary array that indicates whether the corresponding pathElement element is on the left or right side of the Merkle tree.
        - `diffs[length]` : an artifact of the SetMembership gadget that we describe below.
    - The following parameter is public.
        - `nullifierHash`, which equals Poseidon(nullifier, nullifier).
        - `recipient`, the address of the recipient (not used in computation).
        - `relayer`, the address of the relayer (not used in computation).
        - `fee`, the fee paid by Alice to the relayer (not used in computation).
        - `refund` described in a section below (not used in any computation)
        - `refreshCommitmentdescribed` in a section below (not used in any computation)
        - `chainID`, the destination chain ID, or in our case 2.
        - `roots[length]` the set of roots Alice's is proving membership in.
- It uses `CommitmentHasher` circuit that takes in the `secret` and `nullifier` and computes the commitment and the nullifierHash.
- The `SetMembership` gadget accepts an element and a set and determines whether or not that element is a member of that set. The circuit also takes in diffs[length].
- The `ManyMerkleTreeChecker` gadget is in charge of ensuring that the Merkle proof is one-of-many. It computes the root of the Merkle tree corresponding to the commitment, `pathElements[levels]`, and `pathIndices[levels]` inputs. It also takes in `roots[length]` and checks whether the calculated Merkle root is in `roots[length]` using the `SetMembership` gadget.

Challenges and Improvements

- We need to ensure that the protocol's state is synced across both blockchains. If a user withdraws funds from both Ethereum and Harmony at the same time, it could result in double spending on both mainnets.
- To avoid this issue, the protocol might be able to add an additional layer during the withdrawal process that locks funds on the other chain while the withdrawal is completed. When a user withdraws tokens on Ethereum, a signal is sent to Harmony. The layer on Harmony protocol causes the state to be locked while the withdrawal process is done, and to be committed to the chain.

Question 1.2

> 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).

- To begin with, it is to note that the wormhole attack in Solana was a logic flow that allows attackers to exploit a signature verfication vulnerability so that the hackers were able to mint tokens without any valid signatures provided. The contracts incorrectly assumed that `==` and `&&` were interchangeable, but verifying transactions in that manner is not correct. `false == false` evaluates to `true`, whereas `'false && false'` evaluates to `false`.

- Performing transfers between Layer 1 could generate possible harms - double spending -, since someone are allowed to fork the main chain while the transfer request has been in progress. I believe that unless we are utilizing zk-proof based bridge, a KYC process might be required to provide possible malicious actions. In addition, guaranteeing smart contract security is also big challenge by auditing and reviewing from various parties.

Question 2

> 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.

Question 2.1

> 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.

Note

- Aztec note is basically encrypted values, which is at the heart of every transactions. A note registry manages the condition of notes for each assets, while the total of all valid notes that is owned by a user address in a specified Note Registry is the user's asset balance on AZTEC network.

Range

- The `Note` is used to confirm that the note is greater than another or vice versa. This is important for demonstrating that post-trade ownership of an asset is below a regulatory maximum. It may also be used to create identification and membership schemes for groups.

Swap proofs

- The bilateral swap proof allows two notes to be swapped atomically. This is important when exchanging two assets, such as fiat currency and a loan, bond, or security. The makers bid note is equal to the takers ask note, and the makers ask note is equal to the takers bid note, according to a certified evidence.

Dividend Proofs

- The prover can use this evidence to show that an input note equals an output note multiplied by a ratio. This can be used to pay interest on an asset.

Join-Split proof

- The Join Split proof joins or splits a collection of input notes into a set of output notes. This is commonly used to merge note values into a bigger note or to split a note into many notes with different owners. The total of the input notes is equal to the sum of the output notes, according to this proof.

Question 2.2

> [Infrastructure Track Only] 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)

- The dApp utilizes 1) Loan zkAsset to represent a loan, 2) Settlement zkAsset to store notes for transferring funds, primary settlement, and interest payment/repayment while using aformentioned techniques of proof.
- See comment on Loan.sol
- See comment on LoanDapp.sol

Question 3

> 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.

Question 3.1

> 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?

- Anchor only accepts deposits and withdrawals in specified amounts (determined in denominations). The following is how the commitment is calculated.

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

- The commitment for VAnchor (Variable Anchor), a variable-denominated shielded pool, is computed as follows.

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

- The commitment would be the following.
  - `data` : any arbitrary data that specifies token amount for example.
  - The withdrawal amount is decided by the token1/token2 exchange rate. Alternatively, we might remove token2 and allow the withdrawer choose which tokens to be withdrawn.

```
commitment = Poseidon(chainID, data, pubKey, blinding)
```

- This is new commitment that allows token swap.

```
commitment = Poseidon(chainID, amount, addr1, addr2, pubKey, blinding).'
```

- Or, to enable token exchange, it is note that withdrawing the original token and calling a decentralized exchange contract to convert to target token would be viable solution.

Question 3.2 (optional)

- Pass

Question 3.3

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

- `VAnchor` utilizes the same mechanism as Bitcoin protocol has to represent variable assets for depositing and withdrawaling.
- All commitments are stored in a merkle tree, while UTXO is the hash input of the commitment.

```
UTXO = (destinationChainId, amount, pubKey, binding)
```

- `transact()` or `transactWrap()` functions combines deposit and withdrawl actions by utilizing split-join techniques for UTXO notes. `transact()` functions commits to the merkle tree after verifying the proof. The function does the following things: 1) verify the proof 2) validate nullifiers so that that given notes have not used before 3) append nullifiers to nullifier array 4) append note4 and note5 to the merkle tree, 5) transfer funds to the specified account according to the amount given from the newly created note.
- When depositing assets, user creates `VAnchor` commitments in web browser then call the contract function `VAnchor::transact()` with the parameters of 1) zk-snarks proof for the commitment 2) hash of the commitment 3) funds.
- When transferring and withdrawaling funds, users are required to combine multiple input UTXO notes with two output notes. One would call `VAnchor::transact()` with 1) commitment's hashes (including old notes and newly created notes for withdrawal) 2) nullifiers for notes to be withdrawn 3) a zk-snark proof that join-split operation has done correctly and fund amounts have been calculated spotlessly.

Question 3.4

> [Infrastructure Track Only] Explain how the relayer works for the deposit part of the tornado contract

- Only the deposit event issued by contract FixedDepositAnchor is saved by Relayer. They do not participate in depositing activities and only participate in transfer and withdrawal activities.
- Relayer retrieves the 1) `chain_id`, 2) `contract address`, 3) `commitment`, 4) `leaf_index`, 5) `block_number` to save the withdrawal event.

```
match event {
            TornadoContractEvents::DepositFilter(deposit) => {
                let commitment = deposit.commitment;
                let leaf_index = deposit.leaf_index;
                let value = (leaf_index, H256::from_slice(&commitment));
                let chain_id = contract.client().get_chainid().await?;
                store
                    .insert_leaves((chain_id, contract.address()), &[value])?;
                store.insert_last_deposit_block_number(
                    (chain_id, contract.address()),
                    log.block_number,
```

```
            )?;

            tracing::debug!(
                "Saved Deposit Event ({}, {})",
                value.0,
                value.1
            );
        }
```

Question 3.5

> [Bonus] Write a program using ethers-rs to interact with the dark-forest smart contract you created for assignment 3.

- Due to the lack of time, I am now attempting to build the contract but yet to finish.
- My goal is to submit the contract and commit to the repository on Github until late submission day. See below.
- GitHub Link

Question 4

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

AZTEC

- Why the protocol chooses to focus on building zk-SNARKs instead of zk-STARKs? In addition, The protocol uses PLONK technology moved from Sigma Protocol. What drives this decision?

Webb

- Why Webb protocol decide to build their implementation in Substrate mainnet? Considering Webb Protocol has its groundings on Tornado Cash, when to support other multiple networks to bridge like Tornado Cash?