Assignment 5

1. Shielding the TX

*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*

1. *You have been asked to present a mechanism that will allow a user to bridge 100,000 UST tokens privately and securely 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.*

   [ANSWER]

   Goal:

   - User can deposit (100K) UST from Ethereum to Harmony, then this user can withdraw same amount of UST to a Harmony account.
   - There is no way to link a specific deposit (on Ethereum) to a withdrawal (on Harmony).

   No-goal

   - The amount can be variable.
   - User don't need to withdraw fund from Ethereum after making the deposit on Ethereum.
   - The bridge is one-way: from Ethereum to Harmony.
   - Deploy mixer in Harmony to hide further fund transactions after the fund is withdraw to Harmony.

   Component:

   - S_e: a smart contract on Ethereum that user can deposit 100K UST.
   - S_h: a smart contract on Harmony that user can withdraw fund by given a proof.
   - C_e: a circuit that can generate proof of a deposit.
   - Light-client:
     - Retrieve merkle roots stored in S_e from Ethereum and submit to Harmony S_h
     - Move fund from Ethereum to Harmony

   Protocol

- Deposit
  - In browser (connects to Ethereum)
    - User generates 2 secrets: s1, s2
    - Calculate a commitment as, c = hash(s1, s2)
    - Calculate nullifier as, n = hash(s1)
    - User call S_e to deposit 100K UST with parameter: `c`
  - S_e:
    - Verify the fund amount is correct.
    - inserts `c` to its deposit merkle tree and calculate a new merkle root: MR_t. Save MR_t to Merkle root history: MR_History_E[].
    - Return merkle root and merkle path to the user.
  - In Browser
    - User saves s1, s2, n, MR_t and merkle path to local storage.
- Light-client
  - It runs periodically to retrieve new merkle roots in MR_History and push them to Harmony by calling contract S_h (then stored in MR_History_E[])
  - It withdraws fund deposited in contract S_e and deposit to S_h. Like a one-way clearinghouse.
- Withdrawal
  - In browser (connected to Harmony)
    - User generates a proof from C_e that prove:
      - It knows s1, s2 and merkle path thus,
        - It can correctly compute MR_t from: s1, s2 and merkle path.
        - It can compute n (nullifier) from s1
    - User sends the proof, MR_t, n and a withdraw recipient address to S_h
  - S_h:
    - Verify MR_t is in the proof history MR_History_E
    - Check n is not in the nullifier_hostory to avoid double spending
    - Verify proof using public input: MR_T and nullifier
    - Transfer 100K to the recipient

Challenges and Improvement

- Light-client: is a centralized and needed to be fully trusted by all users. This seems scary. It has 2 major vulnerabilities
  - Malicious merkle root sync: dishonest light-client can push fake merkle root to Harmony network and then withdraw fund from Harmony. Possible solution:
    - Run multiple light-clients that are governed by different party. To push new merkle root to Harmony, the value needs to be multi-signed by all parties and then verified by S_h.
  - Full access to fund: malicious light-client can steal all fund deposited to ETH pool. Possible trustless solution (based on Witness Encryption):

        - Instead of depositing fund to S_E, user can send the fund to an ETH deposit address owned by both light client and the user. The fund is locked in this address. User then get a proof for this deposit.
        - User can mint respective token in Harmony by submitting the ETH deposit proof.
        - Upon amortizing the Harmony token, user can get a witness that enable her to unlock the fund in ETH deposit address.
- Full anonymity: we can use relay service, a trusted third party to hide the payment of gas and provides full anonymity.
- Delay: after deposit, user needs to wait some time to get the merkle root synced to Harmony. We could make the light-client run more frequently but this will result on more gas consumption.
- Further improvement
    - Variable amount: to support deposit and withdraw a variable amount. We will need to implement a join/split UXTO mechanism (like: Tornado Cash Nova and Webb).
    - Two-way deposit/withdrawal: if we want to support deposit on Harmony and withdraw from Ethereum. Or deposit in Ethereum and withdraw from Ethereum. We need to store 2 merkle root histories and keep them synced between Ethereum and Harmony. We also need to add a withdraw pending status to prevent withdrawal same fund from 2 network simultaneously (double-spending).
    - Better Mixer: we could add a more sophisticated mixer in both network so that after user depositing fund to the pool, we can hide all later transactions in Ethereum and Harmony (as what Webb did).

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).*

   [ANSWER] ZKP can help to reduce such risk significantly. As described by above protocol, double withdrawal can be prevented by using nullifier check. Every withdrawal, we require user to a circuit proof to prove the fund ownership and validity of the transaction.

   Light client (or relayer) needs to be trustless and governed by multiple parties (or even better: banks). Please refer to the challenge and improvement section above.

   In addition, I believe a full security audit by 3rd party and good engineering practice in place are also very important to improve the security of every critical Web3 app.

3. ***[Bonus]*** *Design a smart contract to be deployed on both blockchain to take care of the user's request and circuits to take care of the privacy*

   [ANSWER] Covered in first question.

2. 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*

   [ANSWER]

   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. How this abstract representation maps to real quantities are a higher-level detail of digital assets that utilize the protocol. Note is comprised 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 7 of proofs:

   1. 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.

   2. 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 makers ask note is equal to the takers bid note.

3. 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.

4. Mint

The mint proof allows the supply of AZTEC notes to be increased by a trusted party. e.g., a stable coin mints an AZTEC note equal to the value of a bank transfer it receives.

5. Burn

The burn proof allows the supply of AZTEC notes to be decrease by a trusted party. e.g., a stable coin burns an AZTEC note of equal value of the bank transfer it sends to the note owner.

6. 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.

7. 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. **[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)

[ANSWER]

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 withdraw 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.

  To build the above functionalities, the dApp combines two confidential assets:

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

  And use following proofs:

  - Mint Proof: to mint a note that represent the value of Loan ZkAsset.
  - Join Split Proof: to validate fund deposit, withdrawal, and transfer.
  - Bilateral Swap Proof: to settle 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.

  <u>Benefit</u>

  - By leveraging AZTEC, developer can build fully anonymous DeFi app while ignoring the detailed implementation of ZKP.

  <u>Challenge</u>

  - 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.

Comments on contract code inline:

https://github.com/geesimon/zku/blob/main/week5/Loan.sol
https://github.com/geesimon/zku/blob/main/week5/LoanDapp.sol

3. 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 mixer, 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?

   [ANSWER]

   - Anchor only supports fixed amount (defined in *denomination*) of deposit and withdrawal. The commitment is computed as:
     - commitment = Poseidon(chainId, nullifier, secret)
   - VAnchor (Variable Anchor) is a variable-denominated shielded pool, and the commitment is computed as:
     - commitment = Poseidon(chainID, amount, pubKey, blinding)
   - We can generalize commitment as:
     - commitment = Poseidon(chainID, data, pubKey, blinding)

     Parameter: *data* can be any arbitrary data. For example: {token1, amount1, token2} defines a specific token1 and amount1 for this note and can be withdrew to token2. The amount to be withdrew is determined by token1/token2 Exchange rate. Or we can remove token2 and let the withdrawer decide which token s/he wants.

     A new Anchor contract (say: ExAnchor) needs to be built to support this use case. To enable token Exchange, one possible solution is to first withdraw the original token and call DEX contract to convert the original token the desired token.

2. *Describe how the UTXO scheme works on the VAnchor contract.*

   [ANSWER] To support variable fund deposit/withdraw, VAnchor use UTXO (like Bitcoin) scheme to represent fund. UTXO is the hash input of the commitment (UTXO = { destinationChainID, amount, pubkey, blinding }), and all commitments are stored in merkle tree. Deposit and withdrawal are unified under a common transact

function (transact() or transactWrap()) and is split/join operation from source UTXO notes to target UTXO notes.

- Deposit: user creates VAnchor commitment in a client (for example: browser) and then call contract function VAnchor::transact () with:
  - hash of this commitment as public input of proof
  - a zkSNARK proof to prove the commitment is well-formed
  - fund token

  In transact(), after proof verification, the commitment is stored in a merkle tree.

- Transfer/Withdrawal: user needs combine multiple UXTO notes (input notes) and output 2 new UXTO notes.

  To illustrate this process, say Alice has 3 UXTO notes:

  - note1: 1 eth
  - note2: 0.5 eth
  - note3: 2 eth

  Its total of 3.5 eth. Now she wants to withdraw 3.2 eth to a specific recipient address.

  - Alice creates 2 new UXTO notes (on her client): note4 (3.2 eth), note5(0.3 eth).
  - Alice call VAnchor::transact () with:
    - Hashes of commitments of: note1, note2, note3, note4, note5. These are public input
    - nullifiers for note1, note2, note3 and note4
    - a zkSNARK proof to prove the notes join/split operation is correctly composed and fund amounts are calculated correctly
      - i.e.,: note1 + note2 + note3 = note4 + note5
  - In transact():
    - Verify the proof
    - Check note1, note2, note3 and note4 are not used (by checking nullifiers)
    - Insert nullifiers to nullifier array
    - Insert note4 and note5 to merkle tree
    - For withdrawal, transfer fund to specified recipient according to the amount specified in note4

3. ***[Infrastructure Track Only] Explain how the relayer works for the deposit part of the mixer protocol***

[ANSWER] Deposit transaction won't go through relayer, only transfer and withdrawal 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

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?*

[ANSWER]

AZTEC

- I realized the first version of AZTEC is available 3 years ago and use Sigma-protocol as the underline ZKP technology, now its PLONK based. It would be interesting to share the product evolvement in the past 3 years and what major challenges have been conquered during this period.
- Why AZTEC need another Private Layer 2 solution based on zkRollup instead of using the existing ones, like zkSync? What are the major advantage/disadvantage of this L2 solution compared with others?

Webb

- Webb is based on Tornado Cash with bridge functionality that support ETH and Polkdot. Given Tornado Cash already support multiple networks (Ethereum, Binance, Polygon) and bridges seem an obviously next move, what would be the edge Webb can offer in the future?
- Any plan to have native token for Webb (as TORN for Tornado Cash)?