Supporting Private Anonymous Transactions with Ethereum

Ishank Agrawal, Paulo Chade, Helen Liu

May 14, 2024

1 Abstract

With the rise of cryptocurrencies, there is a growing movement for central bank digital currencies, but in the United States, there is an equally strong push back against them for their lack of privacy in transactions. Our project aims to address this issue by support private anonymous transactions, specifically using Ethereum. In this paper, we propose a system that obscures the amount contained in a transaction, along with making both the sender and receiver indistinguishable from any other participant in the network. The scheme involves generating temporary addresses, signing transactions to prove ownership to the network, and verifying these transactions securely. The target audience for our project is financial institutions because there hasn't been as much work done for them yet, so we can reduce the scope of the project to exclude scalability and focus on functionality first.

2 Background

The impetus behind creating a scheme for private anonymous transactions comes from the idea of central bank digital currencies (CBDCs) and how the major opposition to implementing CBDCs in the United States is their lack of privacy. CBDCs are based on the blockchain, which are usually public, so transactions would be public as well.

2.1 The blockchain

By definition, the blockchain is "a distributed database or ledger shared among a computer network's nodes" [1]. It stores data in blocks linked by cryptographic proofs. It is often used in relation to cryptocurrency, and it is relevant to our project because of this, but it does not have to be used for cryptocurrency.

The blockchain type that we will discuss in the rest of this paper is a decentralized blockchain, where no single person or group has control. Because blockchains are distributed, multiple copies of the data are stored on different machines, and they all must match. Different cryptocurrencies handle committing transactions to the blockchain differently, but in the case of Ethereum, every transaction must be validated before being committed to the blockchain, and once they are committed, they are immutable. Many blockchains also act as public databases, so every transaction would be visible to anyone with an internet connection.

2.2 Central bank digital currencies

Central bank digital currencies are a digital form of currency issued by a country's central bank. As cryptocurrencies become more ubiquitous, countries are increasingly adopting CBDCs. The Bahamas, Jamaica, and Nigeria have already fully launched a CBDC, and there are 36 ongoing CBDC pilots, with the European Central Bank piloting a digital euro and China's bank piloting a digital yuan [2]. However, the major opposition to CBDCs in the US is their lack of privacy being based on the blockchain, where all transactions can be seen by everyone. The implication here is that all of our financial transactions specifically will be public knowledge.

3 Overview

3.1 Problem

We want to design a system that obscures the amount contained in any given transaction, along with making both the sender and receiver indistinguishable from any other participant in the network. Since Ethereum is public by nature, one challenge is to obscure a given user's identity from the network while still allowing them control over their complete balance. Another

challenge is the problem of the network verifying a given transaction when it does not know the sender, the receiver, and the amount being transacted. Finally, with transactions using single-use addresses for sender and receiver, the last challenge lies in ensuring a given user has access to the combined balance of all the amounts associated with their many addresses.

3.2 Related work

There have been a few systems that already aim to tackle the problem we are addressing, and they have been helpful in the construction of our scheme:

- Monero [3] is a separate blockchain created with the primary purpose of enabling private financial transactions. Every user is anonymous, and all transactions are private, but it is completely separate from the main blockchains (e.g. Bitcoin, Ethereum), meaning it requires users to start from scratch on it.

- Aztec [4] is a layer 2 system built on top of Ethereum (which is composed of layer 1 and layer 2). While Aztec is able to achieve privacy within the layer, it achieves this by sacrificing two key aspects. Its value storage system is based on Bitcoin's UTXO, which is completely different from Ethereum's, making it not possible to use existing applications. Secondly, some transactions are not anonymous, namely those that send and receive funds from layer 2.

- Zama [5] is similar to what we want to accomplish, as it is built on top of Ethereum and hides the content of transactions. However, it does not hide the transaction participants.

- The idea of stealth addresses [6] was proposed to hide the recipient in transactions by having the recipient create a stealth address that obscured their true identity. We wanted to extrapolate this idea to the user privacy aspect of our project.

4 Ethereum

Ethereum is a decentralized blockchain network that acts as a robust infrastructure for a wide range of applications, digital assets, and organizations.

Unlike Bitcoin, a pioneer in peer-to-peer monetary transactions, Ethereum innovates with the concept of smart contracts. These contracts allow for the execution of arbitrary code on the blockchain, greatly enhancing its functionality beyond simple financial transactions. Consequently, Ethereum supports various decentralized applications (dApps), including decentralized currencies and finance (DeFi) systems, as well as decentralized voting mechanisms. This broad capability gives Ethereum a distinct edge over networks like Bitcoin, increasing its utility across diverse sectors. Central to Ethereum, as with all blockchain technology, is its public nature - every transaction is visible to everyone on the network, embodying the principle of "don't trust, verify". This transparency is crucial for security and trust in the system. However, it can affect many applications that use sensible data (should be private).

4.1 Ethereum Tokens with Privacy

In a very simplistic way, Ethereum tokens are a smart contract that have the following characteristics:

1. The have mappings of addresses to balances. E.g $balances[address] = amount$.
2. They have a function that allows a user to transfer tokens to another address. E.g $transfer(address, amount)$.
   - 2.1 $balances[sender] -= amount$
   - 2.2 $balances[receiver] += amount$

As one of the main benefits of blockchains is that anyone can verify the authenticity of the data, by default, everything is public. This means that anyone can see the balance of any address, as well as the transactions that were made with it. This is a problem when the use case needs privacy (e.g a bank account balance should not be public, as well as the transactions that were made with it).

Privacy

In order to achieve privacy two main characteristics are needed: Anonymity and Confidentiality. We define them as follows:

1. Confidentiality: The data of the transaction should look completely random.

2. Anonymity: It should be hard to link a transaction to a person.

To address confidentiality we propose the use of homomorphic encryption, in that way we would have the following: $balances[address] = Enc(amount)$

To address anonymity we propose the use of different addresses for each transaction. More specifically, we define a way to generate different addresses using the same secret key. We suggest a LWE based assumption as it offers post-quantum security. We will describe the scheme in the next section.

5 Proposed scheme

We will describe the proposed scheme, giving a example of how it would work. Imagine that Alice wants to send some funds to Bob. The transaction flow would be as follows:

1. Bob generates an address with his sk

2. Bob sends the anonymous address to Alice.

3. Alice generates her signature, encrypts value she wishes to send Bob and generates a ZKP of her transaction

4. Alice sends a message to the network

5. Everyone is able to verify the state update without any information on the message value

Now, a couple parts of that can still feel a bit abstract, for example, how does Bob generate a new address? How does the network verify the balance of the sender and the receiver if everything is encrypted? How can you claim ownership over an asset if no one knows your address? And finally, if every transaction is generated a different address, how to "combine" balances?

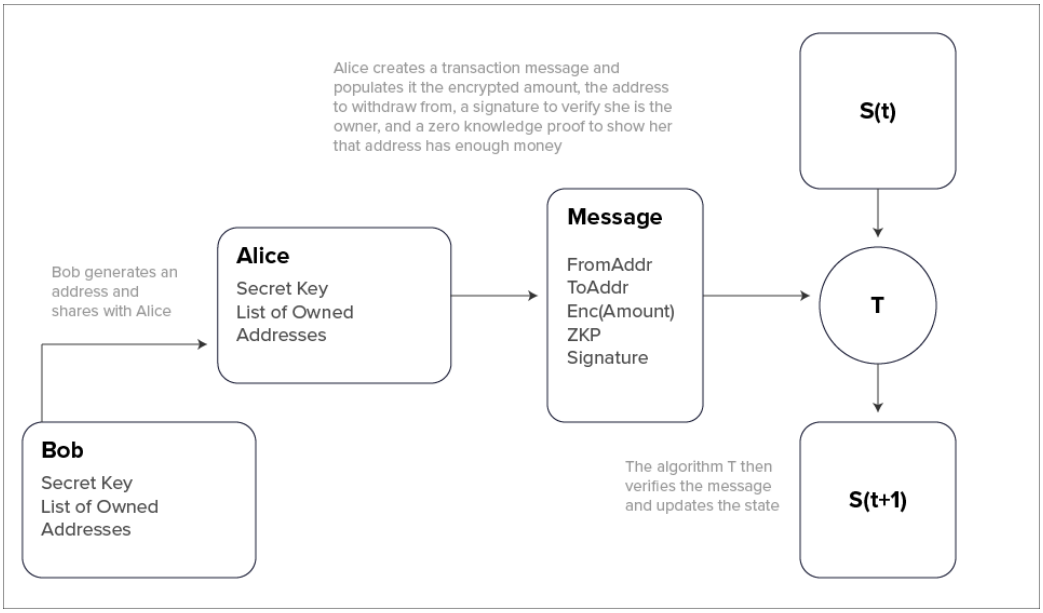


Figure 1: Solution Diagram

5.1 Address generation

LWE operation is defined as follows:

$$\begin{bmatrix} \\ A \\ \\ \end{bmatrix} \cdot \begin{bmatrix} s \end{bmatrix} + \begin{bmatrix} e \end{bmatrix} = \begin{bmatrix} b \end{bmatrix}$$

The proposed cryptographic scheme can be detailed as follows:

1. The public key consists of the matrix A and the vector b, thus $pk = (A, b)$.

2. The secret key is composed of the vectors s and e, thus $sk = (s, e)$.

3. The address is derived by hashing the public key, hence $addr = H(pk)$. **Note: This follows the Ethereum standard where the address is the hash of the public key.*

In this scheme, the same secret key sk facilitates the generation of multiple addresses. Since each transaction requires a new address, using different pks without overwhelming storage of multiple (s, e) pairs necessitates varying the A matrix while keeping sk constant.

Transaction Flow

Consider a transaction scenario where Alice wishes to send funds to Bob:

1. Alice requests Bob's address.
2. Bob provides his address to Alice.
3. Alice completes the transaction by sending funds to Bob's address.

A critical point to address is the method of varying the A matrix to generate distinct addresses. How does this variation occur?

Naive approach

The naive approach would be to generate a new A matrix for each transaction. So step 2 would be:

2.1 Bob generates a new A matrix.

2.2 Bob uses the shared A matrix, and his secret key sk to generate the public key $pk = (A, b)$. He sends pk to Alice.

2.3 Alice uses the public key pk to generate the address $addr = H(pk)$. She sends the transaction to this address.

Although this approach is viable, it presents several challenges:

- Firstly, the requirement for Bob to transfer the matrix A to Alice introduces significant download and upload costs, making the data exchange burdensome.
- Secondly, Bob, along with the entire network, must maintain a record of all generated A matrices. This task demands substantial memory resources, which can become unmanageable as the number of users increases.

Costs: **where* $A \in \mathbb{Z}^{m \times n}$

1. **Data transfer:** $O(m \cdot n)$.
2. **Store:** $O(m \cdot n \cdot p)$. Where p is the number of addresses generated.

Pseudo-random approach

The pseudo-random approach involves generating the matrix A using a pseudo-random function based on a seed, ensuring that the same seed produces the same A matrix. The process is outlined as follows:

2.1 Bob generates a seed K. This seed allows anyone to generate the corresponding A matrix.

2.2 Using the shared seed K, Bob generates the A matrix and computes the public key $pk = (K, b)$. He then sends pk to Alice.

2.3 Alice, using the public key pk, generates the address $addr = H(pk)$ and directs her transaction to this address.

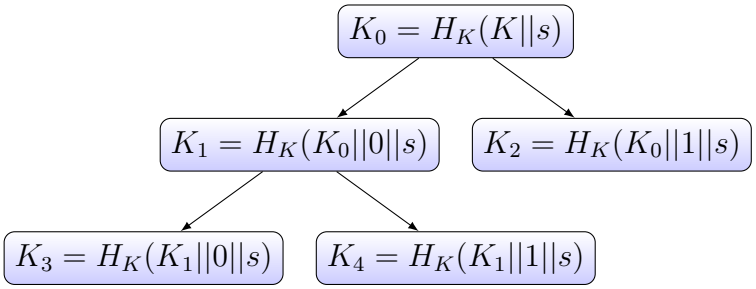
This method addresses one of the major issues of the naive approach: it significantly reduces the download and upload costs since the seed K is much smaller than the entire A matrix. However, the challenge of storing new data with every transaction remains unresolved.

Costs:

1. **Data transfer:** $O(|seed|)$. Where $|seed| << m \cdot n$.
2. **Store:** $O(|seed| \cdot p)$. Where p is the number of addresses generated.

Reverse merkle tree approach

To solve the problem of having to keep track of all the seeds that were generated, we can use a reverse merkle tree. Assume there is only one seed K, and a secret s:



In that way Bob only needs to keep track of the root K. Combining this with the pseudo-random approach, we have solved all the problems.

Costs:

1. **Data transfer:** $O(|seed|)$. Where $|seed| << m \cdot n$.
2. **Store:** $O(|seed|)$.

5.2 Signatures

Continuing the example of having Alice send funds to Bob, the next question is, how does Alice prove ownership of her ETH block to the network while revealing minimal private details? The solution we came up with is using a digital signature scheme. This digital signature scheme can be summarized by being LWE signatures using non-interactive zero-knowledge proofs of knowledge with Fiat-Shamir.

Signing

We begin with the signing step of this signature scheme. To make this step more understandable, we first explain the proof as if it was interactive:

- Alice wants to sign her message, so she sends the network the following proof, which proves that she knows $s \in \mathbb{Z}_q^n$ and $e \in \mathbb{Z}_q^m$:
    1. Alice computes $v \leftarrow Au + e' \in \mathbb{Z}_q^m$ for $u \leftarrow \mathbb{Z}_q^n, e' \leftarrow \chi'^m$ and sends v to the "verifier".
    2. The "verifier" chooses $\beta \leftarrow \{0, 1\}$ and sends β to Alice.
    3. Alice responds with $z \leftarrow u + \beta s \in \mathbb{Z}_q^n$.

4. The "verifier" accepts the proof if $(v + \beta b - Az) \in \mathbb{Z}_q^m$ is $(B + B')$-bounded.

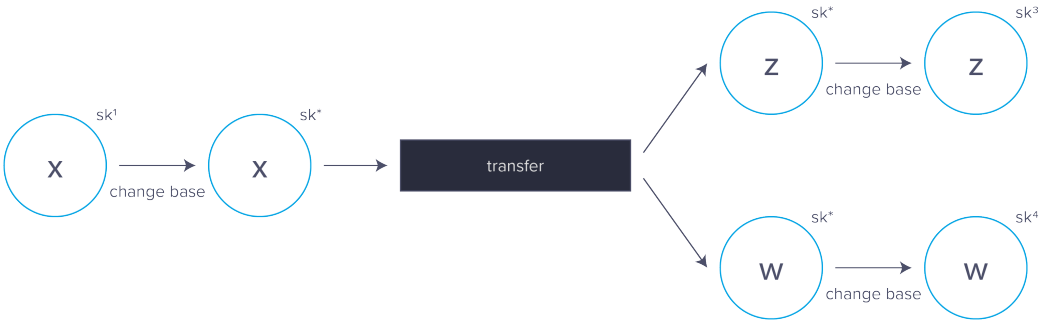
- Using this protocol with the noise-flooding lemma (where e' is sampled from $B' >> B$) only gives us a knowledge soundness error of $\frac{1}{2}$, so we then run this protocol t times in parallel to bring it down to $\frac{1}{2^t}$.

We then make this interactive proof non-interactive by having Alice construct $(\beta_1, ..., \beta_t) \leftarrow Hash(A, b, m, v_1, ..., v_t) \in \{0,1\}^t$, where m is the message to be signed. The final signature Alice sends is $(\overrightarrow{V}, \overrightarrow{\beta}, \overrightarrow{Z})$.

Verifying

The network runs the verifier for the zero-knowledge proof using the LWE instance (A, b) and the hash function $Hash(m, \cdot)$. If the verifier accepts, the network confirms Alice's ownership of the ETH block and proceeds to other checks on the transaction request before committing it to the blockchain.

5.3 Encryption and Transfer



Alice wants to send money from her account to Bob's account. Since all communication is public, Alice want's to accomplish the following things:

1. Prove that the transaction she wants to make is valid
2. Encrypt the new token with Bob's keys instead of using Alice's own keys.

To this end, we propose the following scheme:

1. First Alice generates three new pairs of secret keys and encryption keys:

$$(sk^*, ek^*), (sk^3, ek^3), (sk^4, ek^4) \tag{1}$$

2. Next Alice initiates a change of base procedure. A change of base procedure changes the encryption of the data to use a different key without changing the underlying value. Crucially, it must be possible to compute this change of base publicly. In this step Alice changes the base of her original token from (sk^1, ek^1) to (sk^*, ek^*).

3. Then Alice sends two new values, $Enc(z, ek^*)$ and $Enc(w, ek^*)$ along with zero-knowledge proofs that

   (a) z, w are positive

   (b) $z + w = x$

4. Alice initiates a change of base procedure on both token z and token w with (sk^3, ek^3) and (sk^4, ek^4) keys respectively.

5. Finally Alice tells Bob the secret key sk^4. Bob can initiate a final change of base procedure to privatise this token.

5.3.1 Changing Base

Let user provide send message $\text{Enc}(ek_2, sk_1)$ as well as a signature that proves user knows sk_1. Then the chain can decrypt with respect to sk_1 using FHE:

$$\begin{aligned} \text{Enc}(ek_2, x) &= \text{Enc}(ek_2, \text{Dec}, \text{Enc}(ek_2, \text{Enc}(ek_1, x)), \text{Enc}(ek_2, sk_1)) &(2)\\ &\approx \text{Enc}(ek_2, \text{Dec}(\text{Enc}(ek_1, x), sk_1)) &(3)\end{aligned}$$

5.3.2 Transfer

Since everything is encrypted with the same (sk^*, ek^*), Alice needs to prove two things

1. $x = z + w$

2. $z, w \in [0, 2^n)$

5.3.3 Invariant Sum

Because of FHE property, both Alice and the chain can compute

$$\text{Enc}(x - z - w) = \text{Enc}(x) - \text{Enc}(z) - \text{Enc}(w) \tag{4}$$

Hence Alice will simply attach a ZKP that this is indeed the encryption of 0.

5.3.4 Bounded property

This proof is similar to what Monero uses. In particular we propose to use a range proof. To show that $Enc(x)$ is the encryption of some $x \in [0, 2^N)$, Alice first computes the binary decomposition:

$$x = w_0 + \ldots + w_{N-1} \tag{5}$$

where w_i is either 0 or 2^i. Alice sends the encrypted values of $\text{Enc}(w_0), \ldots, \text{Enc}(w_{N-1})$, as well as ZKPs that w_i is either 0 or 2^i. Use also sends a ZKP that

$$\text{Enc}(x) - \sum_{i=0}^{N-1} \text{Enc}(w_i) \tag{6}$$

corresponds to the encryption of 0.

6 Combining values

In our previous discussion, we examined a straightforward transaction from Alice to Bob, assuming Alice had sufficient funds in her selected address. Suppose, however, that this is not the case. Consider a scenario where Alice possesses two addresses, $addr_1$ and $addr_2$, each holding a balance of x. She aims to transfer $1.5x$ to Bob. How would this transaction be structured?

Single Signature

A straightforward and intuitive approach would involve combining the balances of the two addresses in a single transaction, signed by the same individual. However, this method has a drawback: it effectively notifies the entire network that the owner of $addr_1$ also owns $addr_2$, thereby compromising the

owner's privacy by revealing linked identities.

Cost:

- One signature
- N proofs (one for each address)

Join Operation

Join-Operation: Another strategy is the "join-operation". In this approach, Alice transfers all funds from $addr_1$ to $addr_2$. Initially, it appears that these addresses are not linked to the same owner, allowing her to merge funds in a standard transaction. At first glance, this method seems viable, especially if employed sparingly. However, repeated instances where multiple addresses become inactive after transactions to the same recipient could arouse suspicions of common ownership.

Cost:

- N signatures
- N proofs

**these costs could be distributed overtime, as she can combine these values whenever (e.g a day after receiving money, she sends to her "main" account).*

N transfers

An alternative method involves executing multiple single transfers. In this scenario, two distinct transactions are conducted:

1. A transfer of x from $addr_1$.
2. A transfer of $0.5x$ from $addr_2$.

This approach makes it significantly more difficult to link both addresses to a single owner. However, it becomes more costly as the number of required transactions increases.

Cost:

- N signatures
- N proofs

**all at once, thus very expensive.*

7 Time and memory complexity

7.1 Address generation

The complexities associated with the address-generating protocol are as follows:

Time Complexity

The process to derive a public key involves the following computational steps:

1. Generating the matrix $A \in \mathbb{Z}^{m \times n}$ from a seed, which requires $O(m \cdot n)$ time.
2. Computing the operation $As + e$, which also takes $O(m \cdot n)$.

Additionally, there is an operation to identify the seed within a reverse Merkle tree. For a tree with h elements, this operation requires $O(\log h)$ time.
Thus, the overall **time complexity is** $O(m \cdot n) + O(\log h)$**.**

Memory Complexity

The memory requirements are proportional to the number of transactions processed, as each transaction necessitates the storage of a new address computed as $hash(A, b)$ where $b = As+e$. With the reverse Merkle tree structure, it is unnecessary to store all seeds since they can be dynamically recalculated. Therefore, assuming t transactions, **the memory complexity is** $O(t \cdot |hash|)$**.**

7.2 Signatures

The following time and memory complexities associated with the digital signatures used to prove block ownership to the network are quite high, but keep in mind that real-world implementations introduce more optimizations

that reduce the time and memory required, such as replacing LWE with a "Ring-LWE" variant and setting the iteration count t smaller by sampling the challenge c from a larger space.

Time Complexity (Signing)

The process to sign a message involves the following computational steps:

1. Computing $v \leftarrow Au + e' \in \mathbb{Z}_q^m$ for $u \leftarrow \mathbb{Z}_q^n$ and $e' \leftarrow \chi'^m$, which requires $O(m \cdot n)$ time.
2. Computing $\beta \leftarrow Hash(A, b, m, v)$, which takes $O(|hash|)$ time.
3. Computing $z \leftarrow u + \beta s \in \mathbb{Z}_q^n$, which takes $O(n)$ time.

Additionally, we run this protocol t times in parallel to bring down the knowledge soundness error to $\frac{1}{2^t}$, so the overall **time complexity is** $O(t \cdot m \cdot n + t \cdot |hash|)$**.**

Time Complexity (Verifying)

The process to verify a message signature involves checking that $(v + \beta b - Az) \in \mathbb{Z}_q^m$ is $B + B'$- bounded, which takes $O(m \cdot n)$ time from the matrix multiplication, but since we run the protocol t times in parallel, the overall **time complexity is** $O(t \cdot m \cdot n)$**..**

Memory Complexity (Signing)

At the end of the signing protocol, the signature is composed of $(\overrightarrow{V}, \overrightarrow{\beta}, \overrightarrow{Z})$, which represents the t values of (v_i, β_i, z_i). The memory requirements of each component are as follows:

1. Storing $v_i \leftarrow Au + e' \in \mathbb{Z}_q^m$ requires $O(m)$ memory.
2. Storing $\beta_i \leftarrow Hash(A, b, msg, v)$ requires $O(|hash|)$ memory.
3. Storing $z_i \leftarrow u + \beta s \in \mathbb{Z}_q^n$ requires $O(n)$ memory.

The overall memory requirement of signing a message is then $O(t \cdot (m + n + |hash|))$.

Memory Complexity (Verifying)

Since the network is not expected to know anything beyond the public key, which it should already know from the address generation step, and the signature, which is sent at the time of verifying, there are no additional memory requirements for verifying. It must store the sent signature though, so the memory requirement remains the same as that for signing, i.e $O(t \cdot (m + n + |hash|))$.

8 Possible attacks

8.1 Address generation

There are key considerations to take into account when using the key generating protocol, particularly concerning the dimensions of the matrix $A \in \mathbb{Z}^{m \times n}$.

Lower bound

Given that the error vector in LWE is bounded by a constant B and not truly random, the outcome of the operation $As + e$ may not appear random to adversaries. However, since the error vector does contain entropy, the Leftover Hash Lemma assures us that matrix multiplication can transform sufficient entropy into true randomness. We therefore require $m \gg (n + 1) \log q$.

Upper bound

A well-known threat in LWE systems is the linearization attack, which becomes feasible when a sufficient number of samples are available. Specifically, the condition is $m \geq \binom{n+|B|}{|B|} - 1$. Consequently, to avoid such attacks, it is necessary that $m < \binom{n+|B|}{|B|} - 1$.

Conclusion

In conclusion, the size of m must be carefully balanced - large enough to ensure security but not so large as to facilitate attacks. The bounds suggest $poly(n) < m < exp(n)$. As lower and upper boundaries grow with $poly(n)$

and $exp(n)$ respectively, selecting an appropriate n becomes crucial. Additionally, periodically changing the secret key (s, e) is recommended as a best practice.

8.2 Reverse merkle tree

Another potential attack on the described address-generating setup concerns the loss of anonymity. The Merkle tree structure allows for the calculation of many leaves from a single sample. As the hash is an one way function, an arbitrary adversary will not be able to determine the input from the output, however, everyone is aware of the tree structure (append 0 or 1 plus the secret). Therefore, it is imperative to keep the seed and the secret of the hash function confidential. If an attacker gains access to them, they could potentially uncover all the public keys associated with an individual, leading to a compromise of their identity.

8.3 Signatures

A potential attack on the digital signature scheme arises if $B' \approx B$. In this case, over many protocol runs, a verifies can take the average of many $(e + e')$ terms to get a good approximation of e, from which it can recover the secret s and learn all the information, making the proof of knowledge not zero-knowledge anymore. The simple implementation we laid out avoids this issue by having $B' >> B$ such that e' hides e, and the verifier learns nothing. The advantage at distinguishing e' from $(e + e')$ is then at most $\frac{B}{B'}$, so $B'2^{\lambda}B$. A less time and memory intensive method of accomplishing the same level of security without needing $B' >> B$ is for the signer to abort the protocol is the value for $(e + e')$ is too large after running for the first two steps.

9 Ethics

Cryptocurrency has emerged as a potent medium for facilitating illicit digital transactions, primarily due to its ability to enable anonymous financial activities, often beyond the scope of traditional regulatory mechanisms. Reports from 2023 indicate that illicit addresses amassed over 24 billion U.S. dollars, highlighting the substantial role of cryptocurrencies in the digital underworld

[7]. This figure is likely an underestimate, as it excludes numerous untraceable transactions - a fundamental reason for the adoption of cryptocurrency technology is its inherent challenge in linking digital addresses to real-world identities.

The developed solution aims to enhance privacy within blockchain systems and is particularly tailored for financial institutions operating within controlled networks. While these networks inherently facilitate transparency by making public keys accessible, the solution introduces an additional layer of privacy. This layer is crucial not only for safeguarding legitimate privacy interests but could also serve as a tool for criminals who wish to exploit it in other protocols, significantly complicating the linkage between a person and an address.

Thus, while the solution promises to revolutionize privacy within traditional financial systems, it simultaneously raises significant ethical concerns. It could potentially be exploited as a tool for criminal activities, masking illicit transactions behind enhanced anonymity. Therefore, the use of such technology needs to be well accessed within a context before its deployment.

10 Conclusion and future work

In this paper, we introduce a privacy-preserving scheme that leverages post-quantum cryptographic assumptions, particularly those based on the Learning With Errors (LWE) paradigm. Our proposal distinguishes itself by being adaptable to any decentralized network. At the heart of our scheme is a Fully Homomorphic Encryption (FHE) system that processes data while maintaining integrity and consistency, facilitated by Zero-Knowledge Proofs (ZK Proofs). The architecture ensures that all computations are verified on-chain, a process strengthened by the key switching mechanism. Additionally, our scheme features a novel method for generating multiple public identities (addresses) from a single set of private data (private key).

As next steps, we prioritize the implementation of this scheme and subsequent usability testing to determine if the benefits can outweigh the inherent longer transaction times. Given that the primary target audience for this project includes large financial institutions, scalability has not been a primary concern. However, for applications requiring scalability, integrating the approach within a Layer 2 solution [8], could enhance this aspect.

11 Contributions

11.1 Ishank

- Played a pivotal role in identifying numerous errors in the proposed schemes and successfully rectified all of them.
- Worked on range proofs and change of base sections.
- Met weekly and contributed significantly to the presentation and final report.

11.2 Paulo

- Idealized the initial scheme
- Studied possible areas of improvement on existing projects/papers
- Created the address generating protocol
- Led meetings every week and contributed significantly to the presentation and final report.

11.3 Helen

- Worked on signature scheme for proving ownership of an asset to the network.
- Researched existing related work and background information to provide necessary context for others.
- Met weekly and contributed significantly to the presentation and final report.

References

[1] *Blockchain Facts: What Is It, How It Works, and How It Can Be Used*. Investopedia. URL: https://www.investopedia.com/terms/b/blockchain.asp (visited on 05/13/2024).

[2] *Central Bank Digital Currency Tracker*. Atlantic Council. URL: https://www.atlanticcouncil.org/cbdctracker/ (visited on 05/13/2024).

[3] *Monero*. getmonero.org, The Monero Project. URL: https://www.getmonero.org/get-started/what-is-monero/index.html (visited on 05/12/2024).

[4] *What is Aztec? — Privacy-first zkRollup — Aztec Documentation*. URL: https://docs.aztec.network/learn/about_aztec/what_is_aztec (visited on 05/13/2024).

[5] *fhEVM — Zama — Confidential Smart Contracts*. URL: https://www.zama.ai/fhevm (visited on 05/13/2024).

[6] *An incomplete guide to stealth addresses*. URL: https://vitalik.eth.limo/general/2023/01/20/stealth.html (visited on 05/13/2024).

[7] Chainalysis Team. *2024 Crypto Crime Trends: Illicit Activity Down as Scamming and Stolen Funds Fall, But Ransomware and Darknet Markets See Growth*. Accessed May 12, 2024. 2024. URL: https://www.chainalysis.com/blog/2024-crypto-crime-report-introduction/ (visited on 05/12/2024).

[8] Vitalik Buterin. *Different Types of Layer 2s*. https://vitalik.eth.limo/general/2023/10/31/l2types.html. Accessed May 13, 2024. Oct. 2023.