Practical usage of Zero-knowledge in different use-cases in blockchains

Lukáš Častven
Faculty of informatics and information technologies
Slovak university of technology
Bratislava, Slovakia
xcastven@stuba.sk

***Abstract*—Zero Knowledge Proofs (ZKP) are a cryptographic primitives which enables a validation of data without revealing the data itself, making it ideal for secure and private applications in blockchain networks. The focus of this study is a design and an implementation of a proof of concept Stealth Address scheme using ZKPs. This scheme will allow any sender to derive a stealth address from recipients public data. Only the recipient has control over this address, yet it does not leak any information about who the recipient is.**

***Index Terms*—cryptography, zero knowledge, privacy, cryptocurrency**

I. Introduction

Zero Knowledge Proofs (ZKPs) are a powerful cryptography primitive. They allow for verification of a statement's truth without disclosing or in any way revealing the actual content of the statement. This characteristic is crucial for maintaining trust between parties while also preserving privacy [1].

The concept of ZKPs was first introduced in a 1989 research paper, "The Knowledge Complexity of Interactive Proof Systems." [2]. This work describes how in traditional proofs, such as demonstrating a graph is Hamiltonian, more information is typically revealed than just the truth of the theorem. This paper develops a computational complexity theory focusing on the knowledge part within a proof. It introduces zero knowledge proofs, a novel concept where proofs only confirm the correctness of a proposition without exposing any extra knowledge.

The paper focuses on interactive proofs, where a dialogue between a prover and a verifier occurs. In these interactive proofs, the prover aims to convince the verifier about the truth of a private statement, with a very small probability of error.

With interactive proofs one can convince a probabilistic polynomial-time verifier of a *PSPACE* statement's truth [3], [4]. These proofs are pivotal in ZKPs, as they allow for the verification of a statement's truth without revealing the actual information or knowledge behind the statement, maintaining the principle of conveying no knowledge beyond the proposition's correctness.

However, these interactive proofs are impractical for real-world applications, as they require a prover to be online and interact with each verifier. Thanks to Fiat and Shamir [5] this limitation was overcome, and non-interactive ZKPs were introduced. These non-interactive proofs are generated by a prover and can be verified by a verifier without any interaction.

To transform a statement into a ZKP, different ZKP systems can be used. The most commonly used ZKP systems are called SNARKs (Succinct Non-interactive ARguments of Knowledge) [6]. This system creates a succinct proofs that can be verified in a very short time by a computationally bounded verifier.

This thesis extends the application of ZKPs to the concept of stealth addresses in blockchain. Such as those outlined in Vitalik Buterin's article "An Incomplete Guide to Stealth Addresses." [7], or in the Peter Todd's proposal to Bitcoin mailing list [8]. Stealth addresses are critical for privacy on blockchains, allowing assets to be transferred without revealing the recipient's identity and making it difficult to link transactions to specific individuals.

Stealth addresses allow a sender (Alice) to transfer assets to a receiver (Bob) without publicly revealing Bob's identity. To achieve this, Bob must first provide a public stealth address generation data. This data has different structure based on the underlying stealth address generation schema. In case of this study and Stealth Address scheme using ZKPs, from this data Alice creates a new stealth address that only Bob can control, and sends the assets to that newly generated address. Bob can then access these assets from another address, only by providing a ZKP of given address ownership.

II. Background

This section begins with a demonstration of interactive proofs with goal to build up intuition behind interactive proofs [2], [9].

A. Demonstration of an interactive zero knowledge proof

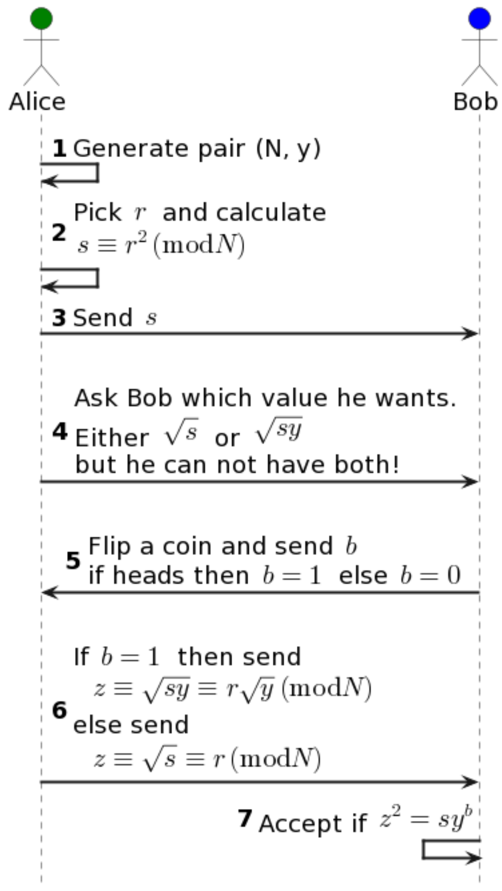
This section explains how Alice attempts to prove to Bob that she knows an algorithm, with which she computes some pair (N, y), such that this pair is part of the quadratic residue language QR. Specifically, Alice needs to convince Bob that there exists an x, such that y equals x squared modulo N, effectively placing the pair (N, y) within the QR language, which includes all pairs where y is a quadratic residue of N.

Alice aims to prove to Bob that she knows an algorithm, which computes some pair (N, y), such that this pair is part of the quadratic residue language QR [2], where QR is defined as:

$$QR = \{(N, y) : \exists x, y \equiv x^2 \pmod N\}$$

The algorithm is as follows [2]:

1) Alice generates pair (N, y),
2) Alice picks a random r such that $1 \leq r \leq N$ and $\gcd(r, N) = 1$, and calculates $s \equiv r^2 \pmod N$,
3) Alice sends Bob s,
4) Alice asks Bob which value he wants. Either $\sqrt{s}$ or $\sqrt{sy}$, but he can not have both,
5) Bob flips a coin and sends b such that if coin landed on heads $b = 1$ else $b = 0$,
6) If $b = 1$ Alice sends to Bob $z \equiv \sqrt{sy} \equiv r\sqrt{y} \pmod N$ else she sends $z \equiv \sqrt{s} \equiv r \pmod N$,
7) Bob accepts if $z^2 = sy^b$.



Alice
Bob
1 Generate pair (N, y)
2 Pick r and calculate $s \equiv r^2 (\text{mod} N)$
3 Send s
4 Ask Bob which value he wants. Either $\sqrt{s}$ or $\sqrt{sy}$ but he can not have both!
5 Flip a coin and send b if heads then $b=1$ else $b=0$
6 If $b=1$ then send $z \equiv \sqrt{sy} \equiv r\sqrt{y} (\text{mod} N)$ else send $z \equiv \sqrt{s} \equiv r (\text{mod} N)$
7 Accept if $z^2 = sy^b$

Fig. 1. Interactive proof of language QR [2], [9]

If Alice was a cheating prover, and she did not have the algorithm for generating pairs from QR, then the probability that Bob's coin toss favors Alice is one half. With one half probability Bob would ask cheating prover Alice to give him the equation she can not solve, because if the prover is cheating, she can not find the $\sqrt{s}$ and $\sqrt{sy}$. If she could, that would mean that she is not cheating.

If the Alice's claim is true, Bob will accept. If Alice is not honest, and cheats, all verifiers will not accept with probability $P(Accept) = 0.5$. But this probability may not be satisfying enough. To make the probability that Alice is cheating smaller, Bob and Alice can start the interaction once again. This would lead to $P(Accept) = (0.5)^2$. They can redo the process as many times as they wish, resulting in $P(Accept) = (0.5)^k$ where k is how many different interactions they performed.

Thanks to the randomness of the coin toss, there are 2^k possibilities how the interaction can go. Since Alice can not reliably predict what the random coin toss will yield, she must be ready to provide both equations. Thus Bob is convinced, that Alice is not cheating, with probability $P(Accept) = (0.5)^k$, and can accept the proof.

B. *Overcoming limitation of an interactive proof*

Interactive proofs require Alice to engage in a unique interaction with each individual verifier, which is not scalable or feasible for widespread application. Many ZKP protocols only require from the verifier a random input (for instance, a coin toss). Protocols, in which the verifier's role is generating some randomness and making it public are called public coin protocols [10], [11]. The paper ”How To Prove Yourself: Practical Solutions to Identification and Signature Problems” [5] by Fiat and Shamir demonstrates how these interactive public coin protocols can be efficiently transformed into non-interactive ones, offering a more scalable and practical solution for ZKPs.

To transform a interactive public coin protocol into a non-interactive, Alice uses a random oracle which can provide a random coin toss based on some input. In practice the source of randomness of the random oracle is a cryptographic hash function, such as SHA256. Instead of sending messages back and forth between Alice and Bob, Alice provides to Bob a transformed transcript of the interaction. Since Bob's role in this interaction would only be generating random coin toss, Alice can in advance query the random oracle for this random value, and supply the message as an input to the query. The transcript string would look like this $(msg1, query(msg1), msg2, ...)$, where $query$ is the output from random oracle. This string and the public input of the proof can then be published and anybody, not just Bob, can validate this proof on their own.

To illustrate, in scenario 2, where Alice sends only two messages and requires one random coin toss from Bob, Bob at the end can compute the validity of the proof from Alice with $validate(public\ input\ x,\ msg1,\ coin\ toss,\ msg2) = accept/reject$ Then after applying Fiat-Shamir transform 3, Alice only sends one message with the transcript string, and Bob can compute the validity of the proof like this $validate(public\ input\ x,\ msg1,\ query(msg1),\ msg2) = accept/reject$.

Alice can publish the transactipt string somewhere public, and anybody can rerun the interaction with the string in order to verify that Alice is lying or not.

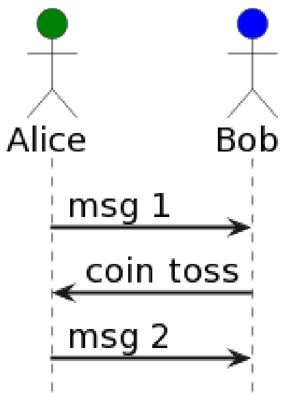


Fig. 2. Interactive public coin proof
[9], [11]

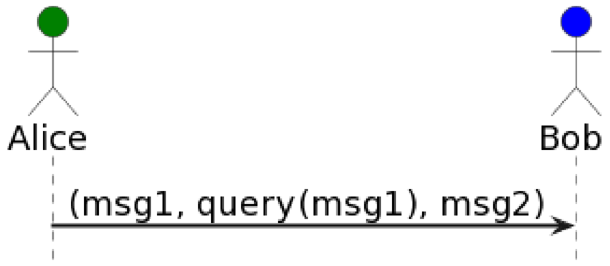


Fig. 3. Non-interactive public coin proof
[5], [9]

C. Arithmetic circuit

While in theory any NP statement [1] can be proven using interactive proofs, practical implementation requires specific definition and encoding of the statement. There are two main models of general computation, those are circuits and turing machines. To trace the computation of a turing machine, the representation needs to somehow handle memory and thus would accrue more complexity than if a circuit is used. To represent a statement as a circuit, an arithmetic circuit, a computation model composed of addition and multiplication operations, is used. This circuit encodes the statement into a form suitable for "zk-ifying", enabling the application of interactive proofs to a broader range of practical scenarios.

Arithmetic circuits in a finite field $\mathbb{F}_p$ are used in majority of implementations. The arithmetic circuit is a function which takes n elements from field $\mathbb{F}_p$ and returns one element from that field.

$$AC : \mathbb{F}^n \rightarrow F$$

The AC can be represented as a directed acyclic graph, or a polynomial. For example, polynomial $x_1x_2 + (x_2 + x_3)^2$ represents the same circuit as this directed acyclic graph

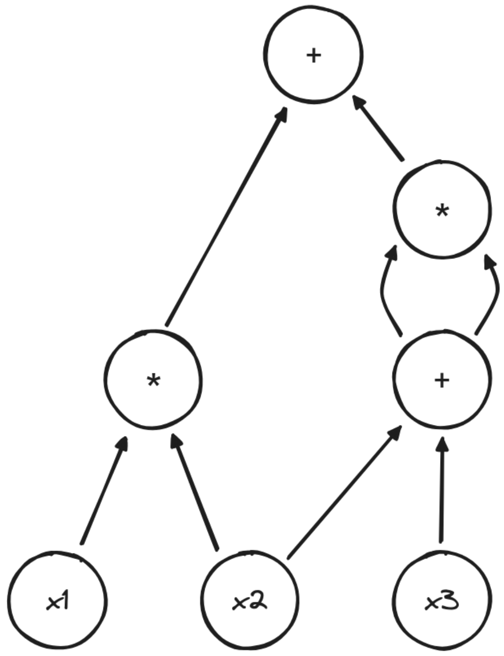


Fig. 4. Graph representation of arithmetic circuit
[12]

Another representation of an arithmetic circuits is called a Rank 1 Constraint System (R1CS). R1CS is a system of equations of a form $\alpha \times \beta = \gamma$, where α, β, γ are affine combinations of variables w and x, where w is a witness (private inputs) and x is a public inputs.

These are some examples of R1CS equations:

$$\begin{aligned}&(w_1 + x_3) \times (w_2 - x_1 + 1) = w_3\\&w_1 \times w_1 = x_1\\&w_1 \times x_2 = w_3\end{aligned}$$

And here is an example of a invalid R1CS equations, because they are not linear combinations of variables w and x:

$$\begin{aligned}&w_1 \times x_3 \times w_2 = w_3\\&w_1 \times w_1 \times w_1 = x_1\\&w_1 \times x_2 + w_3 = w_4 \times w_5\end{aligned}$$

To constrain the operation $w_1^3 = x_1$ in R1CS, there must be two equations, with a new intermediary variable w_2:

$$\begin{aligned}&w_1 \times w_1 = w_2\\&w_1 \times w_2 = x_1\end{aligned}$$

D. SNARKs

After defining an arithmetic circuit, it can be transformed into a SNARK, a Succinct Non-interactive ARgument of Knowledge. The succint part means that the size of the proof must be sublinear in size of witness (when talking about strong succinctness, the proof size is logarithmic in size of circuit), and the verification must be sublinear in size of circuit and linear in size of public input (when talking about strong efficiency, the verification time is logarithmic in size of circuit and linear in size of public input). A SNARK is a tripple of algorithms $(Setup, Prove, Verify)$ [6].

1) $Setup$ takes the arithmetic circuit $C(x, w) \longrightarrow \mathbb{F}$, where x is a public input (from $\mathbb{F}^n$) and w is a witness (from $\mathbb{F}^m$), and is preprocessed, which creates public parameters pp (prover's parameters) and vp (verifier's parameters). This setup process is what enables the proof to be logarithmic in size of circuit. It is a summary of the circuit, so the verifier does not need to know the whole circuit, just the summary,
2) $Prove$ algorithm takes the prover's parameters pp, public input x, witness w and creates a proof π which proves that $C(x, w) = 0$.
3) $Verify$ algorithm takes the verifier's parameters vp, public input x and proof π and returns $accept/reject$.

SNARKS combine two cryptographic primitives, a functional commitment scheme and an interactive oracle proof, in order create and verify the proof. Functional commitment scheme is a cryptographic primitive, which allows the prover to commit to a function f and the verifier can query the commitment and prover must provide the evaluation of the function f at the queried point, and a proof that the evaluation is correct. Interactive oracle proofs begin with a vector of function commitments. Verifier can interactively query any of the commitments and prover must provide the evaluation of the function at the queried point, and a proof that the evaluation is correct [13].

E. Stealth addresses

The concept of stealth addresses was introduced in 2014 by Peter Todd [8]. They were based on elliptic curve cryptography and enabled senders to send funds to a recipient without leaking the recipient's identity to the public. Only the recipient can then prove that they own the stealth address, and can spend the funds from it. Recipients must publish a meta stealth address, from which senders can derive a new stealth address. This new stealth address is not be linked to the meta stealth address, and thus the recipient's identity is protected.

This sutdy showcases how to swap elliptic curve cryptography for ZKPs. They can be used to prove that the recipient owns the stealth address, and thus can spend the funds. The proof can be sent from any address, but only the recipient can generate it, because only he/she knows the private data that is required to generate the proof. The zero knowledge property of the proof ensures that the recipient's identity is not revealed.

III. Related Work

With the growth of cryptocurrencies, ZKPs have gained a substantial popularity not only in academic circles, but also in the startup and venture capital ones. Many researchers are studying this field and it is not considered as a cryptography for nerds anymore.

On the other hand, there are not many contributions to the stealth address ecosystem. There are two protocols on Ethereum that implement stealth addresses. First one is called Umbra [14], and it is based on elliptic curve cryptography.

The second one is called Nocturne [15]. The protocol is a mix of elliptic curve cryptography and ZKPs with few intermediate smart contracts. In this protocol you lock your funds in the Nocturne ecosystem, in which new stealth addresses are created with Elliptic curve stealth addresses scheme, and you access your funds by submitting ZKPs. However according to their Twitter post they are discontinuing their protocol.

IV. Solution Design

The main idea behind the solution is that both receiver and sender generate a random value. These two values can then be used to prove to a stealth address that whoever owns these values is the owner of the stealth address, and can control it.

Bob, as a receiver, publishes the hash of his random value to a public registry. With this hash he also publishes his public key, which corresponds to his private key. This tuple is Bob's meta stealth address 5.

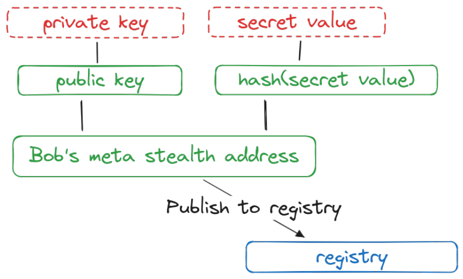


Fig. 5. Bob's Meta Stealth Address

When Alice wants to send funds to Bob, she finds his meta stealth address in the public registry, generates her own random value and hashes it together with Bob's hash to create a code. Then she deploys a new stealth wallet contract with this code in it, and amount of funds that she wanted to send to Bob. After that, she encrypts her random value with Bob's public key, this encrypted value is called ephemeral key. Alice publishes it to a public registry 6.

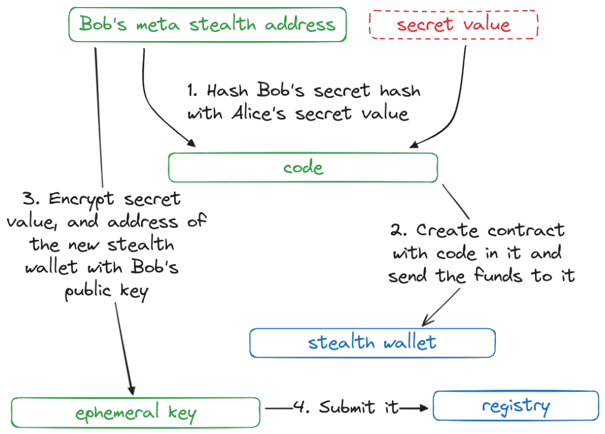

Bob's meta stealth address
secret value
1. Hash Bob's secret hash with Alice's secret value
code
3. Encrypt secret value, and address of the new stealth wallet with Bob's public key
2. Create contract with code in it and send the funds to it
stealth wallet
ephemeral key
4. Submit it
registry

Fig. 6. Alice sends funds

Bob then scans the registry, tries to decrypt ephemeral keys. When the decryption is successful, Bob can save the decrypted Alice's secret value and the address of the corresponding stealth wallet.

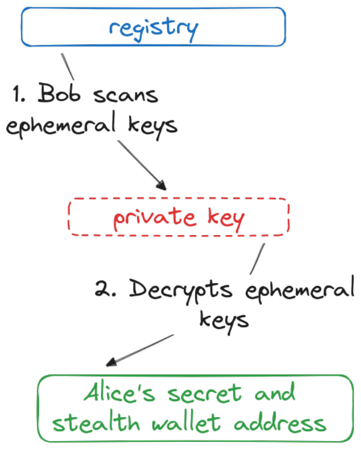

registry
1. Bob scans ephemeral keys
private key
2. Decrypts ephemeral keys
Alice's secret and stealth wallet address

Fig. 7. Bob scans ephemeral keys

To use the funds, Bob must generate a ZK proof and submit it to the stealth wallet. This proof proves that Bob knows Alice's random value and his own random value, such that

$$code = hash(hash(Bob's\, value),\ Alice's\, value)$$

where $code$ is the one submitted by Alice into the stealth wallet contract.

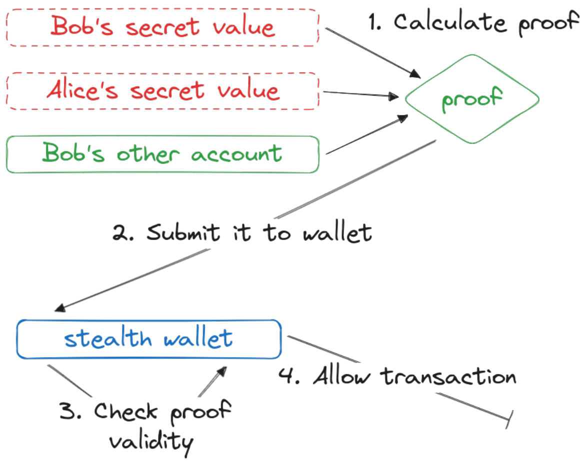


Fig. 8. Bob's interaction with wallet

To prevent others from copying the proof when Bob sends it in a transaction, and gaining control over wallet with a higher fee transaction, the proof must also verify the sender of the transaction. In the proof, there must be a check that the address with which Bob wants to interact with the stealth wallet is the one sending the transaction. Without this check a malicious actor could copy the proof and send a transaction with higher fee, which would give him access to the wallet sooner, because it would be placed higher in the block. Since only Bob knows both secret values and an address from which he will be sending transaction, only he can create a valid proof and thus interact with the wallet.

V. Implementation

In this section different terminology is used, instead of Bob, owner is used. And instead of Alice sender is used. The implementation of this study is separated into 3 modules 9, module for the circuit which will compute ownership proofs, another one for smart contracts needed to implement stealth wallet with ZKP stealth address schema, and module with a web app wallet client used to interact with the stealth wallets.

The implementation is only a proof of concept, not a full featured wallet. The sender's part is the same as described until now. The owner however, can only withdraw all funds from the wallet into another address. This does not detract from demonstrating the core concept of a ZKP stealth address schema. Fractional withdrawals, ERC20 token transfers, transaction proxying and other wallet features can be implemented in future work.

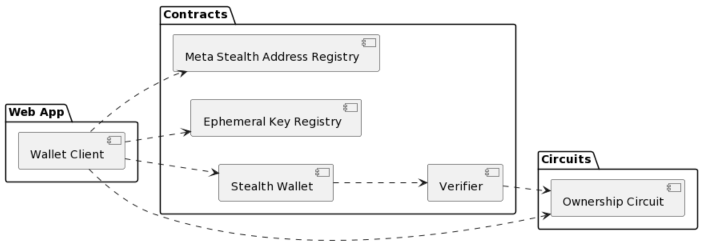


Fig. 9. Component diagram

A. Circuits

Circuits for proving and verifying ZKPs are written in Circom 2 [16]. With circom compiler, a circuit is compiled into a Groth16 ZK-SNARK proof system [6] representation. Circom also generates either C++ of Wasm source files for computing witness (proof) for the circuit. Circuit written in this study, which proves ownership, has these private inputs:

1) owner's secret value,
2) sender's secret value,
3) and the withdrawee address.

And these public inputs:

1) code that was submitted by sender on contract creation,
2) and the transaction sender.

Groth16 requires a trusted setup for each circuit. For Groth16 trusted setup, there are two parts:

1) Powers of Tau ceremony [18],
2) and phase 2 dependent on the circuit.

Powers of Tau ceremony is used to generate initial parameters for a SNARK. In this ceremony, the initial parameters are generated using a multi-party computation. If only one party in this computation is acting honestly, the entire process is trustworthy and the initial parameters are secure for use with any circuit. There is ongoing public powers of tau ceremony called Perpetual Powers of Tau [19]. Its goal is to securely generate SNARK parameters for circuits. In this study a powers of tau ceremony file from SnarkJS [17] is used, which is from this ceremony. Phase 2 is generated with SnarkJS [17] and is specific for circuit.

B. Smart contracts

Smart contracts in this implementation are written in programming language Solidity [25] and the are developed and tested with Foundry [23]. They are deployed through Infura [22] Ethereum RPC provider to the Sepolia Ethereum testnet [24].

There are four smart contracts implemented. A meta stealth address registry which holds meta stealth addresses. A ephemeral key registry which holds the ephemeral keys. A verifier of previously mentioned circuit. And contract for the stealth wallet.

Owners submit their meta stealth addresses to meta stealth address registry, and senders query it for them. To the ephemeral key registry senders submit ephemeral keys and owners query it for them. Senders deploy stealth wallets with funds and computed codes. Owners send proofs to stealth wallets, which interact with the verifier.

C. Web app

Wallet client used to interact with stealth wallets is written in frontend framework SolidJS [20], with Web3JS [21] as the main library for interacting with blockchain, and Infura [22] is used as a Ethereum RPC provider.

It consists of two sub-pages, one for sender (Alice), in which a sender can query meta stealth addresses and deploy stealth wallets only accessible by the owners of corresponding meta stealth addresses. And the second one, for owners (Bob) which displays all stealth wallets with balance and enables owners to withdraw funds from them by generating ZKPs.

VI. Future work

This study opens up few avenues for future research and improvements to this proof of concept:

1) Adding functionality as mentioned in V,
2) How to setup initial parameters of a circuit to achive trustworthiness and security,
3) Formal proof of security and soundness of proposed solution,
4) Smart contract optimizations via gas optimizations in the Solidity code,
5) Exploring different ZKP proof systems with comparisons and benchmarks,
6) Incorporate recoverability into stealth wallets.

VII. Conclusion and Future Work

This study investigated the application of Zero-Knowledge Proofs (ZKPs) in stealth addresses schema on Ethereum. The primary research goal was to design and implement a proof of concept stealth address schema which utilizes ZKPs to achieve privacy of transactions. The solution aimed to protect a recipient's identity (Bob) while enabling a sender (Alice) to transfer funds without any way of identitying the recipient.

The implementation demonstrates the core concept of a ZKP-based stealth address schema. Senders deploy stealth wallets associated with a code derived from a combination of their secret value and the recipient's hashed secret. The recipient can generate ZKPs, proving knowledge of both secret values, to withdraw funds from their corresponding stealth wallets. The design also ensures that proofs can only be valid when submitted by the specific recipient who owns both secret values. It is important to note that the current implementation serves as a proof of concept, and further development would be necessary for real-world deployment as a full-featured wallet.

In conclusion, this research work demonstrates the feasibility and advantages applying ZKPs into stealth address schema. The privacy protection provided by ZKPs in this context highlights a promising direction for strengthening anonymity in cryptocurrency transactions.

References

[1] O. Goldreich, S. Micali, and A. Wigderson, "Proofs that yield nothing but their validity or all languages in NP have zero-knowledge proof systems," Journal of the ACM (JACM), vol. 38, no. 3, pp. 690–728, 1991.

[2] S. Goldwasser, S. Micali, and C. Rackoff, "The Knowledge Complexity of Interactive Proof Systems," SIAM Journal on Computing, vol. 18, no. 1, pp. 186–208, Feb. 1989, doi: 10.1137/0218012.

[3] A. Shamir, "IP = PSPACE," Journal of the ACM, vol. 39, no. 4, pp. 869–877, Oct. 1992, doi: 10.1145/146585.146609.

[4] C. Lund, L. Fortnow, H. Karloff, and N. Nisan, "Algebraic methods for interactive proof systems," Journal of the ACM, vol. 39, no. 4, pp. 859–868, Oct. 1992, doi: 10.1145/146585.146605.

[5] A. Fiat and A. Shamir, "How To Prove Yourself: Practical Solutions to Identification and Signature Problems," in Lecture Notes in Computer Science, Springer Berlin Heidelberg, pp. 186–194. doi: 10.1007/3-540-47721-7_12.

[6] J. Groth, "On the Size of Pairing-Based Non-interactive Arguments," in Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2016, pp. 305–326. doi: 10.1007/978-3-662-49896-5_11.

[7] An incomplete guide to stealth addresses — vitalik.eth.limo. https://vitalik.eth.limo/general/2023/01/20/stealth.html. [Online]. Available: https://vitalik.eth.limo/general/2023/01/20/stealth.html

[8] [Bitcoin-development] Stealth Addresses — lists.linuxfoundation.org. https://lists.linuxfoundation.org/pipermail/bitcoin-dev/2014-January/004020.html. [Online]. Available: https://lists.linuxfoundation.org/pipermail/bitcoin-dev/2014-January/004020.html

[9] ZKP MOOC Lecture 1: Introduction and History of ZKP — youtube.com. https://www.youtube.com/watch?v=uchjTIlPzFo&list=PLS01nW3Rtgor_yJmQsGBZAg5XM4TSGpPs. [Online]. Available: https://www.youtube.com/watch?v=uchjTIlPzFo&list=PLS01nW3Rtgor_yJmQsGBZAg5XM4TSGpPs

[10] L. Babai and S. Moran, "Arthur-Merlin games: A randomized proof system, and a hierarchy of complexity classes," Journal of Computer and System Sciences, vol. 36, no. 2, pp. 254–276, Apr. 1988, doi: 10.1016/0022-0000(88)90028-1.

[11] S. Goldwasser and M. Sipser, "Private coins versus public coins in interactive proof systems," 1986. doi: 10.1145/12130.12137.

[12] ZKP MOOC Lecture 3: Programming ZKPs — youtu.be. https://youtu.be/UpRSaG6iuks?si=CeuQSD6N8PslfNJa. [Online]. Available: https://youtu.be/UpRSaG6iuks?si=CeuQSD6N8PslfNJa

[13] E. Ben-Sasson, A. Chiesa, and N. Spooner, Interactive Oracle Proofs. Cryptology ePrint Archive, Paper 2016/116, 2016. [Online]. Available: https://eprint.iacr.org/2016/116

[14] Umbra — app.umbra.cash. https://app.umbra.cash/. [Online]. Available: https://app.umbra.cash/

[15] Nocturne — nocturne.xyz. https://nocturne.xyz/. [Online]. Available: https://nocturne.xyz/

[16] Circom 2 Documentation — docs.circom.io. https://docs.circom.io/. [Online]. Available: https://docs.circom.io/

[17] GitHub - iden3/snarkjs: zkSNARK implementation in JavaScript & WASM — github.com. https://github.com/iden3/snarkjs. [Online]. Available: https://github.com/iden3/snarkjs

[18] V. Nikolaenko, S. Ragsdale, J. Bonneau, and D. Boneh, "Powers-of-Tau to the People: Decentralizing Setup Ceremonies," in Applied Cryptography and Network Security, 2024, pp. 105–134.

[19] GitHub - privacy-scaling-explorations/perpetualpowersoftau: Phase 1 of a multi-party trusted setup ceremony for zk-SNARKs — github.com. https://github.com/privacy-scaling-explorations/perpetualpowersoftau. [Online]. Available: https://github.com/privacy-scaling-explorations/perpetualpowersoftau

[20] SolidJS — solidjs.com. https://www.solidjs.com/. [Online]. Available: https://www.solidjs.com/

[21] Web3.js — Javascript Ethereum API — web3js.org. https://web3js.org/. [Online]. Available: https://web3js.org/

[22] Infura. https://www.infura.io/. [Online]. Available: https://www.infura.io/

[23] Foundry Book — book.getfoundry.sh. https://book.getfoundry.sh/. [Online]. Available: https://book.getfoundry.sh/

[24] Networks — ethereum.org — ethereum.org. https://ethereum.org/en/developers/docs/networks/sepolia. [Online]. Available: https://ethereum.org/en/developers/docs/networks/sepolia

[25] Home — Solidity Programming Language — soliditylang.org. https://soliditylang.org/. [Online]. Available: https://soliditylang.org/