Slovak University of Technology in Bratislava

Faculty of informatics and information technologies

Reg. No.: FIIT-16768-116160

Lukáš Častven

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

Bachelor’s thesis

Thesis supervisor: Ing. Kristián Košťál PhD.

May 2024

Slovak University of Technology in Bratislava

Faculty of informatics and information technologies

Reg. No.: FIIT-16768-116160

Lukáš Častven

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

Bachelor's thesis

Study programme: Informatics

Study field: Computer Science

Training workplace: Institute of Computer Engineering and Applied Informatics

Thesis supervisor: Ing. Kristián Košťál PhD.

May 2024

Slovak University of Technology in Bratislava
Ústav počítačového inžinierstva a aplikovanej informatiky

Faculty of Informatics and Information Technologies
Academic year: 2023/2024
Reg. No.: FIIT-16768-116160

STU FIIT

BACHELOR THESIS TOPIC

Student:	**Lukáš Častven**
Student's ID:	116160
Study programme:	Informatics
Study field:	Computer Science
Thesis supervisor:	Ing. Kristián Košťál, PhD.
Head of department:	Ing. Katarína Jelemenská, PhD.

Topic: **Practical usage of Zero-knowledge in different use-cases in blockchains**

Language of thesis: English

Specification of Assignment:

Zero-knowledge proofs (ZKPs) are an exciting breakthrough in applied cryptography that will unlock new use cases across an array of industries, from Web3 to supply chains to the Internet of Things. By verifying the authenticity of information without revealing it, ZKPs can help enhance digital systems' privacy, security, and efficiency. Current use cases are decentralized identity, private transactions, secure and scalable Layer-2s, voting systems, IoT, supply chains, etc. Examine existing solutions and proposals in this domain by conducting state-of-the-art analysis. Propose a system that will utilize Zero-knowledge techniques in some of the existing blockchain networks to enhance its functionality by one of the picked goals. Implement such a solution, which can be done in a smart contract way or as a core network plugin. Potential blockchain networks to use are Polkadot, Kusama, Near, Tezos, Algorand, Moonbeam, Ethereum, etc. Evaluate the implemented solution and compare it with existing ones. Discuss the results and conclude with new findings.

Length of thesis: 40

Deadline for submission of Bachelor thesis:	21. 05. 2024
Approval of assignment of Bachelor thesis:	18. 04. 2024
Assignment of Bachelor thesis approved by:	prof. Ing. Valentino Vranić, PhD. – study programme supervisor

Čestné prehlásenie

Čestne vyhlasujem, že som túto prácu vypracoval(a) samostatne, na základe konzultácií a s použitím uvedenej literatúry.

V Bratislave, 20.5.2024 .....

Lukáš Častven

Anotácia

Slovenská technická univerzita v Bratislave
Fakulta informatiky a informačných technológií
Študijný program: Informatika

Autor: Lukáš Častven
Bakalárska práca: Praktické využitie dôkazov s nulovým vedomím v rôznych prípadoch použitia v blockchainoch
Vedúci bakalárskej práce: Ing. Kristián Košťál PhD.
Máj 2024

Bakalárska práca skúma aplikáciu dôkazov s nulovým vedomím (ZKPs) v blockchainoch. Dôkazy s nulovým vedomím sú kryptografická metóda, ktorá umožňuje overenie dát bez odhalenia samotných dát, čo je ideálne pre bezpečné a súkromné aplikácie v blockchainoch. Táto práca je zameraná na návrh a implementáciu konceptu schémy tajných adries s použitím ZKPs. Táto schéma umožňuje odosielateľom vytvoriť tajnú adresu z verejných údajov príjemcu. Iba príjemca má kontrolu nad touto adresou, a zároveň neodhaľuje žiadne informácie o tom, kto príjemca je.

Výskum zahŕňa analýzu kryptografických princípov za ZKPs, nasledovanú vysvetlením návrhu schémy tajných adries a jej integráciou do blockchainu Ethereum.

Táto práca prispieva do oblasti ukázaním, ako možno využiť ZKPs v blockchaine na riešenie problémov súkromia, ponúkajúc riešenie vo forme konceptuálnej implementácie schémy tajných adries s použitím ZKPs.

Annotation

Slovak University of Technology in Bratislava
Faculty of informatics and information technologies
Degree Course: Informatics

Author: Lukáš Častven
Bachelor's thesis: Practical usage of Zero-knowledge in different use-cases in blockchains
Supervisor: Ing. Kristián Košťál PhD.
May 2024

This bachelor's thesis investigates application of Zero Knowledge Proofs (ZKPs) in blockchains. Zero Knowledge Proofs are a cryptographic method which enables validation of data without revealing underlying data itself, making it ideal for secure and private applications in blockchain networks. Focus of this thesis is design and implementation of a proof of concept Stealth Address scheme using ZKPs. This scheme allows 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.

The research includes an analysis of the cryptographic principles behind ZKPs, followed by an explanation of the Stealth Address scheme's design and its integration into a an Ethereum blockchain.

This thesis contributes to the field by showcasing how ZKPs can be utilized in blockchain to address privacy concerns, offering a solution in the form of a proof of concept implementation of Stealth Address scheme using ZKPs.

Table of contents

List of Figures

List of abbreviations used

AC	Arithmetic Circuit
ERC	Ethereum Request for Comments
EVM	Ethereum Virtual Machine
SNARK	Succinct Non-interactive Argument of Knowledge
R1CS	Rank-1 Constraint System
ZK	Zero Knowledge
ZKP	Zero Knowledge Proof

Chapter 1

Introduction

Zero Knowledge Proofs (ZKPs) are a powerful cryptography primitive. They allow for the verification of 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 [**goldreich1991proofs**].

The concept of ZKPs was first introduced in a 1989 research paper, "The Knowledge Complexity of Interactive Proof Systems." [**Goldwasser1989**]. 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 con-

vince 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 [**Shamir1992**, **Lund1992**]. 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 [**Fiat**] 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) [**Groth16**]. 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. "[**ButerinIncompleteGuide**], or in the Peter Todd's proposal to Bitcoin mailing list [**ToddStealthAddresses**]. 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 meta stealth address generation data. This data has different structure based on the underlying stealth address generation schema. From this data Alice computes 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.

1.1 Document structure

The rest of this thesis is structured as follows. Chapter 2 contains an in-depth analysis of the cryptographic principles that underpin ZKPs, and how they can be utilized to construct a stealth address scheme. Chapter 3 explores related work in the field, highlighting existing research and projects that have contributed to the development of ZKPs and stealth addresses. Chapter 4 describes the design of the solution, explaining the process of how the stealth address scheme is implemented. Chapter 5 discusses the security considerations of the ZKP stealth address scheme, listing the assumptions made and potential vulnerabilities. Chapter 6 outlines the implementation details of the scheme, covering the tools and technologies used.Finally, Chapter 8 concludes the thesis by summarizing the findings and proposing potential avenues for future work.

Chapter 2

Analysis

The analysis section of the thesis starts with demonstration of interactive proofs with goal to build up intuition behind interactive proofs [**Goldwasser1989**, **youtubeMOOCLecture1**]. 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.

The analysis section continues by highlighting a practical limitation of interactive proofs in real world cryptography. It notes that for Alice to prove something to multiple parties, she would need to engage in separate interactions with each one. This approach is not scalable and becomes impractical for widespread verification needs. However, the thesis introduces the Fiat-Shamir transform [**Fiat**], a significant breakthrough that addresses this issue. This transform allows for converting the interactive proof into a non-interactive format by processing the interaction transcript, making the proof

more practical and scalable for real-world cryptographic applications.

While in theory any NP statement [**goldreich1991proofs**] 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 a 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.

The analysis section then explores SNARKs (Succinct Non-interactive ARguments of Knowledge), which are most commonly used ZKP system today. They are succinct, meaning that the size of the proof is small and have a relatively fast verification time.

The final part of the analysis examines how ZKP systems such as SNARKs enable the creation of a stealth address scheme that upholds privacy and security. This section assesses how these systems fulfill the necessary properties for a stealth address scheme, focusing on their ability to ensure transaction confidentiality while maintaining the anonymity of the recipient’s identity.

2.1 Proof of quadratic residuosity

This first part focuses on demonstrating an interactive proof where 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 [**Goldwasser1989**]. QR is defined as:

$$QR = \{(N, y) : \exists x, y \equiv x^2 \pmod{N}\} \tag{2.1}$$

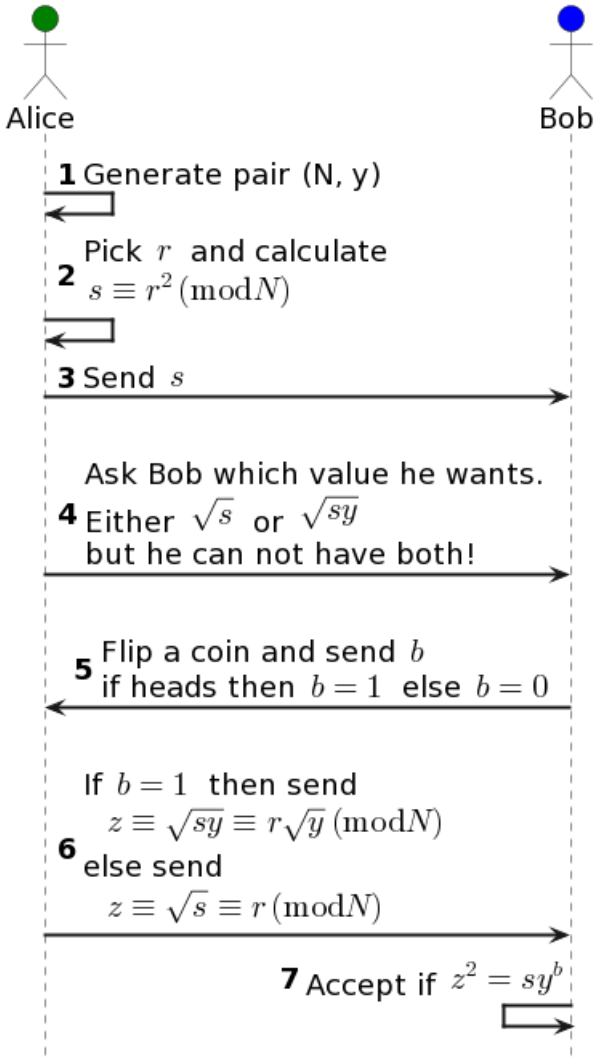

Figure 2.1: Interactive proof of language QR
[**Goldwasser1989**, **youtubeMOOCLecture1**]

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 $z^2 = sy^b$.

If Alice was a cheating prover, and she didn't 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't reliably predict what the random coin toss will yield, she must be ready to provide both equations. Thus Bob is convinced, that Alice isn't cheating, with probability $P(Accept) = (0.5)^k$, and can accept the proof.

2.2 Non-interactive proofs

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 [**Babai1988**, **Goldwasser1986**]. The paper "How To Prove Yourself: Practical Solutions to Identification and Signature Problems" [**Fiat**] 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.

In scenario when 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

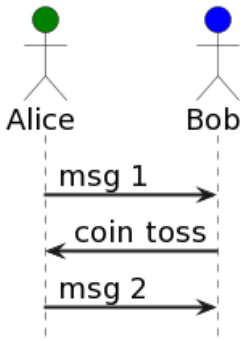

Figure 2.2: Interactive public coin proof
[**Goldwasser1986**, **youtubeMOOCLecture1**]

Alice with $validate(public\,input\,x,\ msg1,\ coin\,toss,\ msg2) = accept/reject$

Then after applying Fiat-Shamir transform, Alice only sends one message with the transcript string, and Bob can compute the validity of the proof like

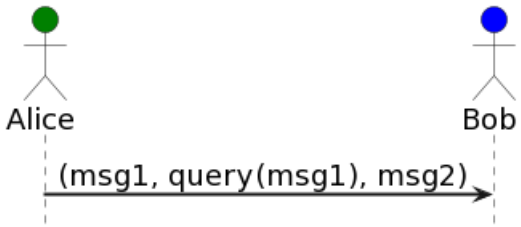

Figure 2.3: Non-interactive public coin proof
[**Fiat**, **youtubeMOOCLecture1**]

this $validate(public\,input\,x,\ msg1,\ query(msg1),\ msg2) = accept/reject$.

2.3 Arithmetic circuit

The computation model used to create ZKPs is an arithmetic circuit in a finite field $\mathbb{F}_p$. The arithmetic circuit is a function which takes n elements from field $\mathbb{F}_p$ and returns one element from that field.

$$AC : \mathbb{F}_p^n \to \mathbb{F}_p \tag{2.2}$$

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

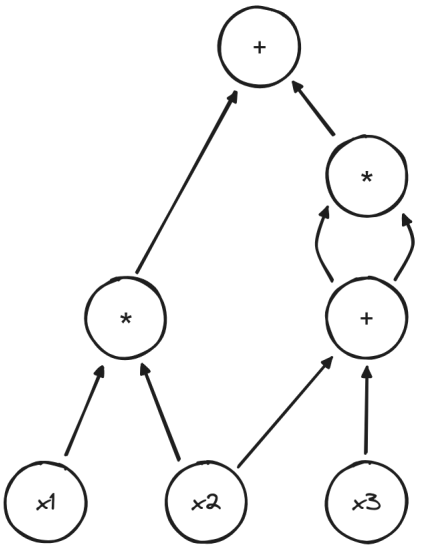

Figure 2.4: Graph representation of arithmetic circuit
[**youtuMOOCLecture3**]

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:

$$\begin{aligned}(w_1 + x_3) \times (w_2 - x_1 + 1) &= w_3 \\ w_1 \times w_1 &= x_1 \\ w_1 \times x_2 &= w_3\end{aligned} \tag{2.3}$$

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

$$\begin{aligned}w_1 \times x_3 \times w_2 &= w_3 \\ w_1 \times w_1 \times w_1 &= x_1 \\ w_1 \times x_2 + w_3 &= w_4 \times w_5\end{aligned} \tag{2.4}$$

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

$$\begin{aligned}w_1 \times w_1 &= w_2 \\ w_1 \times w_2 &= x_1\end{aligned} \tag{2.5}$$

2.4 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 $(Setup, Prove, Verify)$.

2.4.1 Setup

The $Setup$ takes the arithmetic circuit $C(x, w) \to \mathbb{F}$, 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.

This setup procedure has 3 types:

- Trusted setup: the setup is of form $Setup(C, r) \to (pp, vp)$, where r is a random value. The random value must be destroyed after the setup, because if it is not, the prover can use it to prove a false statement,
- Universal trusted setup: the setup is split into two parts. The first part takes only r and creates gp (global parameters). This part is ran once and the r must be destroyed, due to the same reason as in trusted setup. The second part takes gp and C and creates pp and vp. This part can be ran for any circuit C,
- Transparent setup: the setup is of form $Setup(C) \to (pp, vp)$, where C is a circuit. This setup does not require any random value, hence anyone can verify that the setup is valid.

2.4.2 Prove and Verify

The $Prove$ algorithm takes the prover's parameters pp, public input x, witness w and creates a proof π which proves that $C(x, w) = 0$. The $Verify$ algo-

rithm takes the verifier's parameters vp, public input x and proof π and returns $accept/reject$. These algorithms combine two cryptographic primitives, a functional commitment scheme and an interactive oracle proof [**SassonIOPs**], 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.

The interactive oracle proof begins with instantiating vp in the setup phase, such that $vp = (comm_{f1}, comm_{f2}, \ldots, comm_{fn})$, where $comm_{fi}$ is a functional commitment for a function f_i. Verifier can interactively query any of the commitments and prover must provide the evaluation of the function f_i at the queried point, and a proof that the evaluation is correct. Then during the interaction prover and verifier exchange additional function commitments $comm_{f1}, comm_{f2}, \ldots, comm_{fm}$, and random values $r_1, r_2, \ldots, r_m$. At the end verifier computes:

$$Verify^{comm_{f1}, comm_{f2}, \ldots, comm_{fm}, \ldots, comm_{fn}}(x, r_1, r_2, \ldots, r_m) = accept/reject \quad (2.6)$$

The $Verify$ algorithm computes if the proof is valid, based on queried function commitments from vp, and the function commitments which were exchanged during the interaction. Verifier checks if evaluations of committed polynomials are equal to evaluations of the polynomials, which verifier can compute from the public input x, proving that the prover's polynomials are equal to the verifier's polynomials. Elaboration why evaluation at random

point is enough to prove that two polynomials are equal can be found in the following subsection.

To make this process non-interactive, the Fiat-Shamir transform is applied on the random values $r_1, r_2, \ldots, r_m$. So the proof π is a string of

$$(comm_{f1}, FS(r_1), comm_{f2}, FS(r_2), \ldots, comm_{fm}, FS(r_m)) \tag{2.7}$$

Verifier than computes $Verify(vp, x, \pi) = accept/reject$, where π is a transcript of the interaction between prover and Fiat-Shamir supplied random values.

2.4.3 Polynomial zero test

The polynomial zero test is at the heart of the SNARK construction. Given a polynomial $f \in \mathbb{F}_p^{\leq d}[X]$, the probability that a random element $x \in \mathbb{F}_p$ is a root of f is d/p.

$$\Pr[f(x) = 0] \leq \frac{d}{p} \tag{2.8}$$

For example, $p = 2^{256}$ and $d = 2^{32}$, the probability that a random element $x \in \mathbb{F}_p$ is a root of f is 2^{224}. This is a negligible probability, and we can assume that f is zero polynomial with high probability when $f(x) = 0$ for a random $x \in \mathbb{F}_p$.

The generalization of the polynomial zero test for multivariate polynomials is known as *SZDL lemma* [**Schwartz1980**, **Zippel1979**, **Demillo1978**].

This can be used to prove that two polynomials are equal. Given two polynomials f, g, we can evaluate $h = f - g$ at a random point x. If $h(x) = 0$, then

$f(x) - g(x) = 0 \implies f(x) = g(x) \implies f = g$ with high probability.

2.5 Ethereum

Ethereum is a decentralized, open-source blockchain, with possibility of running arbitrary Turing complete code as specified by the EVM specification [**ethereumEthereumVirtual**]. This allows for the creation of smart contracts. Smart contracts are programs stored on the Ethereum blockchain and their functionality can be invoked by anyone, as long as the smart contracts code allows it.

In the Ethereum ecosystem, addresses serve as some sort of an unique identifiers for accounts. Ethereum addresses are 42-character hexadecimal addresses that identify an Ethereum account. These addresses are derived from the last 20 bytes of the hashed public key associated with the private key. This private key is used to control the account. These bytes are written in hexadecimal format and to indicate this explicitly, "0x" is appended.

Ethereum wallets are software applications that store private keys and interact with the Ethereum blockchain, enabling users to send and receive Ether and other assets on the network. They are controlled by associated private keys, which are used to sign transactions.

Transactions on the Ethereum network are how users transfer Ether or interact with smart contracts. Each transaction includes the sender’s and recipient’s addresses, the value being transferred, a transaction fee, and optionally, input data for smart contracts. These transactions are grouped into blocks, which are then added to the blockchain.

2.5.1 Stealth addresses

Since Ethereum is a public blockchain, all transactions are visible to anyone, and all addresses are visible to anyone. This is a problem for privacy, because if someone knows your address, they can see all your transactions.

One solution to this problem is to generate a new address for every transaction. However, if somebody wants to send you some funds, they need to wait for you to generate a new address and send it to them. Which quickly becomes impractical.

The concept of stealth addresses was introduced in 2014 by Peter Todd [**ToddStealthAddresses**]. It enables 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.

For a recipient to receive funds, they need to publish a meta stealth address, from which senders can generate a new stealth address. This new stealth address can not be linked to the meta stealth address, and thus the recipient's identity is protected. Additionally, senders must publish some information, which allows the recipient to gain control over stealth address to which the sender sent funds [**Yu2020**].

ZKPs 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.

Chapter 3

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. This field gets some traction, for example, work done by Fan, Jia and Wang, Zhen and Luo, Yili and Bai, Jian and Li, Yarong and Hao, Yao [**FanJiaWang2019**] introduce a stealth address scheme designed to enhance user privacy in blockchain transactions. The scheme aims to address the limitations of existing solutions, such as the need for users to manage multiple key pairs or engage in additional private communication before each transaction. In their approach, a user only needs to maintain a single key pair for initial certification, simplifying key management and reducing storage costs. The sender creates a one-time transaction address and attaches it to the transaction, while the receiver utilizes their private key to verify the transaction directly from the blockchain, eliminating the need

for a separate private channel. The scheme also offers flexibility for regulation, allowing transactions to be either fully or partially regulated based on security requirements.

Or, Wang [**Wang2023**] introduced the concept of Fast Stealth Addresses (FSAs) to improve the search efficiency of stealth address schemes in blockchain transactions. Their approach aims to overcome the linear search time required in existing schemes by allowing constant recognition time to determine if a block contains a recipient's transactions and logarithmic search time to locate the specific transactions. The authors provide a generic construction of the FSA scheme under subgroup membership assumptions related to factoring and instantiate concrete schemes based on specific number-theoretic assumptions. They also formalize the security model of an FSA scheme and provide provable security analysis.

There are two protocols on Ethereum that implement stealth addresses. First one is called Nocturne [**nocturne**]. 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.

The second one is called Umbra [**umbra**]. Umbra is a protocol on Ethereum that implements stealth addresses using elliptic curve cryptography. In this protocol, the recipient a private keys. The sender generates a random number, encrypts it using the recipient's public key, and then computes the stealth address from the recipient's public key and the random number. The encrypted random number, ephemeral public key, and stealth address are

then published. The recipient can scan these announcements, decrypt the random number using their private key, and check if the derived stealth address matches the one in the announcement. If it does, they can then use their spending key to access the funds.

Kovács and Seres [**Kovacs2023**] conducted an analysis of Umbra's recipient anonymity guarantees on Ethereum and its layer-2 solutions. They identified four heuristics based on user behavior that could compromise the anonymity and linkability of Umbra transactions. These heuristics exploit patterns such as the reuse of registrant addresses, the overlap of sender and receiver addresses, the collection of multiple payments to a single address, and unique transaction fees. The study found that a significant portion of Umbra transactions could be deanonymized using these heuristics, raising concerns about the protocol's privacy in practice. The authors suggest countermeasures such as avoiding address reuse and using different addresses for different on-chain activities to mitigate these risks.

Chapter 4

Solution Design

The core problem this thesis addresses is the inherent lack of privacy in blockchain transactions. While blockchains like Ethereum offer transparency and security, the public nature of transactions exposes user identities and financial activities, raising significant privacy concerns. This work introduces a novel solution to this problem by ZKPs to create a stealth address scheme.

The key novelty and scientific contribution of this thesis lie in the design and implementation of a proof-of-concept stealth address scheme that utilizes ZKPs. This scheme allows users to receive funds without publicly revealing their identities, enhancing privacy in blockchain transactions. The proposed solution enables the creation of stealth addresses derived from the recipient's public data, ensuring that only the recipient can control and access the funds sent to these addresses. The use of ZKPs allows the recipient to prove ownership of the stealth address without disclosing any private information, thus preserving anonymity.

Furthermore, this thesis contributes to the field by demonstrating the practical application of ZKPs in addressing real-world privacy challenges in blockchain technology. By presenting a conceptual implementation of the stealth address scheme, this work showcases the potential of ZKPs to enhance privacy and security in blockchain applications.

The Solution Design section outlines the design of a stealth address scheme utilizing ZKPs. The scenario which is being solved involves Alice (sender), who wishes to send funds to Bob (recipient) discreetly, ensuring no one else can identify Bob as the recipient, and that only Bob can control the funds sent. This part of the thesis details the application of ZKPs to achieve this privacy, allowing Alice to complete the transaction without compromising Bob's identity.

4.1 Requirements

The solution enables a proof of concept stealth wallet. This will not be a full featured wallet, only one satisfying these requirements:

Functional requirements

- **Meta Stealth Address Retrieval:** The solution must enable senders to query and retrieve meta stealth addresses from the registry.
- **Ephemeral Key Exchange:** The solution needs a mechanism for senders to submit ephemeral keys and owners to query them.
- **Stealth Wallet Deployment:** Senders must be able to deploy stealth wallet smart contracts, specifying the code and depositing funds.

- **Zero-Knowledge Proof Generation:** The solution must provide a way for owners to generate ZKPs that demonstrate ownership.
- **Zero-Knowledge Proof Verification:** Smart contracts must be able to verify the ZKPs submitted by owners.
- **Stealth Wallet Withdrawal:** Upon successful ZKP verification, the smart contract must execute a withdrawal, transferring the entire stealth wallet balance to the owner's designated address.

Non-functional requirements

- **Security:** The solution must prioritize the protection of private keys.
- **Privacy:** Stealth addresses and the link between owners and their stealth wallets should be obscured to preserve on-chain privacy.
- **Compatibility:** The solution should be compatible with relevant Ethereum testnet environment, not only a local development environment.

4.2 High Level Overview

The main idea behind the solution is a fact that both receiver and sender generate a private 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, only publishes the hash of his random value. Alice then generates her own random value and hashes it with Bob's hash to create a code which will be submitted to the stealth address contract. After that, she encrypts her random value, and address of the new stealth address contract

with Bob's public key. This is called ephemeral key, and it is published to a public registry. Bob then scans the registry, decrypts the ephemeral key and saves Alice's secret value and the address of the contract with funds from Alice.

When Bob wants to use the funds, he submits a proof to the stealth address contract. This proof proves that Bob knows Alice's random value and his own random value, such that the combination of hash of Bob's secret value and Alice's secret value is equal to the code submitted by Alice into the stealth address contract. Also, to prevent others from copying the proof when Bob sends it in a transaction, and gaining control over the stealth address with a higher fee transaction, the proof must 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 address 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 address.

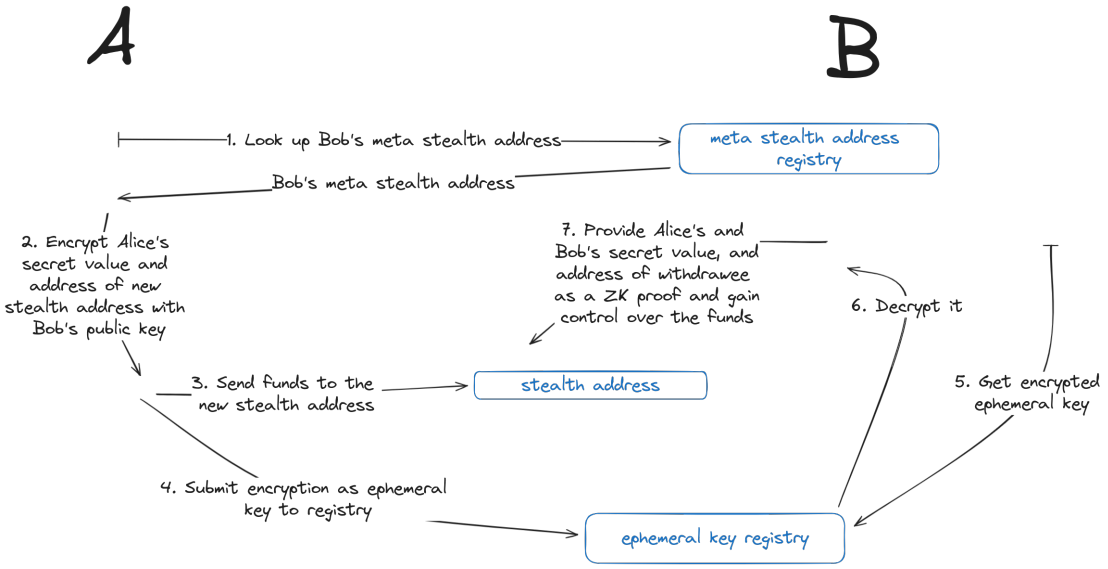

Figure 4.1: High Level Overview

4.3 Initial Setup

Before Alice can send funds to Bob, Bob must first publish his meta stealth address to some public location. Bob computes the following:

- A private key k,
- A corresponding public key K,
- A secret value x,
- A hash of the secret value $h = hash(x)$.

Bob then publishes his meta stealth address in the form of a tuple (K, h).

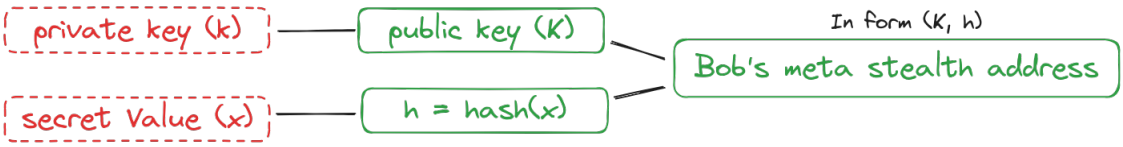

Figure 4.2: Initial Setup
[**ButerinIncompleteGuide**]

4.4 Sending Funds

When Alice wishes to send funds to Bob, she queries meta stealth address registry for Bob's meta stealth address (K, h) and does the following:

- Generate random secret value c,
- precompute address of the stealth address contract sa [**stackexchangeAddressEthereum**
- compute $code = hash(h, c)$,
- deploy new stealth wallet contract with code in it,
- fund the deployed contract,
- encrypt ephemeral key $ek = encrypt(value = [c, sa], key = K)$,
- submit the ephemeral key to registry.

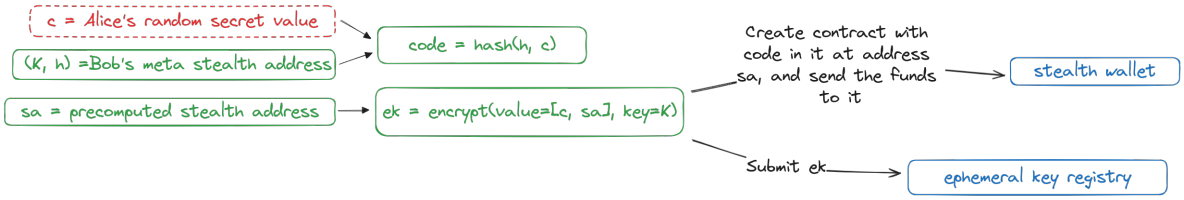

Figure 4.3: Sending Funds
[**ButerinIncompleteGuide**]

4.5 Scanning ephemeral keys

Bob queries ephemeral key registry for keys from the last point he scanned. The registry contract returns a list of ephemeral keys. Bob then tries to decrypt each ephemeral key with his private key k. If the ephemeral key is meant for Bob, then the decryption will succeed, and the decrypted value will contain the secret value c from Alice and the stealth address sa.

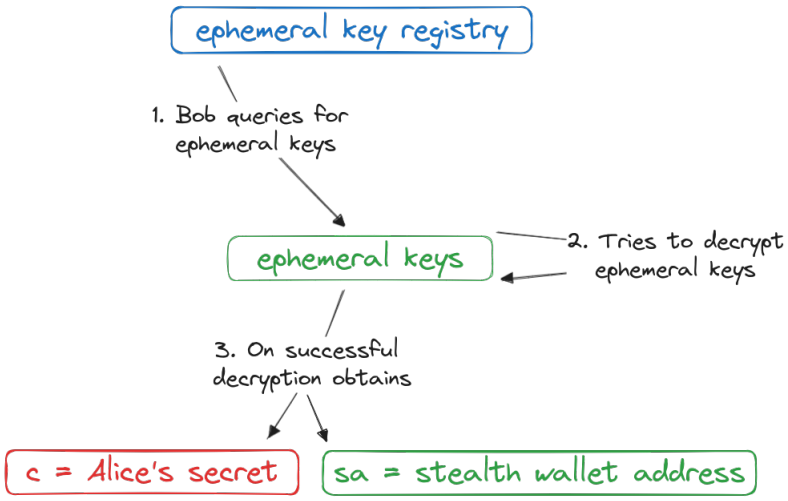

Figure 4.4: Scanning ephemeral keys

4.6 Gaining control of Stealth Addresses

Bob can gain control of the stealth address SA by proving to the stealth address contract that he is the owner of the values x and c, such that $C = hash(h, c) = hash(hash(x), c)$. Bob does this by computing a ZKP for this statement and sending it in a transaction from any address (preferably not from any Bob's publicly known addresses) to the stealth address SA. The SA contract verifies the proof and if it isn't valid, the transaction is rejected.

4.7 Overview

The whole solution design is illustrated in Figure 4.5, and was inspired by the work of Vitalik Buterin [**ButerinIncompleteGuide**].

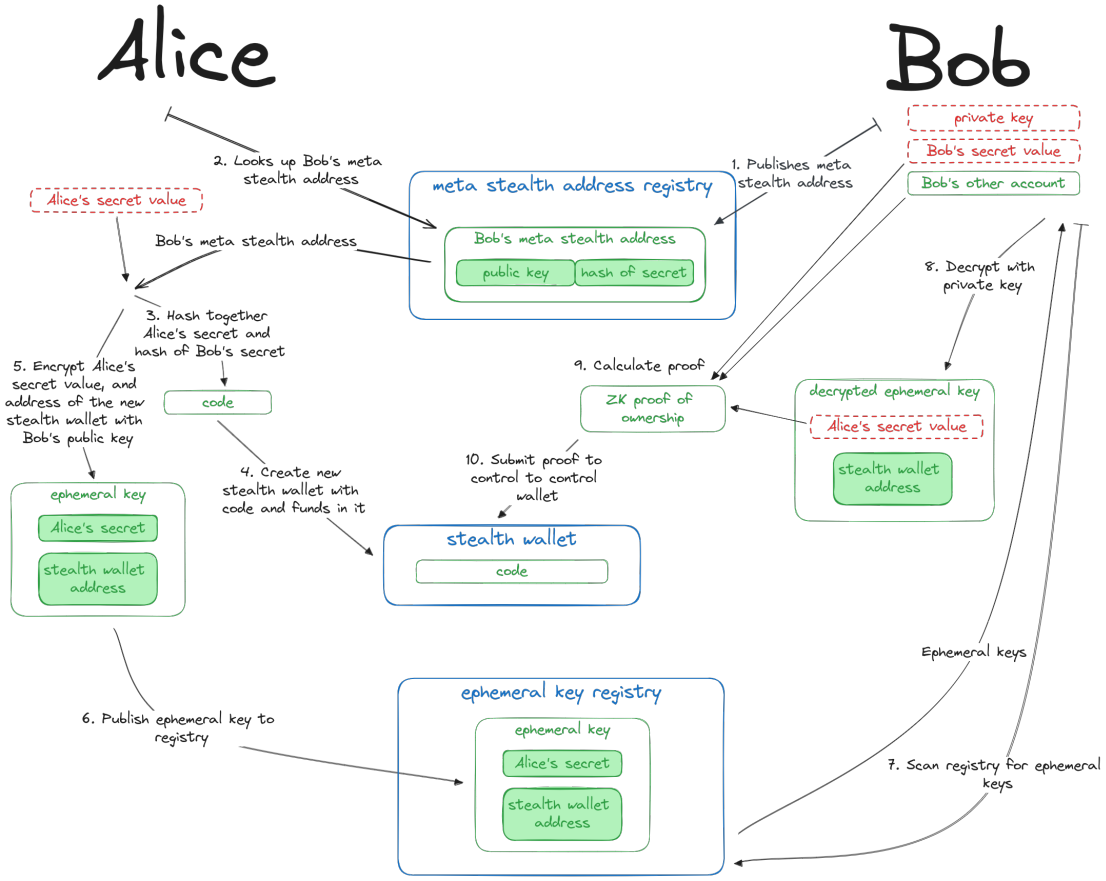

Figure 4.5: Solution Design
[**ButerinIncompleteGuide**]

Chapter 5

Discussion on security

In this chapter, the security of the Stealth Address ZKP Scheme is informally discussed.

5.1 Assumptions

The security of the stealth address scheme relies on several key assumptions. These include the security of the Groth16 ZK-SNARK system, the integrity of the trusted setup process, the collision resistance of the hash function, the robustness of the elliptic curve cryptography used, and the confidentiality of the recipient's and sender's secret values. These assumptions and consequences in a case of their falsehood are explained in next subsections.

5.1.1 Groth16

It is assumed that the pairing-based ZK-SNARK system Groth16 [**Groth16**] has these properties:

1. Completness - honest prover will always convince honest verifier
2. Soundness - dishonest prover will not convince honest verifier
3. Zero Knowledge - dishonest verifier will learn nothing more then the truth of the given proposition

If the completness property would not hold, Bob could be locked out of his stealth addresses, because he could not prove to them that he possessed both secrets.

If the soundness would not hold, adversary could forge a false proof, which would be verified as true, and thus could gain access to Bob's assets on given stealth address.

And finally, if the zero knowledge property would not hold, an adversary could learn Bob's identity, hence it would no longer be a stealth address.

5.1.2 Trusted Setup

The setup phase of Groth16 [**Groth16**] includes a trusted setup in order to generate public parameters. The premise of the trusted setup is that the original parameters used to create the public ones are thrown away, if not then the party which produced public parameters can forge false proofs, yet the verifier would recognize them as valid ones.

In context of this work, if a party creating the circuit didn't delete the original parameters, then it could prove malicious statements about ownership of the secret values needed to gain control over any stealth address.

5.1.3 Collision resistant hash function

First assumption is that, the used instantiation of a hash function (H) is a collision resistant one. Meaning it is computationally hard to find two inputs a, b, such that

$$a \neq b \wedge H(a) = H(b)$$

Otherwise, an adversary could, for example, find a value x', such that Bob's secret value x and x' are not equal, but their hashes are. Then the x' could be used to gain control over Bob's stealth address, if the adversary knew Alice's secret value, or in the case when Alice is a malicious actor, she could send assets to Bob's stealth address, but with x' and her secret value, could withdraw them without any public proof that she did it.

As showcased earlier, this vulnerability alone is not enough to gain control over Bob's stealth address, adversary also must know Alice's secret value to gain control over a Bob's stealth address, but only the one she interacted with.

Also, if used hash function is not a collision resistant one, adversary still can not discover the identity of Bob, or any of his other stealth addresses.

5.1.4 Elliptic Curve Crypthography - Secp256k1

Next assumption is that the Secp256k1 elliptic curve parameters used in Ethereum and Bitcoin [**bitcoinSecp256k1Bitcoin**] are well-chosen and do not contain any backdoors. Additionally, it is assumed that advancements in cryptographic research and technology do not compromise the security of Secp256k1.

If not, an adversary could recover Bob's private key and decrypt all of his ephemeral keys submitted to the Ephemeral Key Registry contract. The adversary would know all Bob's stealth addresses and, also have all secret values used to create codes in those stealth addresses. However, with this vulnerability alone, the adversary can not control Bob's stealth addresses, because he/she is missing the Bob's secret value x.

5.1.5 Knowledge of recipient's secret value

If an adversary somehow learns Bob's secret value x, he/she does not have enough information to control any of Bob's stealth addresses. On the other hand, if Alice, or any other sender, would learn Bob's secret value x, she could, as in the 5.1.3 steal Bob's assets from the stealth address she interacted with and Bob would not have any public proof that it was her.

5.1.6 Knowledge of sender's secret value

If an adversary somehow learns Alice's secret value c, the corresponding address is still safe, because, the attacker does not know which stealth address it is, and if somehow learned it, he/she still does not have Bob's secret value.

5.2 Discussion summary

Given these assumptions, the presented Stealth Address ZKP Scheme can be used to send assets to a recipient, without leaking any information about him/her. And guarantees to the recipient that only he/she has access to stealth addresses.

Chapter 6

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 (depicted in 6.1), 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.

6.1 Interaction between components

The interaction between components, and users is depicted in Figure 6.1.

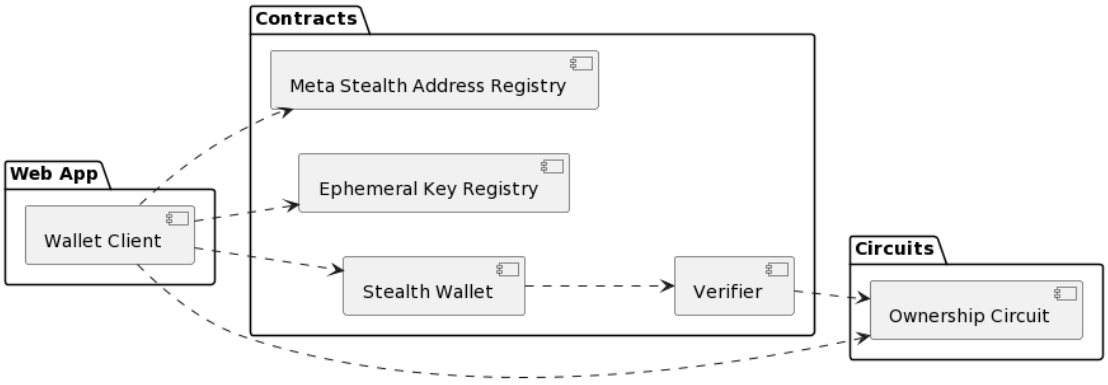

Figure 6.1: Component diagram

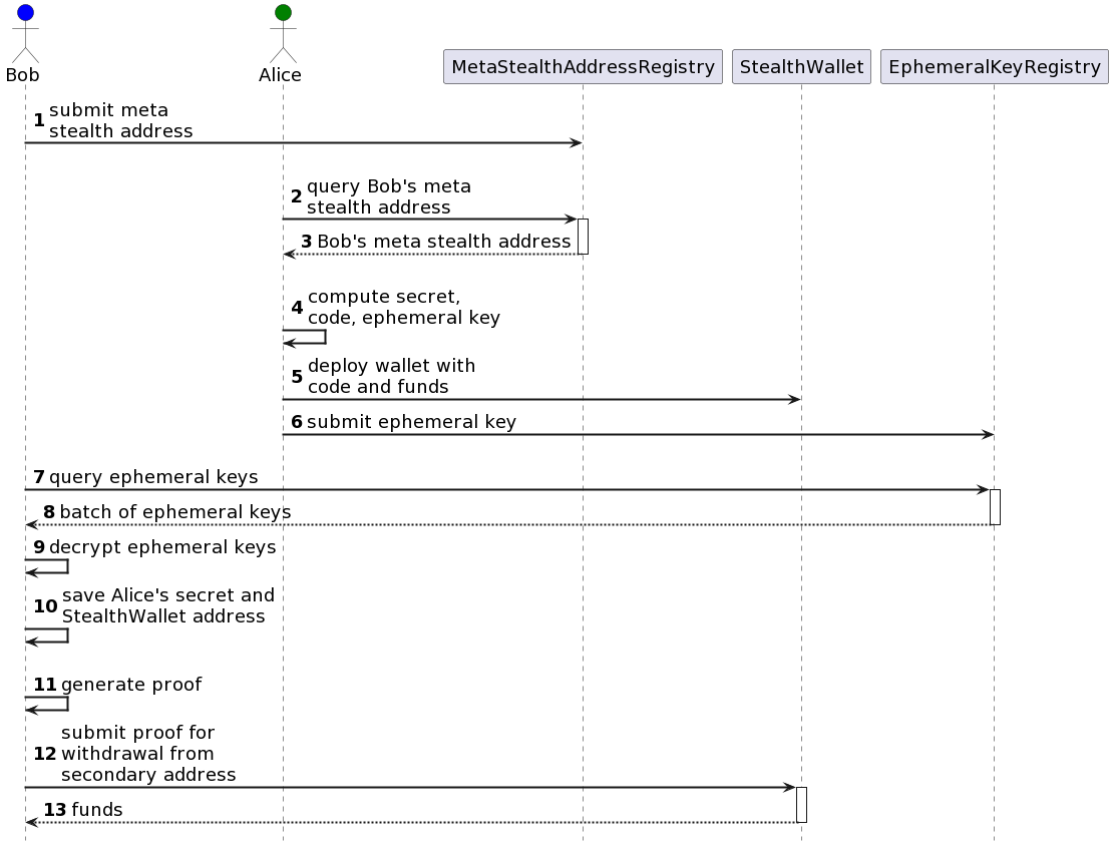

Figure 6.2: Component diagram

6.2 Circuits

Circuits for proving and verifying ZKPs are written in Circom 2 [**circomCircomDocument**
With circom compiler, a circuit is compiled into a Groth16 ZK-SNARK proof system [**Groth16**] 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 [**PowersOfTau**],
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 [**PerpetualPTAU**]. Its goal is to securely generate SNARK parameters for circuits. In this study a powers of tau ceremony file

from SnarkJS [**snarkjs**] is used, which is from this ceremony. Phase 2 is generated with SnarkJS and is specific for circuit.

6.3 Smart contracts

Smart contracts in this implementation are written in programming language Solidity [**solidity**] and are developed and tested with Foundry [**foundry**]. They are deployed through Infura [**infura**] Ethereum RPC provider to the Sepolia Ethereum testnet [**sepolia**].

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.

This module also contains tests. Code coverage report for these is shown in table 6.1.

File	% Lines	% Statements	% Branches	% Funcs
script/DeployConfig.s.sol	0.00% (0/5)	0.00% (0/7)	0.00% (0/2)	0.00% (0/3)
script/DeployContracts.s.sol	0.00% (0/7)	0.00% (0/9)	100.00% (0/0)	0.00% (0/1)
src/EphemeralKeyRegistry.sol	92.86% (13/14)	94.44% (17/18)	66.67% (4/6)	66.67% (2/3)
src/MetaStealthAddressRegistry.sol	100.00% (8/8)	100.00% (11/11)	100.00% (4/4)	100.00% (2/2)
src/StealthWallet.sol	91.67% (11/12)	92.31% (12/13)	83.33% (5/6)	66.67% (2/3)
src/Verifier.sol	100.00% (0/0)	100.00% (0/0)	100.00% (0/0)	100.00% (1/1)
Total	69.57% (32/46)	68.97% (40/58)	72.22% (13/18)	53.85% (7/13)

Table 6.1: Solidity code coverage

The first two files are not covered, because they only contain simple logic for deploying contracts and do not interfere in any functionality of the implemented stealth address scheme.

6.4 Web app

Browser wallet client used to interact with stealth wallets is written in frontend framework SolidJS [**solidjs**], with Web3JS [**web3js**] as the main library for interacting with blockchain, and Infura [**infura**] 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.

Chapter 7

Evaluation

7.1 Requirement satisfaction

How the implementation of the stealth address scheme, as detailed in Chapter 6, fulfills the functional requirements outlined in the Chapter 4, can be seen in Figure 7.1.

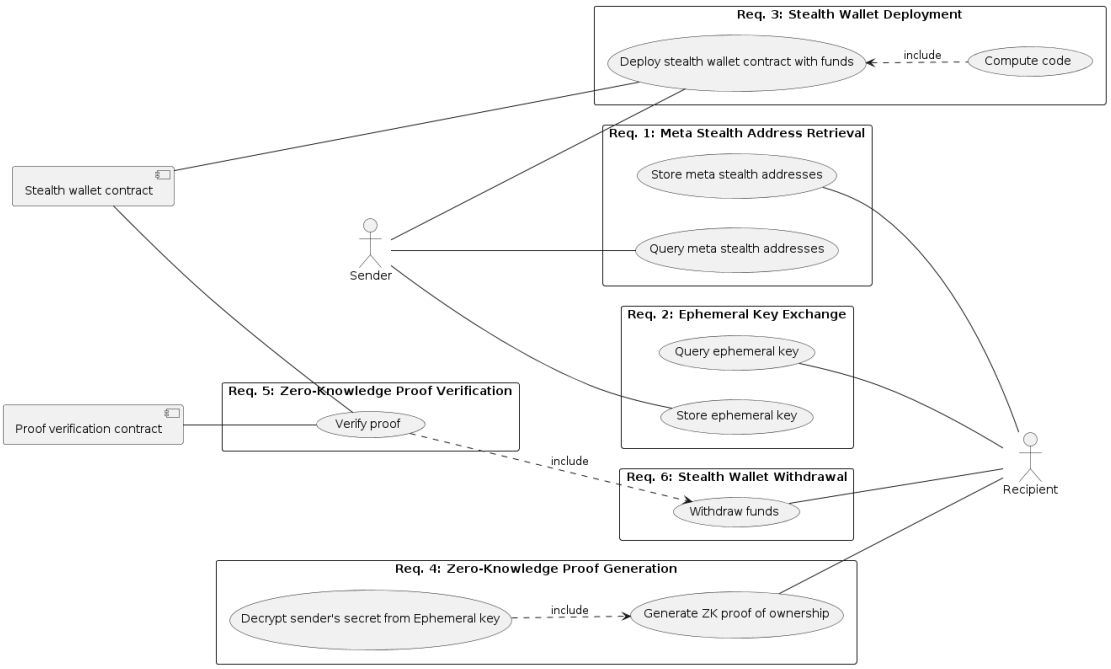

Figure 7.1: Usecase diagram

7.2 Demonstration on Sepolia Testnet

This section walks through a demonstration of the ZKP stealth address scheme deployed on Sepolia testnet. For this demonstration, this will be Alice's address 0x2668Bc59Fa9001DBBA2bD255Bf24b9Fc8c1AbAd7 this is Bob's primary address 0x1c6Ff1028E166C9D4A15a5a019041077793c64D3 and this is Bob's second address 0xc86B622175eDd2274d374dD86e13Ba521e4B876b.

Both Alice's and Bob's secondary addresses have one transaction each, which were only used to fund the address with some Ether for this demonstration. These transactions are disregarded in this demonstration. Bob's primary address contains one funding transaction, and two transactions which were used to setup Bob's meta stealth address (the first transaction was a mistake, that is why there are two). The initial state of both Alice and Bob are

shown in figure 7.2 and 7.3 respectively.

The address of the meta stealth address registry is 0xa08AaF964300121f6C1bD962f7B53C7e06d909BD and the address of the ephemeral key registry is 0x156Bcce17d7409A32C408e2AAb8F04C8CBd7c450.

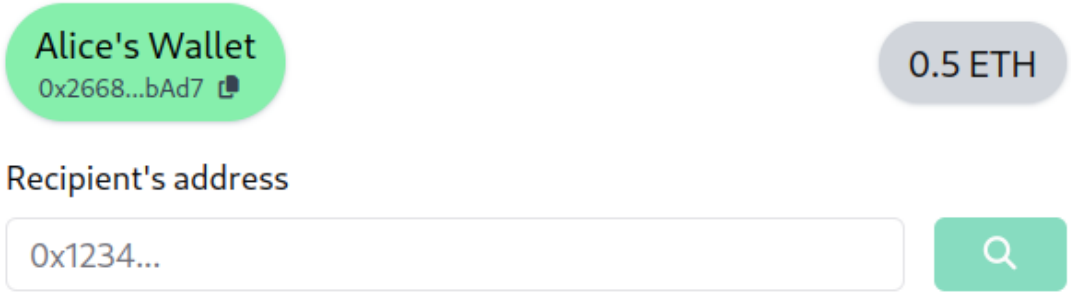

Figure 7.2: Initial state of Alice's wallet

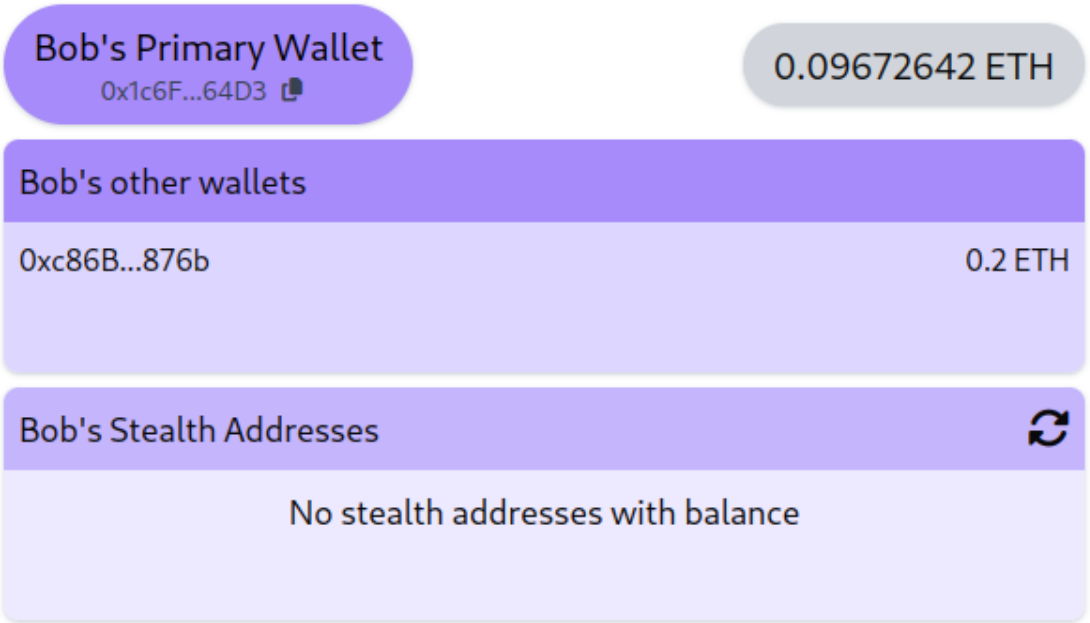

Figure 7.3: Initial state of Bob's wallet

Alice can paste Bob's primary address and query the meta stealth address registry for his meta stealth address, shown in figure 7.4.

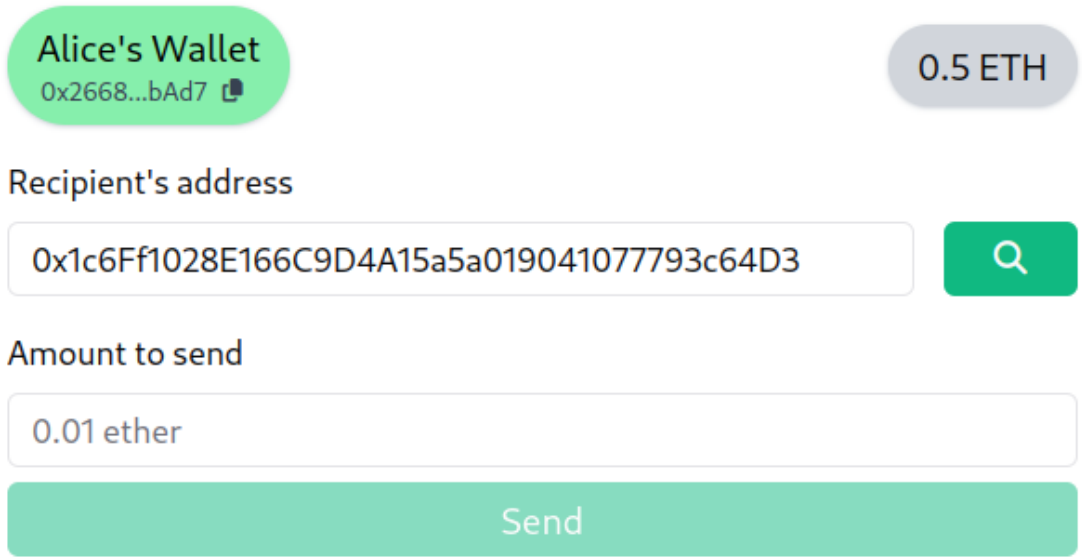

Figure 7.4: Querying Bob's meta stealth address

Figure 7.5 displays scenario when Alice queries for an address without meta stealth address.

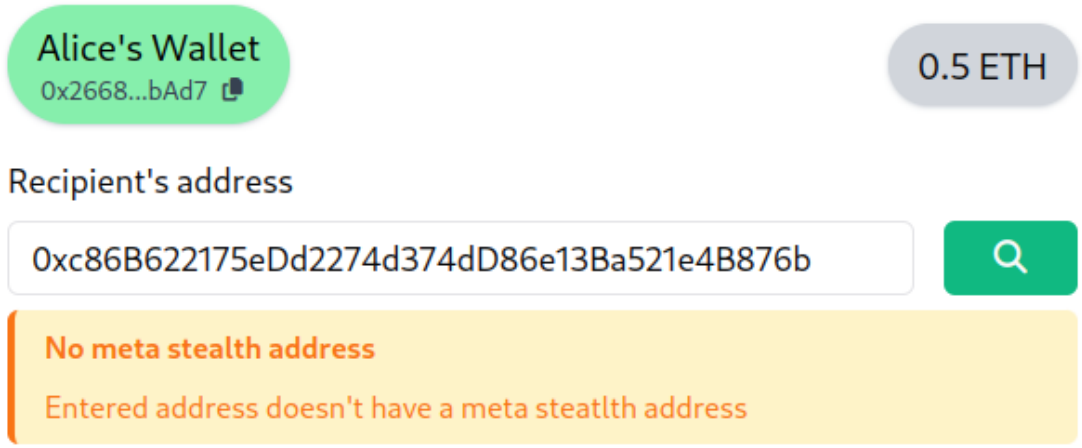

Figure 7.5: Querying address without meta stealth address

When Alice sends funds to Bob's stealth address, she first deploys, the stealth wallet contract and then submits the ephemeral key to the ephemeral key registry. Bob can then scan the ephemeral key registry and decrypt Alice's secret, and the address of his stealth wallet with sent funds. Stealth wallets are then tracked, shown in figure 7.6.

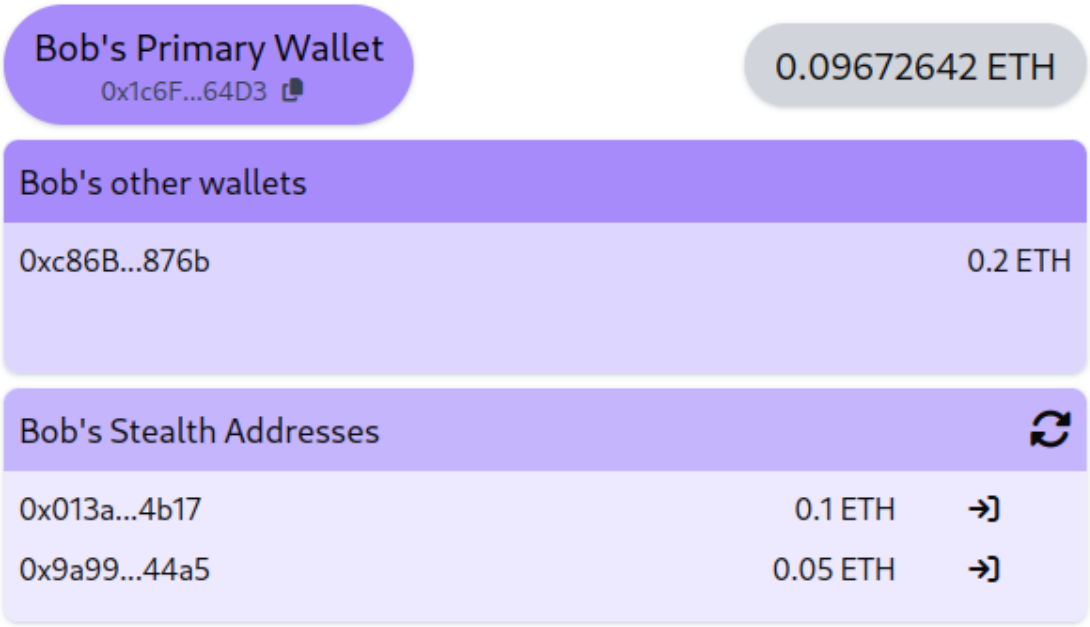

Figure 7.6: Tracking of Bob's stealth addresses

If Bob wants to use the funds, he can withdraw them to the his secondary address, which is not linked to his primary address, as depicted in figure 7.7.

When accepted, the browser wallet generates a proof from Bob's secret value, Alice's secret value and address which will be doing the withdrawal. The proof is submitted and funds are transferred to the chosen withdrawal address, as shown in figure 7.8.

Figure 7.7: Withdrawing of funds

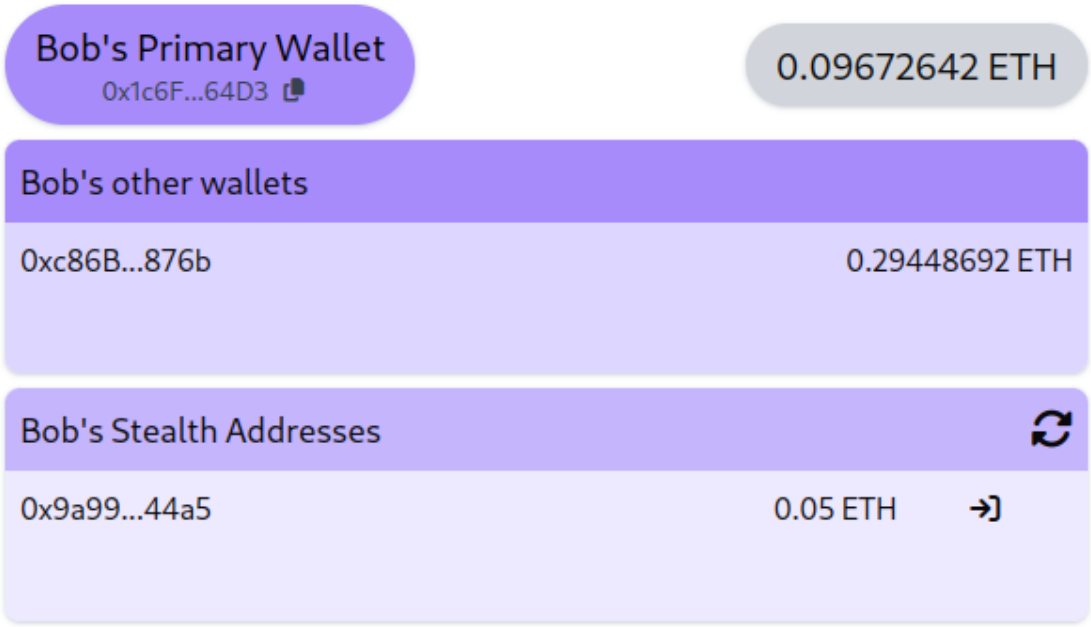

Figure 7.8: State of Bob’s secondary address after withdrawal

7.3 Gas analysis

The gas consumption of the smart contracts was analyzed using the `forge test -gas-report` command, which is part of the Foundry development

framework. The most important gas metric is the one for withdrawing funds from StealthWallet contract, because in that function the proof validation occurs. The median for it is `223902` gas, little higher than a Uniswap V3 swap. The gas cost of the withdrawal function is dominated by the proof verification, which is done by the `verifyProof` function in the `Groth16Verifier` contract. This function has a constant gas cost of `195686`. The remaining gas is used for other operations in the withdrawal function, such as transferring funds to the recipient.

EphemeralKeyRegistry				
Deployment Cost 400998	Deployment Size 1642			
Function	min	avg	median	max
getKeysBatch	423	4340	4920	10612
submit	58133	160677	161129	178229

MetaStealthAddressRegistry				
Deployment Cost 501153	Deployment Size 2107			
Function	min	avg	median	max
addressMetaStealthAddress	1956	3105	1956	5404
setMetaStealthAddress	25346	60353	36146	119051

Groth16Verifier				
Deployment Cost 371621	Deployment Size 1506			
Function	min	avg	median	max
verifyProof	195686	195686	195686	195686

StealthWallet				
Deployment Cost 268322	Deployment Size 1257			
Function	min	avg	median	max
withdraw	23100	182078	223902	257409

7.4 Static analysis

The implementation of the smart contracts was analyzed using the static analyzers Aderyn [**githubCyfrinaderyn**] and Slither [**githubCryticslither**].

7.4.1 Aderyn results

Static analysis of the smart contracts using the Aderyn tool revealed four low-severity issues.

1. **Imprecise Solidity Pragma:** The use of a broad Solidity pragma `^0.8.13` was flagged, suggesting the use of a specific version for better compatibility and security. This was addressed by changing the pragma to pragma solidity 0.8.20;
2. **Missing Zero Address Check:** The `StealthWallet.sol` lacked a check for the zero address of a verifier contract. This was fixed by adding the necessary check.
3. **Public Function Optimization:** Some public functions that were not used internally were identified as potential candidates for being marked as external to optimize gas usage. These functions were modified accordingly.
4. **PUSH0 Opcode Compatibility:** The report noted that the Solidity compiler version used could generate bytecode with PUSH0 opcodes, which might not be supported on all (EVM) chains. This was acknowledged, and it was decided to deploy the contracts only on chains that support EVM Shanghai, which includes the PUSH0 opcode.

The whole report generated by Aderyn can be found in

`stealth-wallet/aderyn-report.md`.

7.4.2 Slither results

Static analysis of the smart contracts using Slither identified several issues, categorized as high, low, informational, and optimization. However, it is important to note that the majority of these findings were in the generated `Verifier.sol` contract and external library code, rather than the core logic of the implemented stealth address scheme.

High Severity

1. **arbitrary-send-eth:** When withdrawing from stealth wallet, Ether could potentially be sent to an arbitrary address due to the lack of validation on the recipient's address in the withdraw function. This is not true, since only a owner can generate a valid proof, so funds will be transfered to address specified by the owner.
2. **incorrect-return:** Three instances of incorrect return values were identified in the Groth16Verifier contract, which could halt execution. This contract is generated with SnarkJS, hence these issues can only be acknowledged. Fixing these issues in the SnarkJS generator is outside the scope of this project.

Low Severity

1. **missing-zero-check:** Stealth wallet did not check if the recipient address is zero address, this check was added.

Informational

Several informational findings were related to the use of assembly code, pragma directives, Solidity compiler versions, low-level calls, naming conventions, and unused state variables. These were primarily found in the generated `Verifier.sol` contract and external library code. While acknowledged, they were not considered to directly impact the security or functionality of the implemented scheme.

Optimization

Two state variables in the stealth wallet contract were identified as potential candidates for being declared as immutable, which could optimize gas usage. These variables were made immutable.

The whole report generated by Slither can be found in `stealth-wallet/slither-report.md`.

Chapter 8

Conclusion

This thesis has presented how ZKPs can be applied to the problem of stealth addresses in Ethereum. How one can receive payments without revealing their identity, and do so without the need to actively communicate with the sender.

This proof of concept of a ZKP Stealth address Scheme on Ethereum allows a sender to send funds to a recipient without any prior communication, and without submitting any data, that links the sender with the recipient, to the blockchain. And allows the recipient to prove that they are the intended recipient of the funds, without revealing their identity, and without the need to actively communicate with the sender.

The key novelty and scientific contribution of this work lie in the successful demonstration of the practical application of ZKPs to address real-world privacy challenges in blockchain technology. By presenting a conceptual implementation of the stealth address scheme, this thesis showcases the potential of ZKPs to enhance privacy and security in blockchain-based applications.

The scheme's design allows for the creation of stealth addresses that are not directly linked to the recipient's identity, while still enabling the recipient to prove ownership and control over the funds. This proof-of-concept implementation serves as a contribution to the ongoing research and development in the field of privacy-preserving blockchain technologies.

8.1 Future Work

The design and implementation of the ZKP stealth address scheme serves only as a proof of concept. For this scheme to be widely adopted, additional work is required. This chapter outlines some of the potential future work that could be done to improve the scheme.

Ephemeral Key Search

The current implementation of the ZKP stealth address scheme requires that sender submits the ephemeral public key to a registry. The registry is just a simple storage that stores these keys in a list. This method does not scale well with mass adoption. Each recipient would have to scan the entire list of keys to check if any belong to them.

Current ERC-5564 [**ethereumERC5564Stealth**] proposal, which differs from here proposed scheme mainly in the use of elliptic curve cryptography instead of ZKPs, uses view tags to reduce the number of keys that need to be scanned. Or, different approach, proposed by Xin Wang, Li Lin, Yao Wang [**Wang2023**] utilizes subgroup membership assumption related to factoring for faster key search.

Recoverability

The current implementation of the ZKP stealth address scheme does not allow for possibility of funds recovery. This means that if the recipient loses their private key, they will not be able to recover their funds.

One way to address this issue would be to use a private key recovery mechanism, such as Shamir Backup. With this approach, the stealth address scheme would not need to be modified, because the private key recovery would be handled outside of the blockchain.

Different approach would be to modify the stealth address scheme to implement a concept of a Social Recovery Wallet. These types of wallets enable user to assign guardians to their wallet. Guardians are different accounts, for example friends, mobile phone, hardware wallet or some kind of institution. The wallet holds a public key to some user owned private key. Only this key can control the wallet. If the user loses their private key, they can request the guardians to change the associated public key in the wallet, thus changing the private key that is used to control the wallet [**ButerinSocialRecovery**]. An incomplete implementation of this concept was proposed by Vitalik Buterin [**ButerinIncompleteGuide**], which uses ZKPs to prove to the stealth addresses, that the user is the owner of the wallet.

Resumé

Appendix A

Project task schedule

A.1 Winter semester

1st-4th week	Exploring ZK ecosystem and finding suitable ZK theme
5th-8th week	Researching ZK-EVM, ZK-Rollups, Stealth Addresses
9th-11th week	Researching Stealth Addresses with ZKP scheme
12th-13th week	Writing Analysis and Solution design

A.2 Summer semester

1st-2nd week	Learning CIRCOM
3rd-4th week	Implementing ZKP with CIRCOM
5th-7th week	Implementing smart contracts in Solidity
8th-9th week	Implementing browser experimentation wallet
10th-11th week	Solution testing
12th-13th week	Revision of final document

Appendix B

Contents of the digital medium

Registration number of the thesis in the information system: FIIT-16768-116160

Contents of the digital medium (ZIP archive):

Name of the submitted archive: BP_LukasCastven.zip.