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.

December 2023

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.

December 2023

Slovenská technická univerzita v Bratislave
Ústav počítačového inžinierstva a aplikovanej informatiky

Fakulta informatiky a informačných technológií
2023/2024

ZADANIE BAKALÁRSKEJ PRÁCE

Autor práce:	Lukáš Častven
Študijný program:	informatika
Študijný odbor:	informatika
Evidenčné číslo:	FIIT-16768-116160
ID študenta:	116160
Vedúci práce:	Ing. Kristián Košťál, PhD.
Vedúci pracoviska:	Ing. Katarína Jelemenská, PhD.
Názov práce:	**Practical usage of Zero-knowledge in different use-cases in blockchains**
Jazyk, v ktorom sa práca vypracuje:	slovenský jazyk
Špecifikácia zadania:	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.
Rozsah práce:	40
Termín odovzdania práce:	21. 05. 2024
Dátum schválenia zadania práce:	
Zadanie práce schválil:	

Anotácia

Slovenská technická univerzita v Bratislave

Fakulta informatiky a informačných technológií

Študijný program: Informačná bezpečnosť

Autor: Lukáš Častven

Bakalárska práca: Praktické využitie dôkazov s nulovým vedomím v rôznych prípadoch použitia v blockchainoch

Vedúci projektu: Ing. Kristián Košťál PhD.

December 2023

Bakalárska práca skúma aplikáciu dôkazov s nulovým vedomím (ZKPs) v blockchaine. 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 odvodiť tajnú adresu z verejných údajov príjemcu. Iba príjemca má kontrolu nad touto adresou, a zaroveň 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.

December 2023

This bachelor's thesis investigates the application of Zero Knowledge Proofs (ZKPs) in blockchains. Zero Knowledge Proofs are a cryptographic method which enables the validation of data without revealing the data itself, making it ideal for secure and private applications in blockchain networks. The focus of this thesis is the 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
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 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.

The concept of ZKPs was first introduced in a 1989 research paper, "The Knowledge Complexity of Interactive Proof Systems."[1]. 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. This interaction is pivotal in ZKPs, as it allows 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.

This thesis extends the application of ZKPs to the concept of stealth addresses in blockchain, as outlined in Vitalik Buterin's article "An Incomplete Guide to Stealth Addresses."[2]. 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.

Chapter 2

Analysis

The analysis section of the thesis starts with demonstration of interactive proofs with goal to build up intuition behind interactive proofs [1, 3]. 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 [4], 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 [5] 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.

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 [1]. QR is defined as:

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

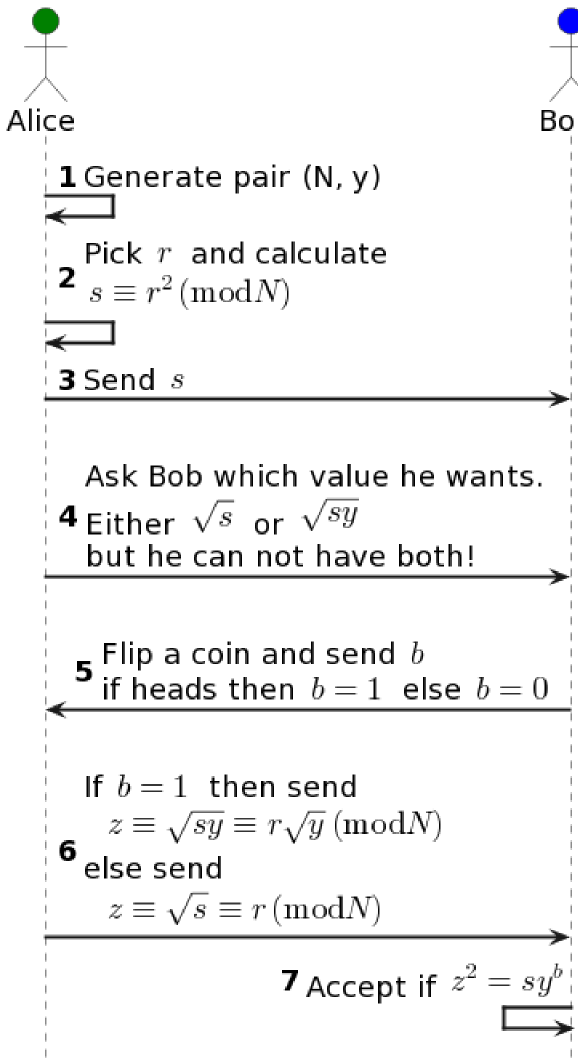
Alice
Bob
1 Generate pair (N, y)
2 Pick r and calculate $s \equiv r^2 (\mathrm{mod} N)$
3 Send s
4 Ask Bob which value he wants. Either $\sqrt{s}$ or $\sqrt{sy}$ but he can not have both!
5 Flip a coin and send b if heads then $b = 1$ else $b = 0$
6 If $b = 1$ then send $z \equiv \sqrt{sy} \equiv r\sqrt{y}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

Figure 2.1: Interactive proof of language QR

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 [6, 7]. The paper "How To Prove Yourself: Practical Solutions to Identification and Signature Problems"[4] 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

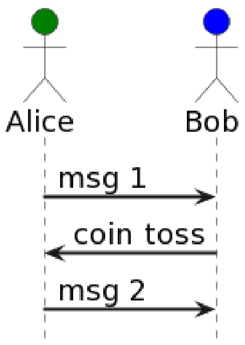

Figure 2.2: Interactive public coin proof

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

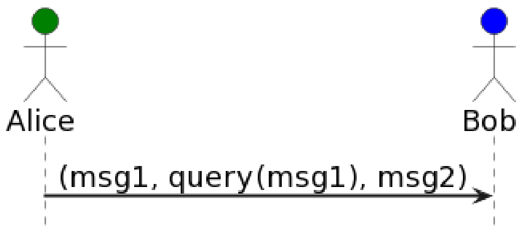

Figure 2.3: Non-interactive public coin proof

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.

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

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

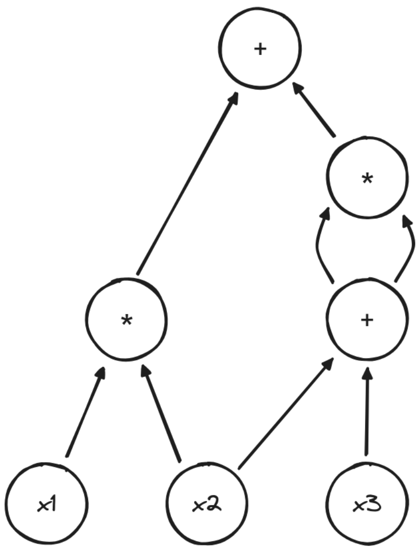

Figure 2.4: Graph representation of arithmetic circuit

Another representation of a 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:

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

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 $C(x, w) \rightarrow \mathbb{F}$, 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 $Setup(C, r) \rightarrow (pp, vp)$, 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 $Setup(C) \rightarrow (pp, vp)$, 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 $Verify$

algorithm 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, 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:

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

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 Appendix A.

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

$$(comm_{f1}, FS(r_1), comm_{f2}, FS(r_2), \dots, comm_{fm}, FS(r_m))$$

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.

2.5 Stealth addresses

Ethereum is a public blockchain, where 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 [8]. 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 derive the private key for the stealth address.

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 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

Solution Desing

The Solution Design section outlines the design of a stealth address scheme using ZKPs for the Ethereum blockchain. The scenario which is being solved involves Alice, who wishes to send funds to Bob discreetly, ensuring no one else can identify Bob as the recipient. This part of the thesis details the application of ZKPs employed to achieve this privacy, allowing Alice to complete the transaction without compromising Bob's identity.

3.1 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). The hash is later used to prove to a stealth address contract that Bob is the owner of the address and can spend funds sent to it.

3.2 Sending Funds

When Alice wishes to send funds to Bob, she must first generate a new stealth address. Alice looks up Bob's meta stealth address (K, h) and computes the following:

- A secret value c
- A new random stealth address SA
- An empheral key $P = encrypt(value = [c, SA], key = K)$
- A code $C = hash(h, c)$

Alice then creates a smart contract at the new stealth address SA, which contains the code C and sends funds to it. Then in the same transaction, Alice sends the empheral key P to public registry contract, which stores the empheral keys for all stealth addresses.

3.3 Scanning for Stealth Addresses

Bob scans for stealth addresses by querying the public registry contract from tha last point he scanned. The registry contract returns a list of empheral keys. Bob then decrypts each empheral key with his private key k. If the empheral key is meant for Bob, then it will contain the secret value c and the stealth address SA.

3.4 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.

3.5 Overview

The solution design is illustrated in Figure 3.1.

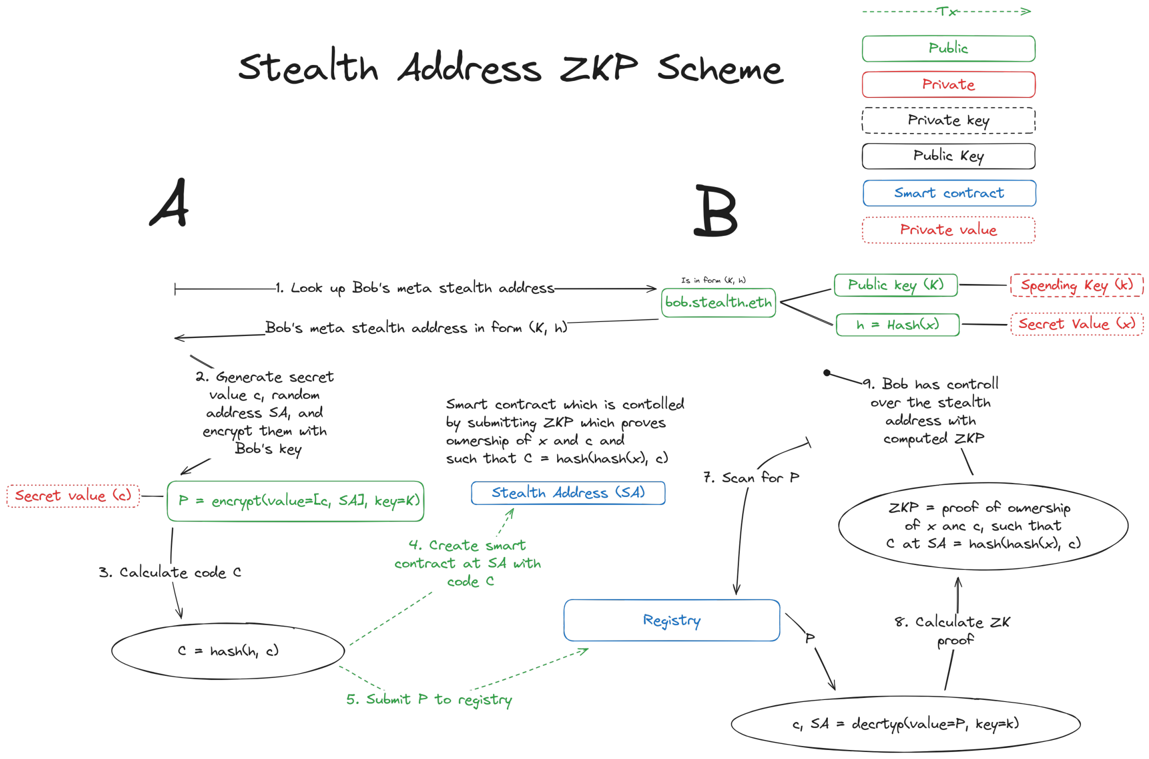
Stealth Address ZKP Scheme
Tx
Public
Private
Private key
Public Key
Smart contract
Private value
A
B
1. Look up Bob's meta stealth address
Is in form (K, h)
bob.stealth.eth
Public key (K)
Spending Key (k)
h = Hash(x)
Secret Value (x)
Bob's meta stealth address in form (K, h)
2. Generate secret value c, random address SA, and encrypt them with Bob's key
Smart contract which is contolled by submitting ZKP which proves ownership of x and c and such that C = hash(hash(x), c)
9. Bob has controll over the stealth address with computed ZKP
Secret value (c)
P = encrypt(value=[c, SA], key=K)
Stealth Address (SA)
7. Scan for P
ZKP = proof of ownership of x anc c, such that C at SA = hash(hash(x), c)
3. Calculate code C
4. Create smart contract at SA with code C
Registry
P
8. Calculate ZK proof
C = hash(h, c)
5. Submit P to registry
c, SA = decrtyp(value=P, key=k)

Figure 3.1: Solution Design

Resumé

References

[1] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. “The Knowledge Complexity of Interactive Proof Systems”. In: *SIAM Journal on Computing* 18.1 (Feb. 1989), pp. 186–208. ISSN: 1095-7111. DOI: 10.1137/0218012. URL: http://dx.doi.org/10.1137/0218012.

[2] *An incomplete guide to stealth addresses — vitalik.eth.limo*. https://vitalik.eth.limo/general/2023/01/20/stealth.html. [Accessed 12-12-2023].

[3] *ZKP MOOC Lecture 1: Introduction and History of ZKP — youtube.com*. https://www.youtube.com/watch?v=uchjTIlPzFo&list=PLS01nW3Rtgor_yJmQsGBZAg5XM4TSGpPs. [Accessed 16-12-2023].

[4] Amos Fiat and Adi Shamir. “How To Prove Yourself: Practical Solutions to Identification and Signature Problems”. In: *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, pp. 186–194. ISBN: 9783540180470. DOI: 10.1007/3-540-47721-7_12. URL: http://dx.doi.org/10.1007/3-540-47721-7_12.

[5] Oded Goldreich, Silvio Micali, and Avi Wigderson. “Proofs that yield nothing but their validity or all languages in NP have zero-knowledge proof systems”. In: *Journal of the ACM (JACM)* 38.3 (1991), pp. 690–728.

[6] László Babai and Shlomo Moran. “Arthur-Merlin games: A randomized proof system, and a hierarchy of complexity classes”. In: *Journal of Computer and System Sciences* 36.2 (Apr. 1988), pp. 254–276. ISSN: 0022-0000. DOI: `10.1016/0022-0000(88)90028-1`. URL: `http://dx.doi.org/10.1016/0022-0000(88)90028-1`.

[7] S Goldwasser and M Sipser. “Private coins versus public coins in interactive proof systems”. In: *Proceedings of the eighteenth annual ACM symposium on Theory of computing - STOC ’86*. STOC ’86. ACM Press, 1986. DOI: `10.1145/12130.12137`. URL: `http://dx.doi.org/10.1145/12130.12137`.

[8] *[Bitcoin-development] Stealth Addresses — lists.linuxfoundation.org*. `https://lists.linuxfoundation.org/pipermail/bitcoin-dev/2014-January/004020.html`. [Accessed 19-12-2023].

Appendix A

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.

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

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* [9, 10, 11].

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.

References

[9] J. T. Schwartz. “Fast Probabilistic Algorithms for Verification of Polynomial Identities”. In: *Journal of the ACM* 27.4 (Oct. 1980), 701–717. ISSN: 1557-735X. DOI: `10.1145/322217.322225`. URL: `http://dx.doi.org/10.1145/322217.322225`.

[10] Richard Zippel. “Probabilistic algorithms for sparse polynomials”. In: *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 1979, 216–226. ISBN: 9783540351283. DOI: `10.1007/3-540-09519-5_73`. URL: `http://dx.doi.org/10.1007/3-540-09519-5_73`.

[11] Richard A. Demillo and Richard J. Lipton. “A probabilistic remark on algebraic program testing”. In: *Information Processing Letters* 7.4 (June 1978), 193–195. ISSN: 0020-0190. DOI: `10.1016/0020-0190(78)90067-4`. URL: `http://dx.doi.org/10.1016/0020-0190(78)90067-4`.

Appendix B

Project task schedule

B.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