VARIANCE:

SECURE TWO-PARTY PROTOCOL FOR SOLVING YAO'S MILLIONAIRES' PROBLEM IN BITCOIN

by
Joshua Holmes

A thesis
submitted in partial fulfillment
of the requirements for the degree of
Master of Science in Computer Science
Boise State University

December 2017

BOISE STATE UNIVERSITY GRADUATE COLLEGE

DEFENSE COMMITTEE AND FINAL READING APPROVALS

of the thesis submitted by

Joshua Holmes

Thesis Title: Variance:
Secure Two-Party Protocol for Solving Yao's Millionaires' Problem in Bitcoin

Date of Final Oral Examination: 19 October 2017

The following individuals read and discussed the thesis submitted by student Joshua Holmes, and they evaluated the presentation and response to questions during the final oral examination. They found that the student passed the final oral examination.

Gaby Dagher, Ph.D.	Chair, Supervisory Committee
Dianxiang Xu, Ph.D.	Member, Supervisory Committee
Marion Scheepers, Ph.D.	Member, Supervisory Committee

The final reading approval of the thesis was granted by Gaby Dagher, Ph.D., Chair of the Supervisory Committee. The thesis was approved by the Graduate College.

dedicated to “my Family”

ACKNOWLEDGMENTS

Also, I would like to thank my family and friends for the constant support and motivation.

Abstract

Secure multiparty protocols are useful tools for parties wishing to jointly compute a function while keeping their input data secret. The millionaires' problem is the first secure two-party computation problem, where the goal is to securely compare two private numbers without a trusted third-party. There have been several solutions to the problem, including Yao's protocol [Yao, 1982] and Mix and Match [Jakobsson and Juels, 2000]. However, Yao's Protocol is not secure in the malicious model and Mix and Match unnecessarily releases theoretically breakable encryptions of information about the data that is not needed for the comparison. In addition, neither protocol has any verification of the validity of the inputs before they are used. In this thesis, we introduce Variance, a privacy-preserving two-party protocol for solving the Yao's millionaires' problem in a Bitcoin setting, in which each party controls several Bitcoin accounts (public Bitcoin addresses) and they want to find out who owns more bitcoins without revealing (1) how many accounts they own and the balance of each account, (2) the addresses associated with their accounts, and (3) their total wealth of bitcoins while assuring the other party that they are not claiming more bitcoin than they possess. We utilize commitments, encryptions, zero knowledge proofs, and homomorphisms as the major computational tools to provide a solution to the problem, and subsequently prove that the solution is secure against active adversaries in the malicious model.

Contents

List of Tables

List of Figures

LIST OF ABBREVIATIONS

ZKP – Zero Knowledge Proof

MPC – Secure Multi-Party Computations

EC – Elliptic Curve

ECDLP – Elliptic Curve Discrete Log Problem

TTP – Trusted Third Party

LIST OF SYMBOLS

$[\![m]\!]$	An EC Exponential Elgamal encryption of m ($(g^m \cdot y^r, g^r)$).
$\mathcal{C}_m$	An EC Pedersen Commitment hiding m ($g^m \cdot h^\alpha$).
$\mathcal{C}_{(m,\alpha)}$	An EC Pedersen Commitment hiding m with a key α ($g^m \cdot h^\alpha$).
g	A generator point of the EC group.
h	A second generator point of the EC group.
y	The distributed EC Elgamal Public Key.

Chapter 1

INTRODUCTION

1.1 Motivation

Secure multiparty computation is an essential and expanding field in cryptography [10]. The ability to run useful computations while maintaining privacy of data opens up opportunities to compute on private and distributed data. Most of the work in secure multiparty computation is focused on encryption schemes. While encryption is usually sufficient, it can eventually be decrypted, especially as technology improves or if the hard problems are solved. To accommodate permanently relevant private data, a secure multiparty protocol should adapt perfectly-hiding techniques to perfectly hide as much data as possible.

In Yao's Millionaires' Problem [25], two parties want to compare their wealth without revealing how much money they have. Since its conception, it has been one of the elementary problems of secure computation, used to demonstrate how more generic secure computational systems can be applied. Yao proposed a two-party protocol [25], utilizing garbled logic gates to simulate a circuit. The main limitation of Yao's protocol is that the construction of the circuit can not be verified, which makes the protocol only secure in the semi-honest model.

Later, Jakobsson and Juels [15] proposed a solution to the Millionaires' Problem that is secure in the malicious model. They designed the concept of Mix and Match tables, which uses shuffled and encrypted truth tables to simulate logic gates and distributed Elgamal

to compare two bits or numbers with a *plaintext equality test* to find a matching row and retrieve the output. It then uses a *verifiable mix network* [16] to randomize the table so the rows are not traceable so that one party can not infer associations between the values in the different columns.

In this thesis, we introduce a solution to the Millionaires' problem in the context of Bitcoin [20]. In the Bitcoin cryptocurrency system, an account is presented as a public address associated with a balance and users hold the associated private keys to their accounts. To determine who owns more bitcoins, it is not sufficient to simply compare two numbers. Owners of bitcoin usually store their holdings in multiple accounts. Due to the nature of Bitcoin, any protocol involving a user's accounts is required to preserve privacy. This adds an extra layer of complexity to the comparison, as not only do the total assets need to be secure, but the specific owned accounts and the number of accounts must also be kept secret. Due to the nature of the Bitcoin's blockchain, any information associating accounts with owners can be used to determine private information about the owner in the future, such as their current active accounts. Variance is a commitment-based protocol employing zero knowledge proofs to run bit comparisons to solve the Millionaires' Problem in the context of Bitcoin.[1]

1.2 Challenges & Concerns

There are several challenges associated with creating a protocol that solves the Millionaires' Problem for Bitcoin. Since it is common for Bitcoin holders to keep their assets in multiple accounts, these accounts have to be added for any real comparison to take place. Any protocol which manages this sum would have to be privacy-preserving, which means

[1] Following convention, we refer to the protocol as 'Bitcoin' and the units of currency as 'bitcoin' or ₿.

the total sum, the amounts in the accounts, and the public keys for the owned accounts must be kept secret. The last requirement is called *unlinkablity* [5][4], meaning a party can not be linked to their accounts and vice versa. However, even if another party can not figure out the accounts through the protocol but can figure out the sums, they may be able to piece together which accounts the party holds, though this is a hard problem.

Due to the sensitivity of some of the data, a commitment based scheme using perfectly hiding commitments would be preferable. Specifically, the owned accounts should be impossible, or at least extremely difficult, to determine. Due to the nature Bitcoin's blockchain, if an adversary manages to associate an account with a party, then they may be able to figure out other accounts associated with the party, even perhaps being able to track owned accounts in the future [22]. It would be best to have as much data about the sum as possible be perfectly hidden, so the protocol is not as dependant on the discrete log problem. Even obscuring small details of the sum may the prevent discovery of claimed accounts.

As the Bitcoin protocol is designed to be secure malicious users, any solution to this problem should also be secure in the malicious model.

1.3 Limitations

A limitation of Variance can only solve the Millionaires' Problem if the goal of each party is to show that they have the most wealth. Accounts can easily be omitted by either party to artificially reduce their apparent net worth, which could be used to avoid appearing to have more bitcoins. This protocol can only work if the goal of each party is to prove they have more Bitcoin with no ulterior motivations.

Another limitation is the issue of collusion, where a group of people pool their private

keys to make a party appear to have more bitcoins than they actually have. There is no known solution to this problem because either party could be colluding with account holders outside the protocol.

Variance is also limited to accepting key schemes to which a Zero Knowledge Proof of Knowledge of the secret is known. For example, to include accounts for which the secret is only hidden behind a pay-to-pub-hash or pay-to-script-hash, a Zero Knowledge Proof of Knowledge of the private key of the preimage of the hash is required. Even with this proof, Provisions's proof of assets could not include these accounts [5]. This is a major issue in the context of the Millionaires' Problem, as wealthy Bitcoin holders tend to keep many of their assets (bitcoins) in non-spending accounts, where bitcoins is put in to an account and not spent. In these cases, only the public address (the hash) is known. However, since one of the common applications of solutions to the Millionaires' Problem is to run auctions, this solution still has many applications while without supporting non-spending accounts. This problem can be solved if a Zero Knowledge Proof of Knowledge of the private key associated with the public address, meaning the private key to the preimage of the hash, can be created, but any such proof would likely be significantly less efficient than the proof of knowledge for a discrete log public key.

1.4 Thesis Statement

The objective of this thesis is to answer the following question: **How can two non-trusting individuals determine in a privacy-preserving manner who owns more bitcoins?**

More specifically, given two non-trusting parties (Bitcoin users) P_1 and P_2, each of whom owns a private set of private keys associated with a subset of the accounts in the public Bitcoin blockchain, the goal of this thesis is to propose a secure and privacy-preserving

protocol for determining whose total assets of bitcoins is higher such that:

1. No account from the public Bitcoin blockchain can be linked back to either party.
2. Neither party can determine the other's total Bitcoin assets or number of accounts.
3. The protocol is portable across multiple cryptosystems.

1.5 Organization of the Thesis

The rest of the thesis is organized as follows:

1. Chapter 2 discusses the related literature over the past years in the fields related to Bitcoin, Zero Knowledge Proofs, and Secure Multi-Party Computation.
2. Chapter 3 discusses the building blocks and preliminaries needed for this proposal.
3. Chapter 4 discusses our protocol and methodology for secure and privacy-preserving comparison of Bitcoin assets.
4. Chapter 5 is an analysis of the security of the protocol.
5. Chapter 6 discusses the computational complexity of the protocol, as well as evaluation of experiments.
6. Chapter 7 concludes the thesis and provides suggestions for future work.

Chapter 2

LITERATURE REVIEW

2.1 Bitcoin

Bitcoin was introduced in 2008 and has become the first successful cryptocurrency [20]. Bitcoin is a pseudoanonymous system based on a blockchain system. It is pseudoanonymous because payments are associated with public addresses, which may not be associated to you but are associated to each other [22]. In Bitcoin, an address is hash of a public key, where the private key is held by the account holder. The public key does not appear on the blockchain unless the account uses a transaction to send bitcoins to another account. When a person is dealing with any kind of currency, they will often be required to prove they have a certain amount of money, or at least show how much money they have. There have been several methods of proving how much bitcoin a person holds. The most basic reasonable approach that is widely used is a signature from a bitcoin wallet service. These signatures are effective, as this confirms that the account owner does holds the account. In 2011, the Bitcoin Exchange Mt. Gox wanted to prove that they had a large stockpile of Bitcoin. To accomplish this, they executed one large transfer of ₿420k. Though this action did prove they had ₿420k, it was certainly not optimal, as it revealed the size of Mt. Gox, the amount of Bitcoin they control, and the addresses of their holdings. It also proved nothing about the amount of Bitcoin they owed. This was hazardous, as knowledge of past addresses may be able to be used to infer future owned accounts. [22]. These proofs of assets all are

technically effective, but they do not meet the our goals because they reveal the accounts held and how much money is in each account. Recall that our goal is to determine who has more bitcoin without revealing the amount each party holds nor any information as to the accounts.

Provisions [5] has the most applicable Proof of Assets, which manages to keep the accounts and the amount of bitcoin secret. It utilizes a series of commitments that are verified and combined in such a way that only the balance of an owned account can be added to the total sum commitment. This fulfills our goals of not revealing the owned accounts or the total sum. However, according to the paper, it would take over 20GB and an excessive amount of time to execute their proof over the entire blockchain. This makes it applicable to the daily or periodic Proof of Solvency that exchanges may be expected to execute, which is the purpose for which it was designed, but it may not be sufficiently efficient for situations that call upon solutions to the Millionaire's Problem. Their specific methodology also stops them from using non-spending accounts and requires that the commitment and key systems use the same cryptographic setting (key schemes can not be mixed), which is less than optimal. We attempt to improve on their conceptual method by making it portable (allows multiple cryptographic settings), more efficient, modular, and more intuitive.

2.2 Secure Multiparty Computation (MPC)

Yao started the field of Secure Multiparty Computation by proposing the Millionaire's Problem [25]. There are several conceptual methods to accomplish secure multiparty computations without revealing any of the party's secret data, but the most general methods involve simulating logic gates to simulate a circuit, like Yao [25] and Mix and Match by

Jakobson and Juels [15], because they can simulate any function using logic circuits, though it may take an inordinately large amount of time.

There has been a significant amount of work in secure multiparty computations [1] [13] [9]. There are several important uses for MPCs. One important purpose is to facilitate cloud computing without having to forgo privacy [19]. Another use is privacy preserving database queries and data mining. [18][24]. To accommodate these schemes, MPC schemes are being designed to be scalable. Though a Yao garbled circuit can execute any executable function securely, it would be extremely inefficient to execute an entire program by running expensive computations on every single basic binary operation. More efficient methods have been designed, though they are still not very efficient for heavy computation on the cloud, both in communication and computation. However, most or all of these schemes rely on public key encryptions and the ability to decrypt in some form or changing the data through homomorphism, which is incompatible with a commitment-based scheme. If the owner of the commitment does not know how the commitment was altered, they no longer have the knowledge required to open the commitment.

2.3 Zero Knowledge Proofs (ZKPs)

Zero Knowledge Proofs are designed to be used to allow users of protocols to prove that they are following the protocol properly without revealing any of their secrets by proving their inputs are correspond with a statement, thus allowing protocols to be run even with malicious parties [5][15].They are also used to prove identity without having to show any sensitive information [23][8]. Zero Knowledge Proofs's were proposed by Goldwasser, Micali, and Rackoff [12]. They introduced the concept of the interactive problem space IP and the idea of measuring the amount of knowledge transferred in a proof, which they

called knowledge complexity. They also proposed a the first zero knowledge proof. Later Oded Goldreich, Silvio Micali, and Avi Wigderson [11] proved that all problems in NP have associated zero knowledge proofs.

ZKP's are often designed as Sigma protocols. A sigma protocol is a 3 step interactive protocol. Sigma protocols were created by Schnorr [23] to accomidate Schnorr's Protocol as part of his Signature Protocol. Since then, they have become the mainstay of Zero Knowledge Proof design, as they are relatively simple to prove security with simulators [6]. ZKP's can easily be made non-interactive through the Fiat-Shamir heuristic [7].

Combining Zero Knowledge Proofs with logical compositions to prove complicated statements can be more intuitive and efficient than a tailor-made proof. It can also be used to make more generic and portable zero knowledge proof. The two major components required to merge existing ZKP's into more intricate proofs are an AND and an OR. Zero Knowledge AND is trivial given knowledge of Zero Knowledge Proofs and Sigma protocols. AND is generally not discussed due to its simplicity. This method also simplifies the security proofs because the compositions are recursively proven secure if the basic components are proven secure. There have been two functionally equivalent formulations of Zero Knowledge OR, addition [3] and XOR [6][13]. To accomplish our goals, we apply a generalization of the of XOR formulation.

Chapter 3

BACKGROUND AND BUILDING BLOCKS

3.1 Public Parameters

We define two generators g and h such that no party knows the discrete log between g and h. g and h are points on an elliptic curve, e.g. secp256k1 [2]). It should be noted that if the cofactor of the curve is 1, then every point but the primitive point INF is a generator. There is also a distributed Elgamal private key y, where all members of the protocol hold a part of the key. Though we work with with Bitcoin public and private keys, we do not perform any ECDSA signatures. Though we use elliptic curves, we employ the standard notations for integer groups $y = g^x$ rather than the standard elliptic curve notation $Y = xG$ ($g \cdot h$ instead of $G + H$).

3.2 Elliptical Curve Exponential Elgamal Cryptosystem [17]

Elgamal is a semantically secure asymmetric cryptosystem based on the Elliptical Curve Discrete Log Problem. It is noted for the ability to rerandomized and the ability to have a distributed private key. The encryptions will be written as $[\![m]\!] = (g^m y^r, g^r)$, with generator g, public key $y = g^k$ where k is the private key. The calculations are executed on a specified curve (e.g. secp256k1), which has an order q. Elgamal also supports scalar multiplication of the ciphertext to be translated in to scalar addition of the plaintext. The main weakness

of exponential elgamal in general is to properly decrypt to a specific message, the discrete log has to be solved or the resulting point has to be recognized. This is not an issue for a small message set.

3.3 Pedersen Commitments [21]

Pedersen Commitments are a perfectly hiding computationally binding commitment scheme based on any system with an associated hard Discrete Log Problem. Commitments will be written as $\mathcal{C}(m) = g^m h^\alpha$, where no party knows the discrete log between g and h When referring to Pedersen commitments, or any other perfectly hiding commitment scheme, a commitment is described as hiding a value. This is not exactly true, as a commitment could hide any value, but finding a second such pairing of m' and α' is at least as hard as solving ECDLP. So, when we say the commitment hides a value m, we mean that the owner knows the values m and α that create that form commitment. In this thesis, we generally use the elliptic curve secp256k1. We refer to m as the message and α as the ephemeral key, as α strongly resembles the ephemeral key from Elgamal.

3.4 Zero Knowledge Proofs [12]

Zero Knowledge Proofs are proofs that reveal no information other than the fact that the statement is true. Most Zero Knowledge Proofs are in the form of Sigma protocols, where the Prover "commits" to a value or values, denoted by a, the Verifier challenges with a t-bit value c, then the Prover sends a response z based on the initial commitment, the challenge, and the secret. The Verifier uses the response on the common knowledge, c, and the a values to confirm that the Prover must know the secret with a extremely low probability

(2^{-t}) of successful deceit. Schnorr's Protocol (Protocol 3.1) is the standard basic example of a Sigma Protocol.

The Zero Knowledge Proofs presented here are presented slightly differently from most other works. When proving a Zero Knowledge Proof is truly Zero Knowledge, one of the requirements is Special Soundness, which uses a simulator. In general, a simulator is used to show that a protocol reveals no information by showing that valid transcripts can be created only using the Verifier's knowledge. In the context of Sigma Protocols, a simulator simulates the actions of the Prover, designated Prover*, and uses pre-knowledge of the Verifier's challenge to forge an a and a z. Usually, simulators are only used when proving the protocol is sound. However, due to the heavy use of Zero Knowledge OR (Protocol 3.2), which utilizes simulators, the simulator will be presented with the protocol. By default, when we refer to the protocol, we are referring to the main protocol, not the simulator. Due to the length of some of the compound ZKPs we design, we will construct compound ZKPs using a function call style notation (e.g. AND(Schnorr(y_1,x_1), Schnorr(y_2,x_2))).

3.4.1 Schnorr's Protocol (Protocol 3.1)

This is the most common example of a zero knowledge protocol. The statement being proved is "I know the discrete log of g^x". If both parties have g^x and the Prover knows x, it allows the Prover to prove that he knows x. Many zero knowledge proofs are heavily based on this basic protocol. We call this protocol using Schnorr(y,x).

Commitment Value Proof (CVP) Protocol

This is a proof that a commitment hides a specific value. As a standalone proof, it appears useless as one could simply open the commitment and get the same result. However, with Zero Knowledge OR, it can be used for a simple set membership proof or to show that a

commitment is equal to a value if certain conditions are met and something else of not. This proof is simple for a homomorphic commitment scheme. We are specifically using Pedersen commitments, so this protocol is simple. The function call notation of this proof is $\mathsf{CVP}(\mathcal{C}_{(m,\alpha)}, m, \alpha)$), where m is the message and α is the ephemeral key. To execute, run $\mathsf{Schnorr}(\mathcal{C}_{(m,\alpha)} \cdot g^{-m}, \alpha)$

Schnorr's Protocol [23]

Input:	Public:	The exponential $y \coloneqq g^x$
	Private (Prover):	x, the discrete log of y
Output:	Public:	Verification that the Prover knows x.

1. The Prover generates a random number r and calculates $a \coloneqq g^r$
2. The Prover sends a to the Verifier.
3. The Verifier generates a random challenge c and sends it to the Prover.
4. The Prover computes $z \coloneqq r + c \cdot x$.
5. The Prover sends z to the Verifier.
6. The Verifier accepts if $g^z = y^c \cdot a$.

Schnorr's Protocol's Simulator

1. The Prover* uses the Verifier's random number generator to generate the Verifier's challenge c.
2. The Prover* randomly generates the response z.
3. The Prover* calculates $a = g^z \cdot y^{-c}$.
4. Run the protocol from protocol step 2, skipping step 4, instead using the generated z.

Protocol 3.1: Schnorr's Protocol

3.4.2 Zero Knowledge OR (Protocol 3.2)

This is an essential for efficiently proving compound statements. The statement being proved is "At least one of these statements is true." It does not reveal which of the statements is true. Zero Knowledge OR is essential if we wish our protocol to be modular. The main conceptual point about this OR protocol is that it allows all but one of the challenges to be predetermined by the Prover the others to be based on input from the Verifier without revealing which is which, allowing the Prover to use the Honest Verifier Simulator to create an acceptable transcript using the protocol's associated simulator σ_i. It should be noted that the initial communications a and responses z is not necessarily one value if the associated protocol calls for more than one initial communication and response. Using Zero Knowledge Or encourages a modular design, so if the protocol needs to be applied in another setting, the larger protocol can still be used as long as the component protocols can be designed for the new setting. The strength of Zero Knowledge Or is when there are a only few possible cases, as it can link them together without adding too many exponentiations. Simulators tend to take more exponentiations than true proofs. The function call notation for OR is as follows: $\mathsf{OR}((\pi_1, \text{condition}_1),...(\pi_n, \text{condition}_n))$ for each ZKP π_i in the OR. The condition is the requirement for running the true proof instead of the simulator. Zero Knowledge OR appears in Hazay and Lindell's book [13], but their proof exists for two proofs that are the same relation. Our proof is a generalization of their proof that covers many differing component protocols.

3.4.3 Zero Knowledge AND (Protocol 3.3)

The purpose of Zero Knowledge AND is to condense two Zero Knowledge Proofs in to one Sigma Protocol. By using the same challenge for multiple different ZKP's, it can be used

in conjunction with Zero Knowledge OR to generate a statement like "Either (S_1 AND S_2) OR S_3". The statement associated with Zero Knowledge AND would be of the form "All of these statements are true." This necessary for the modular design to function, as many cases require multiple facts to be true. The function call notation is $\mathsf{AND}(\pi_1, \pi_2, ..., \pi_n)$.

3.4.4 Protocol 3.4: Bit Verification

One essential Zero Knowledge Proof is the Bit Verification Proof, a proof that a commitment hides a bit ($\mathcal{C}_b = g^b \cdot h^\gamma$). This is essential because the protocol uses bit-wise comparisons. This will use Zero Knowledge OR, with the statement "This commitment hides a 0 OR this commitment hides a 1." Its function call notation is $\mathsf{BV}(\mathcal{C}_b, b, \gamma)$, which is calls the following construction: $\mathsf{OR}((\mathsf{CVP}(\mathcal{C}_b, 0, \gamma), b = 0), (\mathsf{CVP}(\mathcal{C}_b, 1, \gamma), b = 1))$

3.4.5 Proof of Equal Discrete Logs (PEDL) [13] (Protocol 3.5)

As the name suggests, this is a proof where, given two bases g and h and two values $y_1 = g^x$ and $y_2 = h^x$, a Prover can prove knowledge of an x that fulfills this relation. It is important to note that the simulator can still simulate a successful response when the discrete logs are not equal. The statement being proven is "The discrete log of y_1 base g and y_2 base h are equal and I know what it is." The function call notation is $\mathsf{PEDL}(y_1, y_2, x)$.

Proving Encryption is a Rerandomization (PER)

The main purpose for the Proof of Equal Discrete Logs protocol in this thesis is to prove proper rerandomization of an Elgamal Encryption. If a Prover takes an encryption $e_0 = (g^m \cdot y^{\alpha_1}, g^{\alpha_1})$ and rerandomizes it using α_2 to obtain $e_1 = (g^m \cdot y^{\alpha_1+\alpha_2}, g^{\alpha_1+\alpha_2})$ and publishes the rerandomization, they can prove that $e_1 \cdot {e_0}^{-1} = (y^{\alpha_2}, g^{\alpha_2})$ and use Protocol 3.5 to prove the two components are equal. The function call notation for this proof

is $\mathsf{PER}(e_0, e_1, \alpha_2)$, where α_2 is the randomization factor, which calls PEDL as follows: $\mathsf{PEDL}(e_2(m) \cdot {e_1}^{-1}(m), e_2(k) \cdot {e_1}^{-1}(k), \alpha_2)$

3.5 Mix and Match Table

A Mix and Match table is an encrypted logic table. An equality test, like PET, is used to determine which of the rows matches the inputs to retrieve outputs. Often, these tables are used to simulate circuits. Between each use of a Mix and Match table, the table needs to be shuffled and rerandomized using a Verifiable Mix Network.

3.5.1 Plaintext Equality Test [15]

This test can use any cryptoscheme that is homomorphic and can have a distributed key, like Elgamal. For Elgamal, if you have two cyphertexts $[\![m_1]\!]$ and $[\![m_2]\!]$, then all parties take $[\![m_1]\!] \cdot [\![m_2]\!]^{-1} = [\![m_1 - m_2]\!]$. All parties then exponentiate their version with a previously committed random value, then they are multiplied together. Then, the parties jointly decrypt. If the encrypted value is a 1, then they are equal. If not, then it is a random number and they are unequal.

1. Each party computes $[\![m_1 - m_2]\!] = [\![m_1]\!] \cdot [\![m_2]\!]^{-1}$.
2. Each party commits a random value r_i and published the commitment.
3. Each party calculates $[\![(m_1 - m_2)^{r_i}]\!] = ([\![m_1 - m_2]\!])^{r_i}$.
4. Each party exchanges their $[\![(m_1 - m_2)^{r_i}]\!]$ to the other parties.
5. Each party runs a Zero Knowledge Proof to prove that the number hidden in the commitment is equal to the number they used to exponentiate the encryption.

6. The encryptions from each party multiplies all the encryptions together:
   $\prod_{i=1}^{n} [\![(m_1 - m_2)^{r_i}]\!] = \left[\!\!\left[(m_1 - m_2)^{\sum_{i=1}^{n} r_i}\right]\!\!\right] = [\![(m_1 - m_2)^{r}]\!]$

7. The parties run a Distributed Decryption scheme on the resulting encryption. If the number $(m_1 - m_2)^r = 1$, then $(m_1 - m_2) = 0$, meaning $m_1 = m_2$. If $(m_1 - m_2)^r$ equals anything else, then $m_1 \neq m_2$ because $(m_1 - m_2) \neq 0$.

Zero Knowledge OR Protocol

Input:	Public:	A list of statements $\mathcal{S}$ with each statement S_i's associated zero knowledge protocol π_i, the associated simulator σ_i, as well as the inputs for each of the protocols.
	Private (Prover):	The information to genuinely prove at least one of the two statements S_i.
Output:	Public:	Verification that at least one of the statements is true.

1. For each j such that the Prover will not attempt to prove S_j: The Prover uses σ_j to pregenerate the initial message a_j, the sub-challenge c_j, and the response z_j.
2. For each i such that the Prover will to prove S_i: The Prover uses π_i to calculate the initial message a_i.
3. The Prover sends all the initial communications to the Verifier.
4. The Verifier randomly generates a challenge c and sends it to the Prover.
5. Take the XOR of all the simulated sub-challenges with the challenge c and call it c_t.
6. For all but the last true proofs: Generate a random challenge c_i and calculate $c_t = c_t \oplus c_i$
7. For the last of the true proof: Set $c_i = c_t$
8. For each i where S_i is a true proof, the Prover uses each c_i with the associated π_i to calculate z_i.
9. The Prover sends the list of responses and sub-challenges to the Verifier.
10. The Verifier accepts if all of the following are true:
    (a) The XOR of all the sub-challenges is equal to c
    (b) For each protocol π_i: (a_i, c_i, z_i) is a valid transcript for π_i, given the protocol's public inputs.

Zero Knowledge OR Simulator

1. The Prover* pregenerates the Verifier's challenge c
2. The Prover* runs protocol step 1 for each protocol π_i except the last one, use σ_i to generate the a_i, c_i, z_i.
3. The Prover* calculates the last sub-challenge c_i as c XORed with all the other sub-challenges and uses the simulator σ_i to generate z_i and calculate a_i.
4. The Prover* and Verifier run the protocol from protocol step 3, skipping any step with true proofs.

Protocol 3.2: Zero Knowledge OR

Zero Knowledge AND Protocol

Input:	Public:	The list of statements, and for each statement S_i, the associated protocols π_i and the protocols inputs.
	Private (Prover):	The information to prove each statement S_i.
Output:	Public:	Verification that both statements are true.

1. For each statement S_i, the Prover uses π_i to calculate the initial communications a_i.
2. The Prover sends the list of initial communications to the Verifier.
3. The Verifier randomly generates a challenge c and sends it to the Prover.
4. For each statement S_i, the Prover uses c with π_i to calculate a response z_i.
5. The Prover sends the list of responses to the Verifier.
6. The Verifier accepts if, for each statement S_i, the transcript (a_i, c, z_i) is a valid transcript for π_i

Zero Knowledge AND Simulator

1. The Prover* uses the Verifier's random number generator to generate his challenge c
2. For each statement S_i, the Prover* uses the protocols' associated simulators σ_i to generate z_i and calculates a_i.
3. The Prover* and Verifier execute the protocol starting at protocol step 2, skipping step 4, instead using the list of generated responses.

Protocol 3.3: Zero Knowledge AND

Bit Verification Protocol

Input:	Public:	A commitment $C_b = g^b h^\gamma$
	Private (Prover):	Knowledge of b and γ
Output:	Public:	Verification that the commitment $\mathcal{C}_b$ hides either zero or one.

1. The Prover and Verifier will prove the statement "C_b hides 0 OR C_b hides 1" by executing Protocol 3.2 to perform a Zero Knowledge OR on the following two Zero Knowledge Protocols:
   - CVP (Protocol 3.4.1) to prove C_b hides 0.
   - CVP (Protocol 3.4.1) to prove C_b hides 1.

Bit Verification Simulator

1. The Prover* and Verifier run protocol step 1 except with Zero Knowledge Or's Simulator instead of Zero Knowledge Or Protocol.

Protocol 3.4: Bit Verification

Proof of Equal Discrete Logs

Input:	Public:	The exponentials $y_1 \coloneqq g^x$ and $y_2 \coloneqq h^x$
	Private (Prover):	x, the discrete log of y_1 base g and y_2 base h.
Output:	Public:	Verification that the Prover knows x and that it is the discrete log of both y_1 and y_2 in their respective bases.

1. The Prover generates a random number r and calculates $a_1 \coloneqq g^r$ and $a_2 \coloneqq h^r$
2. The Prover sends a_1 and a_2 to the Verifier.
3. The Verifier generates a random challenge c and sends it to the Prover.
4. The Prover computes $z \coloneqq r + c \cdot x$.
5. The Prover sends z to the Verifier.
6. The Verifier accepts if $g^z = {y_1}^c \cdot a_1$ and $h^z = {y_2}^c \cdot a_2$.

Proof of Equal Discrete Logs Simulator

1. The Prover* uses the Verifier's random number generator to generate the Verifier's challenge c.
2. The Prover* randomly generates the response z.
3. The Prover* calculates $a_1 \coloneqq g^z \cdot {y_1}^{-c}$ and $a_2 \coloneqq h^z \cdot {y_2}^{-c}$
4. Run the protocol from step 2, skipping step 4, instead using the generated z.

Protocol 3.5: Proof of Equal Discrete Logs

Chapter 4

VARIANCE PROTOCOL

4.1 Solution Overview

Our solution to this problem is a three step scheme based on perfectly hiding commitments.

1. ZK-OR Proof of Assets: First, a Prover has to make a verified commitment of a sum of their accounts. To accomplish this, we create commitments related to each account in the blockchain or anonymity set. They are individually verified and added up to the sum of the owned accounts.

2. Bit Conversion: Once the commitment of the assets is complete, then it needs to be converted to commitments of bits so that a bitwise comparison can be accomplished. The Prover will create a series of bit commitments and prove that when put together that they are equal two the sum of assets commitment.

3. Zero Knowledge Comparison: We employ Mix and Match to select a table to which each party proves their data is consistent.

4.2 ZK-OR Proof of Assets

To accomplish a Proof of Assets, we will use the conceptual approach used by Provisions's Proof of Assets, where a commitment is made for each individual account based on own-

ZK-OR Proof of Assets Protocol

Input:	Public:	Generator g, Distributed Public Key h, The list of accounts L, where each account is a pair (y_i, B_i).
	Private (Prover):	A set of owned private keys $x_i \in \mathbb{Z}_q$ such that $g^{x_i} = y_i$ for account i.
Output:	Public:	A verified commitment of the sum of the Prover's claimed accounts $\mathcal{C}_{\text{sum}}$

1. For each account (y_i, B_i) in list of accounts L:
   (a) If the prover owns the account, owning the account meaning he has knowledge of x_i, and wishes to claim it then the Prover creates a commitment $C_{\hat{B}_i} \coloneqq g^{B_i} h^{\gamma_i}$. Else, the Prover creates a commitment $C_{\hat{B}_i} \coloneqq g^0 h^{\gamma_i}$
   (b) The Verifier requires the Prover to prove the following statement: "Either the commitment $C_{\hat{B}_i}$ hides B_i AND the Prover knows the secret key x_i OR the commitment $C_{\hat{B}_i}$ hides 0." To accomplish this, the Prover and Verifier execute Protocol 3.2 to perform Zero Knowledge OR on the following two Zero Knowledge Protocols, creating the Account Verification Proof:
      - The statement "The commitment $C_{\hat{B}_i}$ hides B_i AND the Prover knows the secret key x_i" is proven by using Protocol 3.3 to perform Zero Knowledge AND on the following two protocols:
        - CVP (Protocol 3.4.1) to prove $\mathcal{C}_{\hat{B}_i}$ hides B_i.
        - Schnorr's Protocol (Protocol 3.1) to prove that the Prover knows the private key x_i associated with y_i. (Can be substituted for the appropriate key proof when applicable).
      - CVP (Protocol 3.4.1) to prove $\mathcal{C}_{\hat{B}_i}$ hides 0.
2. Now, the Verifier has a list of $|L|$ length of commitments of $\hat{B}_i$, each of which either holds a 0 or the balance of account i, and it has been verified that if it holds the balance, then the Prover knows the private key.

   Using the additive homomorphism of Pedersen Commitments:

$$\mathcal{C}_{\text{sum}} \coloneqq \prod_{i=1}^{n} \mathcal{C}_{\hat{B}_i} = \prod_{i=1}^{n} g^{\hat{B}_i} h^{\gamma_i} = g^{\sum_{i=1}^{n} \hat{B}_i} h^{\Gamma}, \text{ where } \Gamma \coloneqq \sum_{i=1}^{n} \gamma_i. \tag{4.1}$$

Protocol 4.1: ZK-OR Proof of Assets

ership of the account and then homomorphically add them together to get a sum. Our protocol has the advantage of greater flexibility and efficiency compared to Provisions's Proof of Assets.

To verify the individual accounts, this thesis applies Protocol 3.2 (OR) and Protocol 3.3 (AND) to prove the statement to be true: "(Either the amount hidden in this commitment is equal to the balance of the account AND I know the secret key for the account) OR the amount hidden in this commitment is equal to 0." To accomplish this, we specifically construct the Zero Knowledge Proof using Pedersen Commitments and Elgamal keys, so the Zero Knowledge Proof will prove the following statement:

"(Either I know the ephemeral key γ_i for the Pedersen Commitment $C_{\hat{b}_i}$, assuming $\hat{b}_i = b_i$ (Assuming $C_{\hat{b}_i}) = g^{b_i}h^{\gamma_i}$) AND I know the private key x_i for the public key $y_i = g^{x_i}$) OR I know the ephemeral key γ_i for the commitment $C_{\hat{b}_i}$, assuming $\hat{b}_i = 0$ (Assuming $C_{\hat{b}_i} = g^0 h^{\gamma_i}$)". We call the following proof Account Verification Proof.

OR((AND(Schnorr(y_i, x_i), CVP($C_{\hat{b}_i}, b_i, \gamma_i$)), x is known), (CVP($C_{\hat{b}_i}, 0, \gamma_i$)), x is not known))

The commitments and the key can be based on different schemes as long as zero knowledge proofs can be created to prove the individual statements. For example, the public key can be RSA if you have a zero knowledge proof of knowledge of an RSA public key. The public key can also use a different curve and generators than the commitments. In Variance, we specifically work with a discrete log based public keys and Pedersen commitments. To accomplish this, we shall make the statement to prove more specific: "Either the amount hidden in this commitment is equal to the balance of the account (Protocol 3.4.1) AND I know the discrete log of the public key (Protocol 3.1) OR the amount hidden in this commitment is equal to 0 (Protocol 3.4.1)."

Using Protocol 4.1, the values are proven to be correct, which means that if $C_{\hat{B}_i}$ holds a non-zero value, then the party has to hold the associated private key. Both parties end up

with a commitment of the sum of the other party's claimed accounts:

$$\mathcal{C}_{\text{sum}} = \prod_{\text{owned}} \mathcal{C}_{B_i} = \prod_{i=1}^{n} \mathcal{C}_{\hat{B}_i} \tag{4.2}$$

Each party also knows their ephemeral key for their $\mathcal{C}_{\text{sum}}$.

$$\Gamma := \sum_{i=1}^{n} \gamma_i \tag{4.3}$$

The two parties will execute this protocol in an interweaved manner, meaning each of the parties perform each step as both Prover and Verifier before continuing to the next step.

4.3 Conversion to Bits

To compare the sums, the two parties need to convert their sums in to a binary representation. Each party commits each individual bit in the bitwise representation of their sum. Normally, we would have to prove that each commitment hides a bit, but that fact is a side effect of Protocol Then, we raise each commitment by 2^j, where j is the bit position. This shifts the bit to the appropriate bit position. The Verifier then uses the homomorphic addition property of Pedersen Commitments by multiplying these shifted bit commitments together. This product equals the $\mathcal{C}_{\text{sum}}$ from the previous section. This is accomplished in Protocol 4.2. The two parties will execute this protocol in an interweaved manner, meaning each of the parties perform each step as both Prover and Verifier before continuing to the next step.

These should be equal if the commitments are set up properly because $\sum_{i=0}^{n-1} \eta_i \cdot 2^i = \Gamma$. Now, we have a bit representation of both party's sums, which will be compared in the Zero Knowledge Comparison Protocol.

Bit Conversion Protocol

Input:	Public:	Generator g, Distributed Public Key h, commitment of the sum $\mathcal{C}_{\text{sum}}$
	Private (Prover):	The number hidden in $\mathcal{C}_{\text{sum}}$, including a bitwise representation, where the ith bit is b_i and sum is equal to $\sum_{i=0}^{n} b_i \cdot 2^i$.
Output:	Public:	A verified bitwise representation of the sum.

1. The parties agree on a number of bits n with which to represent their numbers, creating an implicit upper bound of $2^n - 1$.
2. The Prover will create a Pedersen commitment for each bit except the least significant bit.
   (a) Generate a random ephemeral key η_i
   (b) Generate commitment $\mathcal{C}(b_i) = g^{b_i} h^{\eta_i}$
3. The Prover will create a Pedersen commitment for the least significant bit.
   (a) Let Γ be the ephemeral key, or random number, from the commitment $\mathcal{C}_{\text{sum}}$
   (b) Calculate number $\eta_0 = \Gamma - \sum_{i=1}^{n-1} 2^i \cdot \eta_i$ so that $\Gamma = \sum_{i=0}^{n-1} 2^i \cdot \eta_i$
   (c) Generate commitment $\mathcal{C}_{b_0} = g^{b_0} h^{\eta_0}$
4. Prover sends commitments to Verifier.
5. The Prover and Verifier confirm that the each commitment is a bit by executing the Bit Verification Protocol (Protocol 3.4).
6. The Verifier uses the additive homomorphism of Pedersen Commitments and accepts if $\prod_{i=0}^{n-1} (\mathcal{C}_{b_i})^{2^i} = \mathcal{C}_{\text{sum}}$, assuming the Bit Verification Protocol's succeed.

Protocol 4.2: Bit Conversion

On the last commitment in the protocol (Step 3), we construct the commitment using the specific method because the other party can get that commitment from the more significant bit commitments and the sum commitment easily and it saves time by not requiring a Zero Knowledge Proof that two commitments are equal.

Table 4.1: Conceptual Model.

$[\![<]\!]$	0	0	$[\![\frac{\leq}{4}]\!]$	——>	$[\![=]\!]$	0	0	$[\![0]\!]$
	0	1	$[\![\frac{\leq}{4}]\!]$			0	1	$[\![\frac{\leq}{2}]\!]$
	1	0	$[\![\frac{\leq}{4}]\!]$			1	0	$[\![\frac{\geq}{2}]\!]$
	1	1	$[\![\frac{\leq}{4}]\!]$			1	1	$[\![0]\!]$
$[\![=]\!]$	0	0	$[\![0]\!]$		$[\![<]\!]$	0	0	$[\![\frac{\leq}{4}]\!]$
	0	1	$[\![\frac{\leq}{2}]\!]$			0	1	$[\![\frac{\leq}{4}]\!]$
	1	0	$[\![\frac{\geq}{2}]\!]$			1	0	$[\![\frac{\leq}{4}]\!]$
	1	1	$[\![0]\!]$			1	1	$[\![\frac{\leq}{4}]\!]$
$[\![>]\!]$	0	0	$[\![\frac{\geq}{4}]\!]$		$[\![>]\!]$	0	0	$[\![\frac{\geq}{4}]\!]$
	0	1	$[\![\frac{\geq}{4}]\!]$			0	1	$[\![\frac{\geq}{4}]\!]$
	1	0	$[\![\frac{\geq}{4}]\!]$			1	0	$[\![\frac{\geq}{4}]\!]$
	1	1	$[\![\frac{\geq}{4}]\!]$			1	1	$[\![\frac{\geq}{4}]\!]$

4.4 Zero Knowledge Comparison Protocol

To perform the comparison, we will execute the Zero Knowledge Comparison Protocol: We also accomplish this with Zero Knowledge Proofs and homomorphism as the computational tools.

To accomplish this, we use a feature that is common in Elliptical Curve groups, including secp256k1, the order of the group is prime. That means that in exponential elgamal on this curve, you can encrypt fractions with denominators less than or equal to the order. This means if we let "less than" $= -1$, "equal" $= 0$, and "greater than" $= 1$, we can construct the two tables illustrated in Table 4.1. The first column of the inner tables correspond with the bit of party 1 and second column similarly corresponds with party 2. We call the table associated with 0 the Equals table, the table associated with -1 the Less Than table, and the table associated with 1 the Greater Than table.

The actual model that would be implemented in the form of a true Mix and Match table

is illustrated in Table 4.2.

Table 4.2: Actual Model

Input	0 0	0 1	1 0	1 1
$[\![-1]\!]$	$[\![-\frac{1}{4}]\!]$	$[\![-\frac{1}{4}]\!]$	$[\![-\frac{1}{4}]\!]$	$[\![-\frac{1}{4}]\!]$
$[\![0]\!]$	$[\![0]\!]$	$[\![-\frac{1}{2}]\!]$	$[\![\frac{1}{2}]\!]$	$[\![0]\!]$
$[\![1]\!]$	$[\![\frac{1}{4}]\!]$	$[\![\frac{1}{4}]\!]$	$[\![\frac{1}{4}]\!]$	$[\![\frac{1}{4}]\!]$

Since the feedback (current state of the comparison) at the first bit is equal, there is no reason to shuffle the table before the first comparison, which will save a little time. The selected row begins as the Equals table or $[\![0]\!]$. After the first bit of the comparison, the feedback is calculated homomorphically based on each party's bit and the selected table.

For each commitment in the list of commitments, starting with most significant:

1. Take the table from the selected row.
2. Each party creates 2 encryptions based the rows of the selected table.
    - For each row in the selected table, if the bit on the party's column is equal to the bit from their bit commitment, rerandomize the resulting encryption.
3. Each party sends the rerandomized encryptions to the other party.
4. Prove that the encryptions are valid using the Bit Table Verification Proof (equation 4.4):
    - $\mathsf{BTVP}(x_1, x_2, y_1, y_2, i)$, where x_1 and x_2 correspond with the rows on the selected table that would match 0 and y_1 and y_2 correspond to the rows that match 1 on the party's column.

5. Multiply all the encryptions (P_1's and P_2's) together to homomorphically obtain the feedback.

6. Shuffle the table using the method from Mix and Match.

7. Use PET to determine which row from the shuffled outer table corresponds with the feedback. The row that the feedback matches with is the new selected row.

Bit Table Verification Proof

The Zero Knowledge Proof for each possibility will be structurally the same. The general ZKP for the party's encryption corresponding to the bit commitment C_{b_i} hiding a specific bit b is as follows:

$\mathsf{p}_1(x_1, x_2, i, b)$ =AND(CVP($\mathcal{C}_{b_i}, b, \eta_i$), PER($row_{x_1}, e_0, \alpha_1$), PER($row_{x_2}, e_1, \alpha_2$))

We compose two instances of p_1 with Zero Knowledge OR to create the Bit Table Verification Proof (BTVP) to prove that one of the two possible cases is true ($b = 0$ OR $b = 1$):

$$\mathsf{BTVP}(x_1, x_2, y_1, y_2, i) = \mathsf{OR}((\mathsf{p}_1(x_1, x_2, i, 0), b_i = 0), (\mathsf{p}_1(y_1, y_2, i, 1), b_i = 1)) \quad (4.4)$$

After the comparison protocol has been successfully concluded, the two parties take the resulting feedback and run a verified distributed decrypt on the feedback. If the result is -1, P_2 has the larger amount of bitcoins. If the result is 0, they have equal bitcoins. Otherwise, P_1 has more bitcoins. The protocol then concludes.

It should be noted that once the feedback (the product of the encryptions) stops matching $[\![0]\!]$, then all subsequent bits remain perfectly-hidden and can not be determined even with infinite computational power, assuming truly random numbers were used.

Chapter 5

SECURITY ANALYSIS

5.1 Zero Knowledge Proofs

We begin our proof of Variance's security with proofs that the sigma protocols are zero knowledge protocols, as it is a prerequisite to the majority of the requirements of security.

To prove that a sigma protocol is an Honest Verifier Zero Knowledge Proof (HVZKP), the protocol must satisfy the following three properties [13]:

1. **Completeness:** Does a honest Prover with proper inputs pass every time? If proper inputs result in the Verifier accepting, it is complete. Completeness is usually simple, simply following the steps of the protocol assuming both parties are honest.

2. **Special Soundness:** Is the secret present? Given one initial communication and two challenges, can the secret be extracted? The extractor (Verifier*) takes the role of the Verifier and has the ability to rewind the Prover while maintaining knowledge of the Prover's previous communication, thus allowing the Verifier* to send the second challenge and get second response. The Verifier* then can extract the information from the transcripts. If a protocol is specially sound, then the probability of a Prover successfully cheating is 2^{-t}, where t is the number of bits in the challenge.

3. **Honest Verifier Zero Knowledge:** Could the Verfier have created the transcript on their own? Does there exist an algorithm (simulator) in the semi-honest model that

can create a proof transcript that is impossible to distinguish from a proper transcript? The simulator (Prover*) takes over the role of the Prover and knows the challenge beforehand. If the Prover* can create a valid transcript using only the inputs of the Verifier, then the protocol reveals nothing to an honest Verifier.

Once a Sigma Protocol is proven to be a Honest Verifier Zero Knowledge Proof, it can be converted to a malicious Zero Knowledge Proof by either using a perfectly hiding commitment of the challenge before the initial communication [13], or by using the Fiat-Shamir random oracle heuristic [7] to eliminate the Verifier's input by making the protocol non-interactive.

We will only prove *completeness* and *special soundness* in this section for each of the sigma protocols, as we have already proven *Honest Verifier Zero Knowledge* by providing the honest simulators with the protocols in Chapter 3.

5.1.1 Schnorr's Protocol

The proof for Schnorr's protocol is relatively simple, so we will begin with it to demonstrate the structure of the proof.

Completeness:

To prove completeness, we will assume an honest Prover. If the Prover knows x such that $y = g^x$:

1. The Prover randomly generates r and calculates g^r and sends it to the Verifier.

2. The Verifier sends a random t-bit challenge c to the Prover.

3. The Prover calculates $z = r + c \cdot x$

Schnorr's Protocol Extractor

1. The Prover and Verifier* run Protocol 3.1 from the beginning to end, resulting in the transcript (a, c_1, z_1)
2. The Verifier* rewinds the Prover to Step 3 of the protocol and sends a second challenge c_2.
3. The protocol resumes to completion, resulting in the transcript (a, c_2, z_2)
4. If both transcripts are valid, the Verifier* calculates $\frac{z_1 - z_2}{c_1 - c_2} = \frac{(r + c_1 \cdot x) - (r + c_2 \cdot x)}{c_1 - c_2} = \frac{c_1 \cdot x - c_2 \cdot x}{c_1 - c_2} = \frac{(c_1 - c_2) \cdot x}{c_1 - c_2} = x$; otherwise, the extractor fails.

Protocol 5.1: Schnorr's Protocol Extractor

4. The Prover sends z to the Verifier.

5. The Verifier calculates g^z and $a \cdot y^c$. Because the Prover is honest, $g^z = g^{r+cx} = g^r \cdot g^{xc} = a \cdot y^c$. Since $g^z = a \cdot y^c$, the Verifier accepts.

Special Soundness:

To prove that the protocol is specially sound, we provide the extractor, Protocol 5.1. Since the extractor can extract the secret x, the protocol is specially sound.

5.1.2 Equal Discrete Log Proof

The proof for the Equal Discrete Log Proof is very similar to Schnorr, as the protocol is based on Schnorr's Protocol.

Completeness

To prove completeness, we will assume an honest Prover. If the Prover knows x such that $y_g = g^x$ and $y_h = h^x$:

1. The Prover randomly generates r and calculates $a_g = g^r$ and $a_h = h^r$ and sends it to the Verifier.

2. The Verifier sends a challenge c to the Prover.

3. The Prover uses x to calculate $z = r + c \cdot x$

4. The Prover sends z to the Verifier.

5. The Verifier calculates g^z, which is equal to $g^{r+cx} = g^r \cdot (g^x)^c = a \cdot {y_g}^c$.

6. The Verifier calculates h^z, which is equal to $h^{r+cx} = h^r \cdot (h^x)^c = a \cdot {y_h}^c$.

Since $g^z = a \cdot {y_g}^c$ and $h^z = a \cdot {y_h}^c$, the Verifier accepts.

Special Soundness

To prove that EDLP is specially sound, we provide an extractor (Protocol 5.2). Since the extractor can extract the secret x, the protocol is specially sound.

5.1.3 Zero Knowledge AND

The proof for the Zero Knowledge AND Protocol's proof requires that all the protocols are Zero Knowledge Protocols in the form of Sigma Protocols. The proof relies on each component protocol's completeness, extractor, and simulator.

Equal Discrete Log Proof Protocol Extractor

1. The Prover and Verifier* run Protocol 3.5 from the beginning to end, resulting in the transcript $((a_g, a_h), c_1, z_1)$
2. The Verifier* rewinds the Prover to Step 3 of the protocol and sends a second challenge c_2.
3. The protocol resumes to completion, resulting in a second transcript $((a_g, a_h), c_2, z_2)$
4. If both transcripts are valid, the Verifier* calculates $\frac{z_1 - z_2}{c_1 - c_2} = \frac{(r + c_1 \cdot x) - (r + c_2 \cdot x)}{c_1 - c_2} = \frac{c_1 \cdot x - c_2 \cdot x}{c_1 - c_2} = \frac{(c_1 - c_2) \cdot x}{c_1 - c_2} = x$; otherwise, the extractor fails.

Protocol 5.2: Equal Discrete Log Proof Protocol Extractor

Completeness

The Prover starts with all the inputs required to properly execute all component protocols.

1. The Prover runs each component protocol's initial communication step to acquire a list of initial communications.
2. The Prover sends the list of initial communications to the Verifier.
3. The Verifier sends a challenge c to the Prover.
4. The Prover uses the component protocols to calculate a list of responses.
5. The Prover sends the list of responses to the Verifier.
6. For each component protocol π_i, the Verifier uses the respective initial communication and response with the challenge on π_i. Since the component protocols are complete, the Verifier accepts each transcript.

Zero Knowledge AND Extractor

1. Run Protocol 3.3 to completion, getting the Verifier*'s first challenge (c_1) as well as a list of transcripts (T_1)
2. The Verifier* rewinds the Prover to Step 3 of the protocol and sends a second challenge c_2.
3. The protocol resumes to completion, resulting in a second list of transcripts (T_2).
4. If any of the transcripts are invalid with their respective protocol, the extractor aborts and fails.
5. For each component protocol π_i, select the associated transcripts $T_1(i)$ and $T_2(i)$. The Verifier* uses the protocol π_i's extractor with the transcripts $T_1(i)$ and $T_2(i)$ to obtain the secret associated with π_i.

Protocol 5.3: Zero Knowledge AND Extractor

Special Soundness

We assume that each component protocol is specially sound and has an extractor. Zero Knowledge AND's extractor is provided as Protocol 5.3 to prove that this composition is specially sound.

5.1.4 Zero Knowledge OR

This proof resembles the Zero Knowledge AND proof, but is a bit more complicated. Like Zero Knowledge AND, the Zero Knowledge OR Protocol's proof requires that all the component protocols are Zero Knowledge Protocols in the form of Sigma Protocols. This means that the component protocols are complete and have simulators and extractors.

Completeness

The Prover starts with the inputs to properly prove at least one of the component protocols.

1. The Prover splits the list of protocols in to two sub-lists: protocols to be run properly (P) and protocols to be simulated ($\mathcal{S}$). The Prover simulates if he does not have the proper inputs.

2. For each protocol π_j in the list $\mathcal{S}$, the Prover generates a sub-challenge c_j. The Prover then use the associated simulator σ_j to randomly generate the responses z_j and the calculate the initial communication a_j.

3. For each protocol π_i in the list P, they run the protocol π_i to calculate the initial communication.

4. The Prover sends the list of initial communications to the Verifier.

5. The Verifier sends a challenge c to the Prover.

6. The Prover XORs each sub-challenge c_j together, then XORs that number with the Verifier's challenge c, which will be called the total sub-challenge (c_t).

7. For all but one of the protocols π_i in P, the Prover randomly generates a sub-challenge c_i and XORs that challenge in to c_t, creating a new c_t.

8. For the remaining protocol π_i in P, set c_i equal to the resulting c_t.

9. For each protocol π_i in P, use the protocol π_i to calculate the response with the associated sub-challenge c_i.

10. The Prover sends the list of responses and the list of sub-challenges to the Verifier.

11. The Verifier takes the XOR of all of the sub-challenges, and sees that the result is equal to c.

12. The Verifier goes through each protocol π_i:

    (a) If the Prover used the protocol's simulator σ_i, then the Verifier will accept, as the simulator is assumed to be valid.

    (b) If the Prover used the proper protocol, then the Verifier will accept, as the protocol is complete and the Prover used proper inputs.

Therefore, Zero Knowledge OR is complete.

Special Soundness

We assume that each component protocol is specially sound and has an extractor. Each transcript from the component protocols contains the initial communication, the sub-challenge, and the response (a_i,c_i,z_i) from that protocol. The extractor is provided as Protocol 5.4. Since the extractor extracts the used secrets, Zero Knowledge OR is specially sound.

5.2 Security Definition

In the following, we define the security goals of Variance:

Definition 1. *A privacy-preserving bitcoin asset comparison protocol is a protocol that fulfills the following criteria:*

1. ***Correctness**. If P_1 and P_2 are honest, then the comparison is always accurate.*

2. ***Privacy**. A dishonest party (or an eavesdropper) cannot learn any information about the other party's list of selected accounts or their total sum.*

Zero Knowledge OR Extractor

1. Run Protocol 3.2 to completion, getting the Verifier*'s first challenge (c_1) as well as a list of transcripts (T_1).
2. The Verifier* rewinds the Prover to Step 4 of the protocol and the Verifier* sends a second challenge c_2.
3. The protocol resumes to completion, resulting in a second list of transcripts (T_2).
4. The Verifier* verifies both lists of transcripts are valid, as well as the sub-challenges. If any of the transcripts are invalid, the extractor fails and aborts.
5. For each component protocol π_i, select the corresponding transcripts $T_1(i)$ and $T_2(i)$. If the sub-challenges from $T_1(i)$ and $T_2(i)$ are not equal, then the Verifier* will run π_i's extractor, using the transcripts $T_1(i)$ and $T_2(i)$ to extract the Prover's secret associated with the protocol.

Protocol 5.4: Zero Knowledge OR Extractor

3. ***Ownership***. *Each party is assured that the other is cannot include any bitcoins that they do not own in the concealed sum such that the probability of a dishonest party successfully cheating in the construction of the concealed sum is extremely low ($< 2^{-30}$).*

4. ***Sound Comparison***. *Assuming the concealed sum is accurate, the probability of a dishonest party successfully cheating in the comparison is extremely low ($< 2^{-30}$).*

5.3 Security Proofs

We will now prove the following theorem:

Theorem 5.3.1. *Variance is a privacy-preserving bitcoin asset comparison protocol.*

To prove this theorem, we will prove that each of the statements in Definition 1 applies to Variance. We will also provide a supplementary proof of security as provided by Hazay and Lindell [14] called Proof by Simulation, which is also a proof of the security definition.

5.3.1 Basic Component Proofs

Lemma 1. *Variance is correct.*

Proof: This proof relies on the three different phases of the protocol all being correct:

ZK-OR Proof of Assets

If the commitments are designed according to the protocol, then the homomorphisms allow the Proof of Assets produce the appropriate sum. Also, the Account Verification Proof is a Zero Knowledge Protocol, thus it is also complete. Each party will accept the other party's proofs, allowing the protocol continue to the next step with the appropriate committed sums.

Bit Conversion

If the bit commitments are designed according to the protocol, then the product of the shifted bit commitments $\prod_{i=0}^{n-1} {\mathcal{C}_{b_i}}^{2^i}$ is equal to the original sum commitment, thus the bit commitments are accepted if they are proven to be bits. This is the case because $\sum_{i=0}^{n-1} \eta_i \cdot 2^i = \Gamma$. Since the commitments hide a bitwise representation of the sum S, $\sum_{i=0}^{n-1} b_i \cdot 2^i = S$, thus $\prod_{i=0}^{n-1} {\mathcal{C}_{b_i}}^{2^i} = \prod_{i=0}^{n-1} \left(g^{b_i} \cdot h^{\eta_i}\right)^{2^i} = g^{\sum_{i=0}^{n-1} b_i \cdot 2^i} \cdot h^{\sum_{i=0}^{n-1} \eta_i \cdot 2^i} = g^S \cdot h^\Gamma = \mathcal{C}_{\text{sum}}$. The Bit Verification Proof is a Zero Knowledge Proof, therefore it is complete. Therefore, the parties will accept the each other's proofs, agreeing that the other party's bit commitments hide a bitwise representation of the sum, thus allowing the protocol to continue.

Zero Knowledge Comparison

First, the Mix and Match protocol, including PET, is complete, as proven by Jakobsson and Juels [15]. Next, the parties rerandomize the correct encryptions based on their current bit and the logic of the table. Recall that the encryptions are additive elgamal and homomorphically add the messages when multiplied. We will now the feedback will be correct in every case.

- On the Equals ($[\![0]\!]$) table:

  1. P_1's bit is 0, P_2's bit is 0, $[\![0]\!] \cdot [\![-\frac{1}{2}]\!] \cdot [\![0]\!] \cdot [\![\frac{1}{2}]\!] = [\![0]\!]$. The bits are equal, so the feedback should match the Equals table. This is the result of the homomorphism.

  2. P_1's bit is 0, P_2's bit is 1, $[\![0]\!] \cdot [\![-\frac{1}{2}]\!] \cdot [\![-\frac{1}{2}]\!] \cdot [\![0]\!] = [\![-1]\!]$. P_1's bit is less than P_2's, so the feedback should match the Less Than table. This is the result of the homomorphism.

  3. P_1's bit is 1, P_2's bit is 0, $[\![\frac{1}{2}]\!] \cdot [\![0]\!] \cdot [\![0]\!] \cdot [\![\frac{1}{2}]\!] = [\![1]\!]$. P_1's bit is greater than P_2's bit, so the feedback should match the Greater Than table. This is the result of the homomorphism.

  4. P_1's bit is 1, P_2's bit is 1, $[\![\frac{1}{2}]\!] \cdot [\![0]\!] \cdot [\![-\frac{1}{2}]\!] \cdot [\![0]\!] = [\![0]\!]$. The bits are equal, so the feedback should match the Equals table. This is the result of the homomorphism.

- On the Less Than ($[\![-1]\!]$) table, if any group of 4 encryptions, which are all $[\![-\frac{1}{4}]\!]$ are rerandomized and homomorphically added, the result will be $[\![-1]\!]$, which leads back to the Less Than table after the shuffle, as intended.

- On the Greater Than ($[\![1]\!]$) table, if any group of 4 encryptions, which are all $\left[\!\left[\frac{1}{4}\right]\!\right]$ are rerandomized and homomorphically added, the result will be $[\![1]\!]$, which leads back to the Greater Than table after the shuffle, as intended.

Also, BTVP is a Zero Knowledge Proof, so it is complete, so the parties will accept the other's proofs. They then run a distributed decrypt on the last feedback, which returns the relation between P_1 and P_2. Therefore, Variance is correct. □

Lemma 2. *Variance maintains privacy.*

Proof: This claim contains two parts. Proof that the accounts are private and proof that the sum is private.

Proof of Private Accounts

ZK-OR Proof of Assets is the only part of the protocol which handles the individual accounts. As long as the sum is not released by any of the other steps, those steps are not relevant to this proof.

ZK-OR Proof of Assets uses perfectly-hiding commitments, which are verified using the Account Verification Proof. This proof is a Zero Knowledge Proof, so it is Zero Knowledge and has a simulator. Due to the nature of the proof, it is impossible to determine whether the account is claimed or not if the sum is not revealed, since it it possible to have two equal transcripts, one in which the account claimed and one in which the account is not claimed. The perfectly-hiding commitments reveal nothing about the balance due to their perfectly-hiding nature.

Proof of Private Sum

ZK-OR Proof of Assets does not reveal the accounts, so the only other component is the homomorphic addition of the commitments, which is also perfectly-hiding.

The Bit Conversion protocol is based on perfectly hiding commitments and homomorphic operations. There exists message-key pairs for a bit in the sum for bitwise commitment to hide a 1 or a 0, thus there is no way to differentiate between them. This leaves the Bit Verification Proof, which is a Zero Knowledge Proof, thus it is Zero Knowledge and has a simulator. Therefore, the Bit Conversion protocol is privacy-preserving.

This leaves the Comparison Protocol. The shuffle and PET is privacy-preserving under the ECDLP, as proven by Jakobsson and Juels in their paper [15]. This leaves the encryptions and the proof. The encryptions are semantically secure, so their contents are protected by ECDLP. BTVP is a Zero Knowledge Proof, and thus is Zero Knowledge has a simulator. However, many of the bits lose their perfect-hiding during this phase. If ECDLP gets solved, all bits up to and including the first different bit from the most significant bit can be revealed, as the parties' encryptions become trivial to open and can be matched to the owner' bit on the equals table. All of the less significant bits remain perfectly-hidden, as the less than and greater than tables both have identical encryptions on all the rows, thus the decryptions reveal nothing about the claimed bits. Therefore, the more significant bits of the sum up to and including the first differing bit are kept private under ECDLP, while the remaining less significant bits remain perfectly hidden, thus are private.

Therefore the sum and the accounts are private, therefore Variance maintains privacy. □

Argument for Strong Account Privacy

We designed the protocol in such a way some of the sum remains perfectly hidden after the comparison, specifically from the most significant bits up to and including the first differing bit. In the event that ECDLP becomes solved, we would like to explore what can be kept secret. Though ECDLP being broken means the bulk of the information about the sum is revealed, the lack of information about the less significant bits actually makes deducing the accounts much harder. This is because every account commitment has two possible message-key combinations: (balance and key_1) and (0 and key_2). If the entire sum were to be revealed, that means that the adversary knows the sum and can easily figure out the associated key using the sum and $\mathcal{C}_{\text{sum}}$. That means that, if there are 10 million accounts, there is a linear system of equations with 20 million variables that is tractable and has a polynomial runtime ($\text{O}(n^3)$), so the accounts would be revealed. Since the sum is not fully revealed, this attack on the accounts are much harder, if not impossible, in most cases.

Lemma 3. *Variance ensures ownership.*

Proof: ZK-OR Proof of Assets uses homomorphisms, which do not make it easier to cheat on the sum commitments. The Account Verification Proof is a Zero Knowledge Proof, so it is specially sound, so it protects against cheating with a probability of successfully cheating at 2^{-t} for each account, where t is the number of bits in the challenge. Since we use perfectly-hiding commitments, they are only computationally binding. This means that they are bound by ECDLP. As long as the discrete log is not solved, the commitment is bound. Therefore, each party's sum is bound to only use accounts to which that party holds the key with a very high probability by ECDLP.

Therefore, Variance ensures ownership. □

Lemma 4. *Variance's comparison is a sound comparison.*

Proof: This proof consists of two parts: Bit Conversion and ZK Comparison.

Bit Conversion uses homomorphisms, which do not reveal anything the original commitments have not already revealed. The Bit Verification proof is a Zero Knowledge Proof, thus it is specially sound (meaning the probability of cheating on any bit is 2^{-t}, where t is the number of challenge bits). The last thing is to prove that the bitwise construction itself is sound. Recall that the homomorphism we use is $\prod_i^n \mathcal{C}_{(b_i,\eta_i)}{}^{2^i}$. This operation bitshifts the message. If one of the bits is invalid, then the ephemeral key on the least significant bit does not result in equality, as the message is different and requires a total ephemeral key that is different from Γ. Since this key is not known, the cheating party has no way to prove with probability greater than 2^{-t} that the final bit in the number is a bit because they do not know the appropriate ephemeral key unless they have broken the discrete log. Therefore, the Bit Conversion is sound under ECDLP.

Mix and Max is sound, as is PET. BTVP is a Zero Knowledge Proof and thus is specially sound, thus successfully cheating on the rerandomization is extremely improbable, 2^{-t}. The homomorphisms on the encryptions are perfectly binding, as encryptions are perfectly binding. The bit commitment is bound with ECDLP.

Variance's comparison is sound under ECDLP. □

5.3.2 Proof by Simulation

Simulator Proof: To ensure privacy, we first present simulator protocols for each of the main components of the protocol, then we present an overarching simulator to show security for the entire protocol by sequential composition. The parties are symmetric, so the simulator can simulate the view of either party. It should be noted that if an internal simulator (a simulator that the Simulator has called and is running) aborts, the main simulation also aborts. The Simulator $\mathcal{S}$ interacts with the Adversary $\mathcal{A}$. Since $\mathcal{S}$ is a simulator,

their commitments and encryptions are simply random numbers, and thus will be called "pseudo."

Ideal Model

First, we describe the actions of the *trusted third party* (TTP) in the Ideal Model.

1. Each party gives the TTP their list of private account keys.
2. The TTP finds each of the associated public keys and creates a sum for each party.
3. The TTP returns the relation between P_1's sum and P_2's sum.

ZK-Proof of Assets Simulator

Public Inputs: The set of accounts, where each account is a pair (public key, balance)

Public Outputs: $\mathcal{A}$'s total sum commitment, $\mathcal{S}$'s total sum pseudo commitment, $\mathcal{A}$'s list of account commitments, $\mathcal{S}$'s list of account pseudo commitments.

Extracted Data: $\mathcal{A}$'s list of used private keys and the message-key pairs for all of $\mathcal{A}$'s commitments.

The Simulator is as follows:

1. For each account in the set of accounts:
   (a) $\mathcal{S}$ generates a random group element as its account pseudo commitment and sends it to $\mathcal{A}$.
   (b) $\mathcal{S}$ receives a group element from $\mathcal{A}$ ($\mathcal{A}$'s account commitment)
   (c) $\mathcal{S}$ runs the Account Verification Proofs with $\mathcal{A}$. The proofs can be run in any order.

- $\mathcal{S}$, acting as the Prover, runs the Account Verification Proof's simulator with $\mathcal{A}$. If $\mathcal{A}$ accepts, continue.
- $\mathcal{S}$, acting as the Verifier, runs the Account Verification Proof's extractor with $\mathcal{A}$. If the extractor succeeds, the values γ_i, $\hat{b}_i$, and $\hat{x}_i$. If $\hat{x}_i \neq 0$, then $\hat{x}_i = x_i$, and x_i is recorded as one of $\mathcal{A}$'s used private keys.
- If the extractor fails, $\mathcal{S}$ sends abort to the TTP and sends the failed challenge to $\mathcal{A}$.

2. $\mathcal{S}$ homomorphically adds $\mathcal{A}$'s account commitments to obtain $\mathcal{A}$'s sum commitment. $\mathcal{S}$ also sums $\mathcal{A}$'s $\hat{b}_i$ values to obtain the claimed assets and sums γ_i values to obtain Γ_i.

3. $\mathcal{S}$ homomorphically adds their account pseudo commitments to obtain $\mathcal{S}$'s sum pseudo commitment. If $\mathcal{A}$ accepts the homomorphic combination, then $\mathcal{S}$ continues. Else, $\mathcal{S}$ aborts.

Bit Conversion Simulator

Public Inputs: $\mathcal{A}$'s commitment, $\mathcal{S}$'s pseudo commitment, and the number of agreed bits n

Public Outputs: $\mathcal{A}$'s list of verified bit commitments and $\mathcal{S}$'s list of verified bit pseudo commitments.

Extracted Data: The contents of all of $\mathcal{A}$'s commitments, including the bits, the original hidden amount, and all of the random keys.

The Simulator is as follows:

1. $\mathcal{S}$ randomly generates $n-1$ group representing the all but the least significant bit. $\mathcal{S}$ calculates the last element to correspond with the homomorphic relation prescribed in the protocol.

2. $\mathcal{S}$ sends the list of n group elements (bit pseudo commitments) to $\mathcal{A}$.

3. $\mathcal{S}$ attempts to receive n group elements from $\mathcal{A}$. If he sends anything other than n group elements, $\mathcal{S}$ aborts.

4. For each index i from 0 to $n-1$:

   (a) Run Bit Verification Proofs. These proofs can be run in either order.

   - $\mathcal{S}$, acting as the Prover, runs the Bit Verification simulator with $\mathcal{A}$ on $\mathcal{S}$'s ith bit pseudo commitment. If $\mathcal{A}$ rejects the simulation, $\mathcal{S}$ aborts.
   - $\mathcal{S}$, acting as the Verifier, runs the Bit Verification extractor with $\mathcal{A}$ on their ith bit commitment to extract $\mathcal{A}$'s ith bit key pair for their bit commitment. If the extractor fails, S sends the failed challenge and aborts.

5. $\mathcal{S}$ homomorphically combines $\mathcal{A}$'s bit commitments as prescribed in the protocol. If the product is equal to $\mathcal{A}$'s total commitment, $\mathcal{S}$ tells $\mathcal{A}$ to continue.

6. $\mathcal{S}$ homomorphically combines its own bit commitments as prescribed by the protocol. If $\mathcal{A}$ does not tell $\mathcal{S}$ to continue, $\mathcal{S}$ aborts.

Zero Knowledge Comparison

Public Inputs: $\mathcal{A}$'s bit commitments and $\mathcal{S}$'s bit pseudo commitments, as well as the initial unshuffled tables, as well as $\mathcal{S}$ and $\mathcal{A}$'s public keys.

Public Outputs: The result of the comparison.

Optional Simulator Data (from calling a simulator): The result of the comparison m and $\mathcal{A}$'s private key x.

Extracted Data: The contents of all of $\mathcal{A}$'s bit commitments, including the bits and all of the random keys, as well as the rerandomization values α and $\mathcal{A}$'s private key x.

The Simulator is as follows:

1. Build the initial table (unshuffled). Select the "Equals" table

2. For each bit i starting from most significant:

   (a) $\mathcal{S}$ randomly generates two pairs of group elements and sends them to $\mathcal{A}$.

   (b) $\mathcal{S}$ attempts to receive two pairs of group elements. If any other data is sent, $\mathcal{S}$ aborts. Else, $\mathcal{S}$ continues.

   (c) Run Bit Table Verification Proof: The order of the proofs is irrelevant.

   - $\mathcal{S}$, acting as the Prover, uses BTVP's simulator with $\mathcal{A}$ using $\mathcal{S}$'s ith bit commitment, the two random pairs, and the selected table as inputs. If $\mathcal{A}$ accepts, $\mathcal{S}$ continues. Else, $\mathcal{S}$ aborts.
   - $\mathcal{S}$, acting as the Verifier, uses BTVP's extractor with $\mathcal{A}$, using the $\mathcal{A}$'s ith bit commitment, the $\mathcal{A}$'s group element pairs, and the selected table as inputs. If successful, $\mathcal{S}$ extracts the bits and the rerandomization exponents and continues. Else, $\mathcal{S}$ aborts.

   (d) Homomorphically combine all four pairs as exponential Elgamal encryptions, called feedback.

   (e) Run Mix and Match's Verifiable Mix Network's simulator to replace the existing tables with random group elements and ensure $\mathcal{A}$ shuffles properly.

   (f) Simulate PET with the feedback to select a single random row. The table in that row is the new selected table.

3. If the result of the comparison is unknown, give the extracted bits to the TTP to get the appropriate result.

4. Take the last feedback (R_1, R_2), where R_1 is the ephemeral public key.

5. $\mathcal{S}$ simulates a distributed decryption such that the result is m. $\mathcal{S}$ also extracts $\mathcal{A}$'s private key. If the simulation is successful, $\mathcal{S}$ continues.

Variance Simulator

Public Inputs: List of accounts, as well as $\mathcal{S}$ and $\mathcal{A}$'s public keys.

Public Outputs: The result of the comparison.

Extracted Data: $\mathcal{A}$'s private key, claimed accounts, total assets, and all random keys from the commitments.

The Simulator is as follows:

1. $\mathcal{S}$ runs the Proof of Assets simulator with $\mathcal{A}$, using the list of accounts. If the simulation is successful, $\mathcal{S}$ extracts the list of $\mathcal{A}$'s used private account keys. The simulator outputs $\mathcal{A}$'s sum commitment and $\mathcal{S}$'s sum pseudo commitment as public information.

2. $\mathcal{S}$ uses the TTP with $\mathcal{A}$'s used private account keys to determine the proper comparison result.

3. $\mathcal{S}$ runs the Bit Conversion Simulator with $\mathcal{A}$, using $\mathcal{A}$'s sum commitment and $\mathcal{S}$'s sum pseudo commitment. The simulator outputs $\mathcal{A}$'s list of bit commitments and $\mathcal{S}$'s list of bit pseudo commitments.

4. $\mathcal{S}$ runs the Zero Knowledge Comparison Simulator with $\mathcal{A}$, using $\mathcal{A}$'s list of bit commitments and $\mathcal{S}$'s list of bit pseudo commitments.

5. If all of the simulators succeed. $\mathcal{S}$ sends a continue to the TTP. Else, $\mathcal{S}$ sends an abort to the TTP.

5.4 Potential Security Threat & Solutions

The use of Zero Knowledge OR could result in a significant attack on the security of the algorithm through analysis of the timings of responses. If the simulator takes a significantly different amount of time than the standard protocol, then an opponent may deduce which provers were utilized in an asymmetric Zero Knowledge OR, like the Account Verification Proof, which, if he is correct, reveals which of the side of a Zero Knowledge OR is true, which would reveal whether or not an account is claimed. This threat can easily be mitigated by adding having the protocol utilize waits so that each step takes the same amount of time. Since, given simulators and provers use approximately the same resources throughout the entire execution, this can also be mostly mitigated by the use of the Fiat-Shamir Random Oracle heuristic [7] to make the zero knowledge proofs non-interactive. It may still be necessary release each proof after a constant amount of time regardless.

Chapter 6

EXPERIMENTAL EVALUATION AND PERFORMANCE ANALYSIS

6.1 The Setting

We implemented Variance to measure the performance of the algorithm. We utilize Bouncy Castle to handle the elliptic curve operations. We created our implementation in Java. We implemented our own Zero Knowledge Proof interface for this system. Our ZKProver interface includes a parallel-prove, where two parties act as both prover and verifier simultaneously on the same protocol using their own data. Zero Knowledge AND and OR are implemented by providing a list of protocols. All Zero Knowledge Proofs are performed with a pre-committed challenge to ensure security against a malicious verifier. Though many parts of this protocol are easily parallelizable with threads, each party uses a single threaded program to better establish a benchmark. As a rule, inputs for the Zero Knowledge Proofs were created as needed and discarded when no longer needed to allow the program to maintain a reasonably sized heap and a small footprint on RAM. The challenges used have 255 bits.

6.1.1 Proof of Assets

Since we are solving the same problem as Provisions's Proof of Assets, we will be comparing our performance to the Provisions algorithm. Since we do not have access to Provisions's source code, we implemented the protocol using ZKProver. This allowed us to run both protocols implemented using the same general methods and running them under the same conditions. All experiments are done assuming secp256k1 public keys, though Variance can handle other key types.

Computational Complexity

Every account requires one run of the Zero Knowledge Proof and one homomorphic addition for each account. Each individual homomorphic addition is a simple point addition, which is independent of the number of accounts, so the individual homomorphic additions are O(1). The instances of the Zero Knowledge Proof are independent of each other and the execution of the proof is independent of the number of accounts, so each individual Zero Knowledge Proofs are O(1). Thus with respect to the number of accounts (n_a), the computational complexity of the Proof of Assets is O(n_a). This analysis is also supported by the experiments, which show a linear growth as the number of accounts increase.

Theoretical Comparison

To compare the theoretical difference between Variance's Proof of Assets and Provision's Proof of Assets, we will be looking at three important and potentially expensive components: Exponentiations, Communications, and Random Numbers generated. We will be conducting a basic count of each for proving that the commitments are consistent with one account. We count any exponentiation where the exponent is a random number that can be

up to 256 bits. We count communications if they require a transfer of 256 bits. It should be noted that Bouncy Castle has a compressed point format which converts an elliptical curve point in to an approximately 256 bit representation. We count random numbers if they generate up to 256 bits.

	Exps	Comms	RNGs
Variance	15	13	8
Provisions	27	17	13

	Exps	Comms	RNGs
Variance	16	13	8
Provisions	27	17	13

Variance's Proof of Assets theoretically requires less resources in every major respect than Provisions's Proof of Assets. This implies that Variance's Proof of Assets will run faster and with a smaller proof size than Provisions. There is an asymmetry between whether the account is claimed or not. We discuss the ramifications of this in the Security Analysis section on Timing Analysis, as well as a solution. Our experiment did not implement any solution to this issue.

It should also be noted that both Proofs of Assets are Embarrassingly Parallel, which means that the tasks can easily be parallelized without having to compensate for data interdependencies. If each party were to execute a properly threaded version of the implementation on n cores, the execution time could be expected to decrease by a factor of $\frac{1}{n}$.

Experimental Results

We executed this protocol on a list of single-key accounts using secp256k1 curve.

To properly compare the results of Variance to the results from Provisions, we ensured that they were optimized in the same way and were operating on the same system by implementing both protocols using the same scheme. To mirror the Prosivions experiment

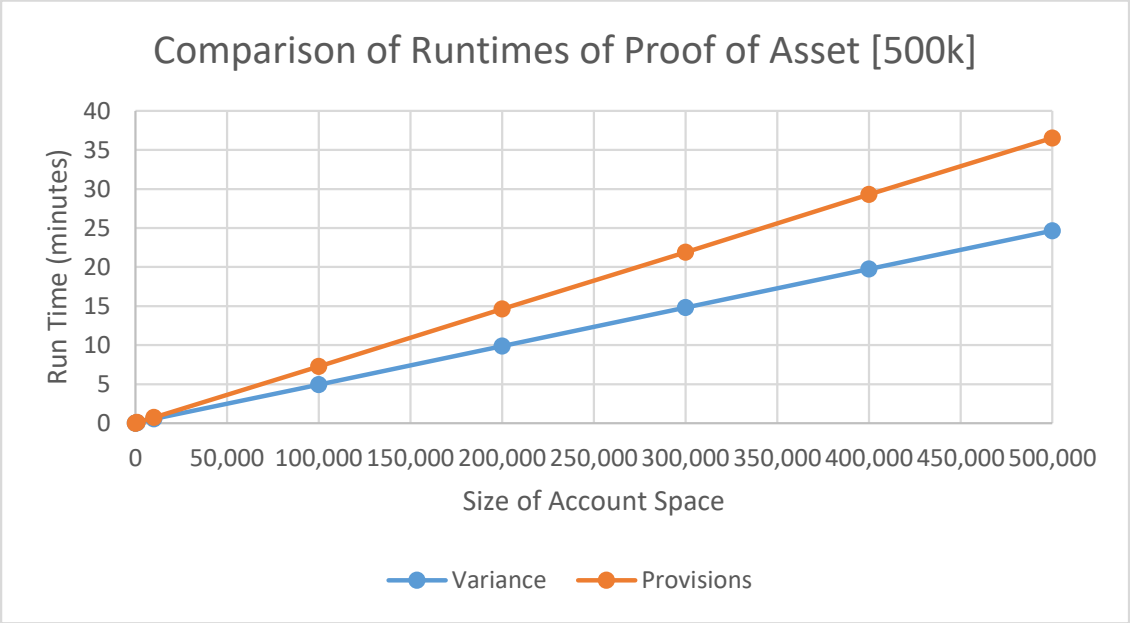

Figure 6.1: Comparison of runtimes of proofs of assets up to 500,000 accounts.

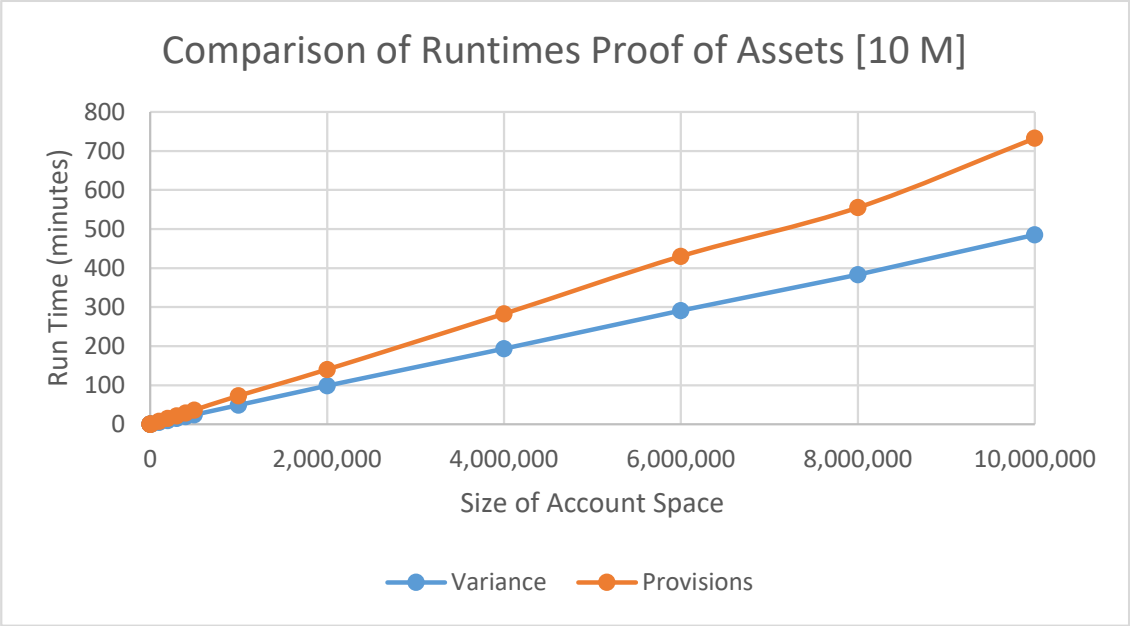

Figure 6.2: Comparison of runtimes of proofs of assets up to 10,000,000 accounts.

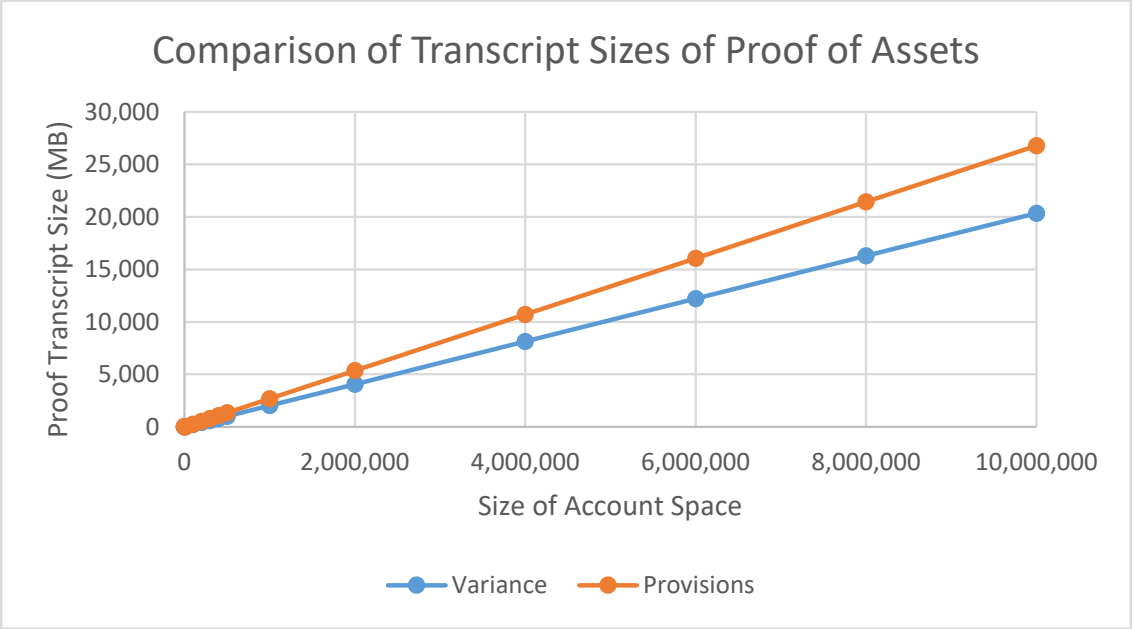

Figure 6.3: Comparison of transcripts sizes of proofs of assets up to 10,000,000 accounts.

for comparison, we include Figure 6.1. It should be noted that our implementation of Provisions's protocol was faster than the implementation used in their paper. It presents the data up to an account space of up to 500,000. It clearly demonstrates a linear trend. Figure 6.2 illustrates an account space of 10,000,000 to show the linear trend continues. This shows that it is reasonable to run Variance 8 hours overnight on the entire blockchain, whereas Provisions's proof would require over 12 hours. This is a non-threaded edition of the protocol, which is *embarrassingly parallel*, so the time could easily be reduced by a factor of $\frac{1}{n}$, where $n =$ MIN(Party 1's cores, Party 2's cores)

We will now compare the size of the proofs by measuring the Verifier's transcript. The data is charted on Figure 6.3. We observe that Variance's transcripts are 1.3 times smaller than Provisions's transcripts, which saves on records space.

These performance improvements, compounded with the increased flexibility of the Variance Proof of Assets, emphasize the superiority of Variance Proof of Assets com-

pared to Provisions's.

6.1.2 Bit Conversion and Comparison

Computational Complexity

This portion of the protocol consists of two loops with the number of bits in the comparison (n_b) iterations. The contents of the first loop consists of creating a bit commitment, a point addition, and a Bit Verification Protocol. These are constant time operations, therefore the contents of the loop is O(1). That means the loop as a whole is O(n_b).

The second loop consists of 2 encryptions, 2 elgamal rerandomizations, a large zero knowledge proof, and a mix and match iteration, which includes a shuffle and a PET. According to Jakkobson and Juels [15], one mix and match iteration is constant time with respect to everything but the size of the table, which is constant, and the number of parties, of which there are two. All the other components are also constant time with respect to number of bits in the comparison, therefore the computational complexity the inside of the loop is O(1). This puts the second loop at O(n_b).

Experimental Results

We ran the comparison protocol at 20 bits and it executed in 1.647 seconds. This amount of time is negligible compared to the Proof of Assets.

Chapter 7

CONCLUSION

7.1 Summary

In this thesis, we design a two-party protocol that allows two parties to determine who owns more bitcoin without revealing which accounts are owned or anything about the total sum except the relation between the two sums.

The tools used in this protocol allow for an efficient modular design. This modular design allowed for a more intuitive design that is also more efficient, direct, and flexible. The ability to handle accounts secured in a different setting, including multisig accounts, could be very useful for making a usable protocol. The experiments showed our modular design for the Proof of Assets was about 1.5 times more efficient in runtime than the less flexible design from Provisions. Being able to use Zero Knowledge Proofs in this modular design also is more consistent with the object-oriented paradigm.

This technique can be applied to real world problems in Bitcoin, such as auctions and credit checks. Unlike most of the literature, this thesis explores not only the verifying power of Zero Knowledge Proofs, but their ability to be a fundamental computational component of a protocol.

7.2 Looking Ahead

There are several further experimental tests and improvements that could improve this protocol.

7.2.1 Proof of Assets

It would be interesting to have an experiment involving multisig accounts, which have multiple keys. These accounts would be harder to work with, as proving ownership of the account involves more proofs of keys. Some accounts with n keys allow transfers if the party holds k keys, i.e. a 3 key account that requires 2 keys for a transaction. To prove ownership of such an account, it would require $\binom{n}{k}$ components to the Zero Knowledge Proof, which can get very expensive. For example, the proof for owning 2 of 3 keys would be "(I own key 1 AND I own key 2) OR (I own key 1 AND I own key 3) OR (I own key 2 AND I own key 3)". This proof has 6 "I know the key" proofs, which is manageable. If $n = 10$ and $k = 5$, the number of individual proofs would be 252, which is still manageable, but slow. These larger proofs are rare, but they are also very expensive.

It would also be interesting to see the performance of proof of knowledge of the private key of the preimage of the hash in this protocol. This would allow the protocol to include non-spending accounts in the sum. Such a proof would likely be expensive, but would allow for a complete sum.

Parallelizing the implementation of the protocol using threads would be good to demonstrate the scalability with cores. It would also be interesting to attempt to implement this protocol for use on a GPU. Use of CUDA could allow these proofs to be executed in large batches, which would improve the efficiency significantly.

7.2.2 Zero Knowledge Comparison

It would be good for the novelty of the protocol to find a way to replace the PET in the comparison. Though it functions properly, it does not fit the theme that the only interactions are Zero Knowledge Proofs or simple data transfers.

Further work could involve refining the comparison to increase efficiency and comparing its multi-party computational efficiency to Mix and Match. Specifically, if the comparison in the Zero Knowledge Proof was converted to be non-interactive with the Fiat-Shamir heuristic [7]. With a brief examination at the problem, PET appears more efficient than our Zero Knowledge Comparison for two parties. However, PET increases in complexity with the number of parties, but a random oracle Zero Knowledge Proof remains constant, which means that if the comparison has a large number of verifiers, our Zero Knowledge Comparison may become more efficient than PET.

Bibliography

[1] Gilad Asharov, Yehuda Lindell, and Hila Zarosim. Fair and efficient secure multiparty computation with reputation systems. *Advances in Cryptology - ASIACRYPT 2013 Lecture Notes in Computer Science*, page 201220, 2013.

[2] Certicom Research. SEC 2: Recommended Elliptic Curve Domain Parameters, Version 1.0., 2000.

[3] Ronald Cramer, Ivan Damgård, and Berry Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In *CRYPTO*, 1994.

[4] G Dagher, B Bünz, Joseph Bonneau, Jeremy Clark, and D Boneh. Provisions: Privacy-preserving proofs of solvency for bitcoin exchanges (full version). Technical report, IACR Cryptology ePrint Archive, 2015.

[5] Gaby G. Dagher, Benedikt Bünz, Joseph Bonneau, Jeremy Clark, and Dan Boneh. Provisions: Privacy-preserving proofs of solvency for bitcoin exchanges. In *Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security*, CCS '15, pages 720–731, 2015.

[6] Ivan Damgård. On σ-protocols, 2010.

[7] Amos Fiat and Adi Shamir. Witness indistinguishable and witness hiding protocols. In *ACM STOC*, 1990.

[8] U. Fiege, A. Fiat, and A. Shamir. Zero knowledge proofs of identity. In *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*, STOC '87, pages 210–217, New York, NY, USA, 1987. ACM.

[9] Juan Garay, Berry Schoenmakers, and José Villegas. Practical and secure solutions for integer comparison. In *Proceedings of the 10th International Conference on Practice and Theory in Public-key Cryptography*, PKC'07, pages 330–342, Berlin, Heidelberg, 2007. Springer-Verlag.

[10] Oded Goldreich. *Foundations of Cryptography: Volume 2, Basic Applications*. Cambridge University Press, 2004.

[11] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to prove all np-statements in zero-knowledge, and a methodology of cryptographic protocol design. In *Proceedings on Advances in cryptology—CRYPTO '86*, pages 171–185, London, UK, UK, 1987. Springer-Verlag.

[12] S Goldwasser, S Micali, and C Rackoff. The knowledge complexity of interactive proof-systems. In *Proceedings of the Seventeenth Annual ACM Symposium on Theory of Computing*, STOC '85, pages 291–304, New York, NY, USA, 1985. ACM.

[13] Carmit Hazay and Yehuda Lindell. *Efficient Secure Two-Party Protocols*. Springer, 2010.

[14] Carmit Hazay and Yehuda Lindell. *Efficient Secure Two-Party Protocols Techniques and Constructions*. Springer Berlin, 2013.

[15] Markus Jakobsson and Ari Juels. Mix and match: Secure function evaluation via ciphertexts. In *ASIACRYPT*, 2000.

[16] Markus Jakobsson, Ari Juels, and Ronald L. Rivest. Making mix nets robust for electronic voting by randomized partial checking. In *Proceedings of the 11th USENIX Security Symposium (USENIX)*, 2002.

[17] Neal Koblitz. Elliptic curve cryptosystems. *Mathematics of computation*, 48(177):203–209, 1987.

[18] Lindell and Pinkas. Privacy preserving data mining. *Journal of Cryptology*, 15(3):177–206, Jun 2002.

[19] Adriana López-Alt, Eran Tromer, and Vinod Vaikuntanathan. On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption. In *Proceedings of the forty-fourth annual ACM symposium on Theory of computing*, pages 1219–1234. ACM, 2012.

[20] S Nakamoto. Bitcoin: A peer-to-peer electionic cash system. Unpublished, 2008.

[21] Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *CRYPTO*, 1992.

[22] Fergal Reid and Martin Harrigan. *An Analysis of Anonymity in the Bitcoin System*, pages 197–223. 2013.

[23] C P Schnorr. Efficient signature generation by smart cards. *Journal of Cryptography*, 4, 1991.

[24] Vassilios S. Verykios, Elisa Bertino, Igor Nai Fovino, Loredana Parasiliti Provenza, Yucel Saygin, and Yannis Theodoridis. State-of-the-art in privacy preserving data mining. *SIGMOD Rec.*, 33(1):50–57, March 2004.

[25] Andrew C. Yao. Protocols for secure computations. In *IEEE FOCS*, 1982.