Variance: Secure Two-Party Protocol for Efficient Asset Comparison in Bitcoin

Joshua Holmes
Computer Science Department
Boise State University

Gaby G. Dagher
Computer Science Department
Boise State University

***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; however, these solutions are either insecure in the malicious model or cannot verify the validity of inputs. In this paper, we introduce** Variance**, a privacy-preserving two-party protocol for solving Yao's millionaires' problem in a Bitcoin setting, in which each party controls several Bitcoin accounts (single and multi signature addresses) and they want to find out who owns more bitcoins without revealing (1) how many accounts they own or the addresses associated with their accounts, (2) the balance of any of 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 zero knowledge proofs to provide a solution to the problem, and subsequently prove that** Variance **is secure against active adversaries in the malicious model.**

***Index Terms*—Bitcoin; blockchain; proof of asset;**

1. Introduction

In this paper, we introduce an efficient and privacy-preserving solution to Yao's millionaires' problem [1] in a Bitcoin context, which could serve as a building block for solving real-world problems in Bitcoin such as credit checks, auctions, and proving solvency of exchanges. In Bitcoin, an account is presented as a public address associated with a balance and users hold the associated private keys to their accounts [2]. To determine who owns more bitcoin, it is not sufficient to simply compare two numbers. Owners of bitcoin usually store their holdings in multiple accounts. A protocol involving a user's accounts ideally should 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. Our proposed protocol, 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].

1.1. 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, the bitcoins associated with the owned accounts have to be added for any real comparison to take place. A privacy-preserving protocol that solves this problem should keep secret (1) the list of the owned accounts, i.e. the address and balance of each owned account and the number of owned accounts, (2) the total sum, and (3) which of the m-of-n keys were used in a multisig account. The first requirement is *unlinkablity* [3], meaning a party cannot be linked to their accounts and vice versa. Due to the *pseudonymity* of Bitcoin, a commitment based scheme using perfectly-hiding commitments would be preferable [4]. It would be best to have as much information about the accounts and the total assets be perfectly hidden as possible, so that the forward security of this protocol is not as dependent on the discrete log problem. Even obscuring small details of the total assets may prevent the discovery of claimed accounts. As the Bitcoin protocol is designed to be secure against malicious users, any solution to this problem should also be secure in the malicious model.

1.2. Contributions

The contributions of this paper are as follows:

1) We introduce Variance, a privacy-preserving protocol with which two parties can compare their assets in Bitcoin. Our modular design allows for Variance to be used in different cryptocurrency settings by substituting the zero knowledge proofs' components.
2) We design an efficient method where a Bitcoin user can create a verified commitment of their total assets of bitcoins. Unlike existing proof of assets [3], our method supports both *single key* and *multisig* accounts.
3) We introduce a novel commitment-based bit comparison protocol to compare the total asset commitments.
4) We prove that Variance is privacy-preserving, and demonstrate empirically its efficiency and scalability.

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

2. Related Work

2.1. Proof of Assets in Bitcoin

There have been several methods of proving how much bitcoin a person holds [3] [5] [6] [7]. 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. However, these signatures reveal the account holder's owned accounts, which is not desirable. In 2011, the Bitcoin Exchange MtGox wanted to prove that they had a large stockpile of bitcoin. To accomplish this, they executed one large transfer of Ƀ 420k. Though this action did prove they had Ƀ 420k, it was certainly not optimal, as it revealed the size of MtGox, the amount of bitcoin they control, and the addresses of their holdings. It also proved nothing about the amount of Bitcoin they owed. These proofs of assets all are technically effective, but they do not meet our goals because they reveal the accounts held and how much money is in each account.

Provisions [3] 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. 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 cannot be mixed), which is less than optimal. Their solution would also require a significant overhaul to become capable of handling multisig accounts. We attempt to improve on their conceptual method by making it portable, more efficient, modular, and more intuitive.

3. Preliminaries 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. $\mathsf{secp256k1}$ [8]. It should be noted that if the cofactor of the curve is 1, then every point excluding the identity 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 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 $Y = xG$ and $g \cdot h$ instead of $G + H$.

3.2. Elliptic Curve Exponential Elgamal Cryptosystem

Elgamal is a semantically secure asymmetric cryptosystem based on the discrete log problem. It is noted for its semantic security, which gives users the ability to randomize existing ciphertexts without changing the plaintext and the ability of multiple parties to create a distributed private key. We use a variant of Elgamal which allows for homomorphic addition and is executed on elliptic curves [9]. The encryption function will be written as $[\![m]\!] = (g^m y^r, g^r)$, with generator g, public key $y = g^x$ where x is the private key. The calculations are executed on a specified curve (e.g. $\mathsf{secp256k1}$), which has an order q. Elgamal also supports multiplication of the ciphertext to be translated in to addition of the plaintext. The main limitation of exponential Elgamal is that a complete decryption requires the discrete log to be solved for generic messages. This weakness does not effect $\mathsf{Variance}$, as all ciphertexts within the protocol hide a small number of potential values, allowing us to simply recognize the elliptic curve point to know the discrete log.

3.3. Pedersen Commitments

Pedersen Commitments [10] are a perfectly hiding and computationally binding commitment scheme based on any system with an associated hard Discrete Log Problem. We use Elliptic Curve Pedersen Commitments. 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 an entirely accurate statement when referring to Pedersen commitments, 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 a pair of values m and α that match the commitment. In this paper, we use the elliptic curve $\mathsf{secp256k1}$. We refer to m as the message and α as the ephemeral key, as α strongly resembles the ephemeral key from Elgamal, though it does not follow the same constraints against reuse as Elgamal's ephemeral key.

3.4. Zero Knowledge Proofs

For $\mathsf{Variance}$, we will be constructing larger zero knowledge proofs from the foundational zero knowledge proofs. We will utilize a notation do describe these constructions of zero knowledge proofs that will be of the following form:

$$\mathsf{OR}(\mathsf{Schnorr}(y_1), \mathsf{Schnorr}(y_2))$$

3.4.1. Schnorr's Protocol

Schnorr's Protocol 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. When used in a construction, we refer to this protocol in compound proofs as follows: $\mathsf{Schnorr}(y)$.

Commitment Value Proof (CVP)

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 with 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 if not. This proof is simple for a homomorphic commitment scheme. We are specifically using Pedersen commitments, so this protocol is simple. We refer to this proof in compound proofs as follows: $\mathsf{CVP}(\mathcal{C}_{(m,\alpha)}, m)$), where m is the message and α is the ephemeral key. This proof is equivalent to $\mathsf{Schnorr}(\mathcal{C}_{(m,\alpha)} \cdot g^{-m})$.

3.4.2. Proof of Equal Discrete Logs (PEDL) [11]

As the name suggests, this is a proof where, given two generators 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 "I know x such that $y_1 = g^x$ and $y_2 = h^x$." This proof is referred to in compound proofs as follows: $\mathsf{PEDL}(y_1, y_2)$.

Proof of Elgamal Randomization (PER)

Proof of Equal Discrete Log's purpose in $\mathsf{Variance}$ is to prove proper randomization of Elgamal ciphertexts. If a Prover takes an encryption $e = (g^m \cdot y^{\alpha_1}, g^{\alpha_1})$ and randomizes it using α_2 to obtain $e' = (g^m \cdot y^{\alpha_1+\alpha_2}, g^{\alpha_1+\alpha_2})$ and publishes the randomization, they can prove that $e' \cdot e^{-1} = (y^{\alpha_2}, g^{\alpha_2})$ and use PEDL to prove the two components share a discrete log. Defining e_m as the message component of the encryption e and e_k as the ephemeral key component, we can define PER as follows:

$$\mathsf{PER}(e_0, e_1) = \mathsf{PEDL}(e'_m \cdot {e_m}^{-1}, e'_k \cdot {e_k}^{-1})$$

3.4.3. Zero Knowledge Polynomial Threshold

Zero Knowledge AND and OR allow a prover to prove n of n statements and 1 of n statements respectively, but to prove a more general k of n statements, a more complicated scheme is required. To do this, we utilize a scheme, similar to Cramer [12], that utilizes polynomials mod a prime to handle the challenges, like Shamir's Secret Sharing [13]. If the Prover needs to prove k of n statements, then the Prover needs to be able to simulate the other $n-k$ proofs. This requires the Prover to be able to generate these challenges beforehand without being able to effect the other k challenges. To accomplish this, the Prover chooses the challenges for the $n-k$ challenges and set them as outputs to the associated values of the polynomial, e.g. if the 5th proof is being simulated, then $f(5)$ is defined here. Then, after the initial communications, the Verifier sends c, which is a random number within $\mathbb{Z}_p$. $f(0)$ is set equal to c. The Prover then solves for the coefficients of the polynomial and uses the polynomial f for the true proof challenges. The coefficients are sent to the Verifier, who then uses them to calculate the challenges, which he then uses to verify the component proofs. There is only one order $n-k$ polynomial that can cover the Verifier's challenge and the Prover's chosen values.

ZK Polynomial Threshold Protocol

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

1) For each j such that the Prover does not intend to prove S_j: The Prover uses σ_j to pregenerate the initial message a_j, the simulated sub-challenge c_j, and the response z_j.
2) For each i such that the Prover intends 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) The Prover define an $n-k$ order polynomial f. Define $f(0) := c$.
6) For each simulated sub-challenge c_j, define $f(j) := c_j$.
7) Solve for the coefficients of the polynomial f.
8) For each i where S_i is a true proof, the Prover defines $c_i := f(i)$ and uses c_i as a challenge with the associated π_i to calculate z_i.
9) The Prover sends the list of responses and the coefficients of the polynomial f to the Verifier.
10) The Verifier accepts if all of the following are true:
    a) $f(0)$ is equal to c.
    b) For each protocol π_i: $(a_i, c_i := f(i), z_i)$ is a valid transcript for π_i, given the protocol's public inputs.

ZK Polynomial Threshold Protocol Simulator

1) The Prover* pregenerates the Verifier's challenge c
2) The Prover* defines the coefficients of a $n-k$ order polynomial f with random coefficients, with c as the constant.
3) For each protocol π_i, the Prover* uses the associated simulator σ_i with the challenge $f(i)$ to generate a_i and z_i
4) The Prover* and Verifier run the protocol from protocol step 3, skipping any step with true proofs.

Protocol 1: ZK Polynomial Threshold Protocol

4. Solution Overview

Our solution to this problem is a twofold scheme based on zero-knowledge proofs and perfectly hiding commitments.

1) ***Multisig Proof of Assets.*** This protocol enables each party to create a verified commitment of the sum of their accounts. To accomplish this, they create commitments related to each account in the blockchain or in an anonymity set. These commitments are individually verified, and they are added up to one commitment representing the sum of the owned accounts.
2) ***Zero Knowledge Comparison.*** This protocol enables the two parties to jointly compare their total asset commitments. They achieve this by selecting a table to which each party proves their data is consistent.

5. Multisig Proof of Assets

For the purposes of this protocol, the parties will agree on an anonymity set to limit the execution time of the protocol. Each account in the set must have the public key or public keys revealed on the blockchain, as the protocol can not prove ownership of the hashed Bitcoin address unless an efficient method to prove knowledge of the private key associated with the preimage of the hash is designed. Ideally, for privacy, the anonymity set would be the set of all accounts with anything in them, but this would be prohibitively expensive. The structure of this anonymity set can be defined outside this protocol.

To accomplish a Proof of Assets within an appropriate anonymity set, a commitment is created for each individual account within the anonymity set, where the commitment hides the amount of bitcoin associated with the account if the account is owned and the commitment hides 0 if not. To verify these commitments, this paper applies Zero Knowledge OR and Zero Knowledge AND to prove the following statement is true: "(Either the amount hidden in this commitment is equal to the balance of the account AND I can prove ownership of the account) OR the amount hidden in this commitment is equal to 0."

To this end, we define the following terms: For the ith account of the anonymity list, we define B_i to be the balance of the account. We also define $\hat{B}_i$. If account is claimed by a party, requiring the public key or keys y_i are known, that party's $\hat{B}_i = B_i$. If the account is not claimed, then $\hat{B}_i = 0$. We use the term *claimed* because the account could be owned by a party, but it could be omitted at will. Each party now calculates a commitment $C_{\hat{B}_i} = g^{\hat{B}_i} h^{\gamma_i}$ and performs the Account Verification Proof:

$$\text{OR (AND (PoO, CVP(}C_{\hat{B}_i}, \hat{B}_i\text{)) , CVP(}C_{\hat{B}_i}\text{,0))}$$

, where PoO is a proof od ownership, and CVP is a proof that a commitment hides a specific value using Schnorr's Protocol. The Proof of Ownership method required will depend on the type of the account. A single signature account will require a simple Schnorr and a multisig account would require a more complicated proof.

Using Protocol 2, the values are proven to be correct, which means that if $\mathcal{C}_{\hat{B}_i}$ 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:

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

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

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

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

Proof of Ownership (Protocol 3). In Bitcoin, there are multiple types of accounts that have different ownership conditions. To be able to incorporate more of the blockchain, more types of Proof of Ownership are required. We provide Proofs of Ownership for single key accounts and multisig accounts. These Proofs of Ownership assume that the public keys associated with the account in question are on the blockchain. Proof of Ownership on single key accounts is simply a Schnorr. m-of-n multisig accounts are more complicated. They can be split in to three cases: $m = 1$, $m = n$, and the inner cases, $m \neq 1$ and $m \neq n$. In all of these cases, a multisig account i has n public keys, $y_{i,j}$, each of which has a private key, $x_{i,j}$.

In the $m = 1$ case, the Prover needs only one of the private keys $x_{i,j}$. To accomplish this, we utilize Zero Knowledge OR to prove that at least one of the keys is know.

In the $m = n$ case, the Prover needs all of the private keys $x_{i,j}$. To accomplish this, we utilize Zero Knowledge AND to prove every key is known. In all other cases, when $m \neq 1$ and $m \neq n$, the Prover needs m keys.

6. Zero Knowledge Comparison Protocol

To compare the two total asset commitments, the two parties jointly execute the Zero Knowledge Comparison Protocol (Protocol 4). To do this, we use elliptic curve Elgamal due to its additive homomorphic properties and similarity to Pedersen commitments, using a distributed public key.

The conceptual model for the comparison is shown in Figure 1. This approach uses three smaller tables within a larger mix and match table [14]. Mix and match tables work by taking encrypted tables of encrypted values, randomizing the ciphertexts, and rearranging the rows. In Figure 1a, we show an unshuffled table. In Figure 1b, we show a shuffle of the other table, where each row has an equal random chance to appear in every other position in the table.

We start on the most significant bit with the current state of the number read to this point as "equal," so the parties use the $[\![=]\!]$ table. Then, the two parties P_1 and P_2 take the two ciphertexts corresponding to their bit and randomizes them. For example, if P_1's bit is 0, then they will randomize the the first and second ciphertexts from the table, whereas P_2 would select the first and third row. They then prove that their bit and ciphertexts match the table using the Bit Table Verification Proof, discussed later. These ciphertexts are homomorphically added together and called the "feedback," which represents the comparison up to this point. After this, the table is mixed using a mix network. The parties now use PET from Mix and Match [14] to determine which of the rows matches the feedback. We then repeat until all the bits have been used. At this point, the two parties decrypt the final feedback and determine the result. Since the result is one of three known values, it is simple to determine what the message is.

6.1. Bit Table Verification Proof

The Zero Knowledge Proof for each possibility will be structurally the same. The general ZKP for the party's

Multisig Proof of Assets Protocol

Input:	Public:	Generators g and h, The list of accounts L, where each account is a pair (y_i, B_i), where y_i is a ECDSA public key and B_i is the balance associated with the address.
	Private (Prover):	A set of owned accounts
Output:	Public:	A verified commitment of the sum of the Prover's claimed accounts $\mathcal{C}_{\text{sum}}$ and a list of the Prover's account commitment

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 $C_{\hat{B}_i} := g^{B_i}h^{\gamma_i}$. Else, the Prover creates a commitment $C_{\hat{B}_i} := g^0 h^{\gamma_i}$
   b) The Verifier requires the Prover to prove the following statement: "Either the commitment $C_{\hat{B}_i}$ hides B_i AND the Prover owns the account OR the commitment $C_{\hat{B}_i}$ hides 0." To accomplish this, the Prover and Verifier execute zk-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 zk-AND on the following two protocols:
        - CVP to prove $\mathcal{C}_{\hat{B}_i}$ hides B_i.
        - Proof of Ownership of the account
      - CVP 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:

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

Protocol 2: Multisig Proof of Assets

Proof of Ownership Protocol

Input:	Public:	Generators g and h, the list of n public keys y_j, and the threshold of required keys m
	Private:	A set of owned m or more private keys $x_j \in \mathbb{Z}_q$ such that $g^{x_j} = y_j$ for account j.
Output:	Public:	Verification that Prover holds m of n keys

- **Case#1:** $n = 1$: Use Schnorr's Protocol to prove ownership of a single key account
- **Case#2:** $m = 1$: Proof ownership of one of the keys:
  OR (Schnorr($y_{i,1}$), Schnorr($y_{i,2}$), ... Schnorr($y_{i,n}$))
- **Case#3:** $m = n$: Prove of ownership of all of the keys:
  AND (Schnorr($y_{i,1}$), Schnorr($y_{i,2}$), ... Schnorr($y_{i,n}$))
- **Case#4:** $m \neq 1$ and $m \neq n$: Ownership of m of n keys:
  PT$_m$ (Schnorr($y_{i,1}$), Schnorr($y_{i,2}$), ... Schnorr($y_{i,n}$))

Protocol 3: Proof of Ownership

encryption corresponding to the bit commitment C_{b_i} hiding a specific bit b is as follows:

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

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 $b = 1$):

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

(a) Pre-shuffled table

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

(b) Post-shuffled table

Figure 1: The conceptual design of our mix and match table.

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

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.

7. Security Analysis

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.

Zero Knowledge Comparison Protocol

Input: Public: g, Distributed Public Key h, bit commitments representing the parties' numbers n_1 and n_2
Private: P_i's number n_i, and all the details of the bit commitments for their bits.

Output: Public: Which of the two numbers is larger.

1) P_1 and P_2 create a distributed public key, y.
   a) Both parties compute a public key $y_i = g^{k_i}$, where i is their party number.
   b) Each party commits to their public key $C_{y_i} = g^{k_i} h^{\alpha_i}$ and sends it to the other party.
   c) Each party reveals their α_i and calculates the other party's $y_i = C_{y_i} \cdot h^{-\alpha_i}$.
   d) Each party executes Schnorr's Protocol to prove knowledge of their respective y_i.
   e) Each party calculates $y = y_1 \cdot y_2 = g^{k_1+k_2}$.
2) Initialize the Mix and Match table, unshuffled, and select the *equals* table.
3) For each commitment in the list of commitments, starting with most significant:
   a) Take the table from the selected row.
   b) Each party creates 2 ciphertexts 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, randomize the resulting encryption by multiplying in an encryption of 0.
   c) Each party sends the randomized ciphertexts to the other.
   d) Prove that the ciphertexts are valid using the Bit Table Verification Proof:
      - 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.
   e) Multiply all the ciphertexts (P_1's and P_2's) together to homomorphically obtain the feedback.
   f) Shuffle the table using the method from Mix and Match.
   g) 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.

Protocol 4: Zero Knowledge Comparison

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.
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.
4) **Sound Comparison**. Assuming the concealed sum is accurate, the probability of a dishonest party successfully cheating in the comparison is extremely low.

We will now prove the following theorem:

Theorem 7.1. Variance *is a privacy-preserving bitcoin asset comparison protocol.*

To prove this theorem, we will focus on Correctness and Privacy (due to limited space).

Lemma 1. Variance *is correct.*

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

7.1. Multisig 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 to continue to the next step with the appropriate committed sums.

7.2. Zero Knowledge Comparison

First, we assume that the method to convert the commitment to bit commitments and prove that the bit commitments are valid is correct. The Mix and Match protocol, including PET, is complete, as proven by Jakobsson and Juels [14]. Next, the parties randomize 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 show the feedback will be correct in every case.

- On the Equals ($[\![0]\!]$) table:
  1) 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 matches the Greater Than table. This is the expected result.
  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 greater than P_2's bit, so the feedback matches the Less Than table. This is the expected result.
  3) 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 matches the Equals table. This is the expected result.
  4) 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 matches the Equals table. This is the expected result.
- On the Less Than ($[\![-1]\!]$) table, if any group of 4 encryptions, which are all $[\![-\frac{1}{4}]\!]$, are randomized 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 $[\![\frac{1}{4}]\!]$, are randomized and homomorphically added, the result will be $[\![1]\!]$, which leads back to the Greater Than table after the shuffle, as intended.

This process repeats for each bit of the comparison, which results in the plaintext of the feedback being the result of the comparison. 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 decryption on the last feedback, which returns the relation between P_1 and P_2. □

Lemma 2. Variance *maintains privacy.*

Proof by Simulation. 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 ciphertexts are simply random numbers or points, and thus we will refer to them as *pseudo*, e.g. pseudo commitment. We will first construct a simulator for each of the major component protocols of Variance, then we construct an overarching simulator for the entire protocol.

First, we define the ideal model in Protocol 5. The ideal model defines how the protocol would work if there existed a trusted third party (TTP). The Variance simulator has access to the TTP and can query what the output of the protocol should be. Once the ideal model and the behavior of the TTP is defined, we define a simulator $\mathcal{S}$ (Protocol 6) for Variance that is capable of simulating the view of the other party. $\mathcal{S}$ extracts the necessary secrets from the adversary $\mathcal{A}$ and uses them with the TTP to create a transcript that is computationally indistinguishable from a proper execution of the protocol.

Variance Ideal Model

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

Protocol 5: Variance Ideal Model

8. Complexity Analysis

Theorem 8.1. *Given a list of l bitcoin m-of-n multisig accounts, complexity of* Variance*'s Proof of Assets in the worst case is $\mathcal{O}(l \cdot (s^3 + n^2))$, where $s = n - m$.*

It is important to note that n and m, and consequently s, are all relatively small values, so the factor of l is the most significant part in the larger proof of assets.

To prove this complexity, we will depend on the following lemmas (we will only prove the first one due to limited space):

Lemma 3. *Given a list of l bitcoin accounts, assuming a complexity of x for the Account Verification Proof, the complexity of Proof of Assets in the worst case is $\mathcal{O}(lx)$, where $s = n - m$.*

Proof: Every account requires one run of the account verification 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 $\mathcal{O}(1)$. Each account also requires an execution of the Account Verification Proof, which has a defined complexity of $O(x)$. Thus with respect to the number of accounts (l), the computational complexity of the Proof of Assets is $\mathcal{O}(lx)$. This analysis is also supported by the experiments, which show a linear growth as the number of accounts increase. □

Lemma 4. *Given an 1-of-n bitcoin account, the complexity of the Account Verification Proof in the worst case per account is $\mathcal{O}(n)$.*

Lemma 5. *Given an n-of-n bitcoin account, the overall complexity in the worst case per account is $\mathcal{O}(n)$.*

Lemma 6. *Given an m-of-n where $m \neq 1$ and $m \neq n$ bitcoin account, the complexity of proving ownership of a multisig accounts is $\mathcal{O}(s^3 + ns)$, where $s = n - m$.*

Theorem 8.2. *The zero knowledge comparison protocol has a complexity of $\mathcal{O}(b)$, where b is the number of bits in the comparison.*

Proof: This portion of the protocol consists of two loops with the number of bits in the comparison (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 $\mathcal{O}(1)$. That means the loop as a whole is $\mathcal{O}(b)$. The second loop consists of 2 encryptions, 2 Elgamal randomizations, a large zero knowledge proof, and a mix and match iteration, which includes a shuffle and a PET. According to Jakkobson and Juels [14], 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 $\mathcal{O}(1)$, which puts the second loop at $\mathcal{O}(b)$. □

9. Experimental Evaluation

9.1. Experimental Setup

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 ZKPProver interface includes a parallel-prove, where two parties act as both pr over 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, each party uses a single threaded program to better establish a benchmark. As a rule, inputs for the Zero Knowledge Proofs were pulled in to RAM 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 are 255 bits.

Variance Simulator

Input:	Public:	The set of accounts, where each account is a pair (public key, balance)
Input:	Private ($\mathcal{A}$):	The private keys associated with the owned accounts.
Output:	Public:	$\mathcal{A}$'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:		$\mathcal{A}$'s list of used private keys and the message-key pairs for all of $\mathcal{A}$'s account commitments.

Multisig Proof of Assets

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.
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.

Zero Knowledge Comparison

4) $\mathcal{S}$ gives $\mathcal{A}$'s account private keys to the TTP obtain the result of the comparison m.
5) $\mathcal{A}$ and $\mathcal{S}$ create a distributed public key.
   a) $\mathcal{A}$ generates $k_{\mathcal{A}}$ and calculates $y_{\mathcal{A}} = g^{k_{\mathcal{A}}}$. $\mathcal{S}$ chooses a random point $y_{\mathcal{S}}$.
   b) $\mathcal{A}$ creates a commitment $C_{y_{\mathcal{A}}} = g^{k_{\mathcal{A}}} \cdot h^{\alpha_{\mathcal{A}}}$. $\mathcal{S}$ creates a commitment $C_{y_{\mathcal{S}}} = g^{k_{\mathcal{S}}} \cdot h^{\alpha_{\mathcal{S}}}$.
   c) $\mathcal{A}$ and $\mathcal{S}$ exchange commitments. If $\mathcal{A}$ does not send an elliptic curve point on the curve or sends other data, $\mathcal{S}$ aborts.
   d) After both parties have sent their commitments, $\mathcal{A}$ and $\mathcal{S}$ exchange α values. If $\mathcal{A}$ does not send an integer or sends other data at this point, $\mathcal{S}$ aborts.
   e) $\mathcal{A}$ and $\mathcal{S}$ use Schnorr's protocol to prove knowledge of their y_i.
      - $\mathcal{S}$ uses Schnorr's Protocol's extractor to extract $k_{\mathcal{A}}$ from $\mathcal{A}$. If the extractor aborts, $\mathcal{S}$ aborts.
      - $\mathcal{S}$ simulates Schnorr's Protocol to simulate knowledge of an unknown $k_{\mathcal{S}}$. If the simulator aborts, $\mathcal{S}$ aborts.
   f) $\mathcal{A}$ and $\mathcal{S}$ calculate $y = y_{\mathcal{A}} \cdot y_{\mathcal{S}}$
6) Build the initial table (unshuffled). Select the "Equals" table
7) 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 bit and the randomization exponents and continues. Else, $\mathcal{S}$ aborts.
   d) Homomorphically combine all four pairs as exponential Elgamal encryptions, called feedback.
   e) $\mathcal{S}$ runs 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) $\mathcal{S}$ simulates PET with the feedback to select a single random row. The table in that row is the new selected table.
8) $\mathcal{S}$ takes the final feedback encryption $f = (e_m, e_k)$ and calculates $f_{\mathcal{S}} = (g^m \cdot {e_k}^{k_{\mathcal{A}}}, e_k)$, and then uses $f_{\mathcal{S}}$ to simulate distributed decryption with $\mathcal{A}$.

Protocol 6: Variance Simulator

We implemented our protocol in Java 1.8. Bouncy Castle, a standard cryptographic library for Java is used to handle the elliptic curve operations. We performed our tests on a PC with i7-6700 CPU @ 3.4GHZ and 32GB RAM with a 120 GB SATA SSD. The full source code can be accessed from here[2].

2. Variance Source Code: https://drive.google.com/drive/folders/16FqTTseY6x4lzLnTpZSdipkuSn4f8_tX?usp=sharing

9.2. Single Key Proof of Assets

Since the first part of our protocol solves a problem that encompasses Provisions's Proof of Assets [3], we implemented Provisions's Proof of Assets protocol to compare performances. All experiments are done assuming single secp256k1 public keys, though Variance can handle other key types.

Theoretical Comparison. We look at three important and potentially expensive components: exponentiation, commu-

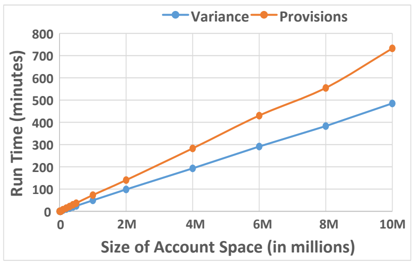

Figure 2: Comparative evaluation of run-time between Variance and Provisions [3].

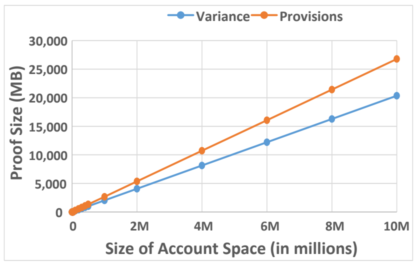

Figure 3: Comparative evaluation of proof size between Variance and Provisions [3].

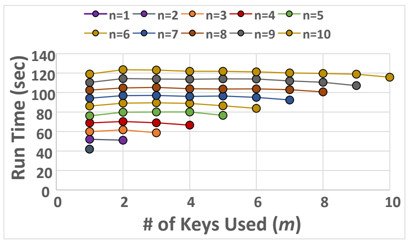

Figure 4: Run Time of *naive method* on $10,000$ accounts for up to $n = 10$.

nications, and random numbers generated. We conduct 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 elliptic curve point in to an approximately 256 bit representation. We count random numbers if they generate up to 256 bits. The results of this comparison is shown in Table 1.

Table 1: Theoretical comparison between with Provision

	Normal Exps	Sim. Exp	Comms	RNGs
Variance	15	16	13	8
Provisions [3]	27	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.

9.3. Experimental Results

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

To properly compare the results of Variance with the results from Provisions, we ensured that they were optimized using the same methods and were operating on the same system by implementing both protocols using the same scheme. Figure 2 illustrates an account space of up to 10,000,000 to show the linear trend with respect to the number of accounts l. This is a non-threaded edition of the protocol, which is almost *embarrassingly parallel*, so the time could be reduced significantly with parallelism. We will now compare the size of the proofs by measuring the Verifier's transcript. The data is charted on Figure 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 compared to Provisions's.

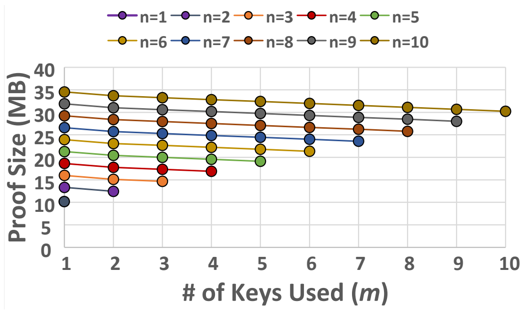

Figure 5: Run Time of *key count method* on 10000 accounts for each combination of m-of-n up to $n = 10$.

9.3.1. Multisig Proof of Assets

Variance is capable of handling multisig accounts as well as single key accounts. We created two major methods and two boundary methods and will now compare their complexity and performance. Multisig accounts are generally referred to as n-of-m. In this section, n refers to the total number of keys on an individual account and is unrelated to the number of accounts, as these proofs apply to one account at a time.

Experimental Results

We analyze our implementation of Variance from 1-of-1 to 10-of-10 to observe scalability on number of keys. Since the multisig Proof of Assets scales linearly with the number of accounts, we ran all experiments with a constant 10,000 accounts on our system. There are a few details of our implementation and experiments that are of note. We ran both implementations on identical anonymity sets with identical sets of owned keys. Though the experiments were run on a single machine, we did take in to account that it is generally bad practice to keep plaintext or decryptable private keys of all the keys of a multisig account on a single computer. To this end, our implementation is set up so that your main process does not have to hold all of the keys as long as they know a friend who can run the proof for them. Though the full communication costs are not taken in to account, as they are highly variable, the additional costs of the SSL encryptions are included, as well as any costs of coordinating the data from the friends internally.

For all experiments, cases where $m = 1$ and $m = n$

use the appropriate boundary case. The runtime of the Key Threshold method up to 10-of-10, as shown in Figure 4. The end points are faster than the middle points, as they use the more efficient boundary cases. The middle points, which use the Key Threshold Method, have a clear trend decreasing as m increases, as expected from the complexity analysis. However, the n values are small, the s^3 trend, so it is not prominently reflected in this graph. The transcript sizes shown in Figure 5 do not reflect the run times in the same way, as transcript sizes scale linearly as m decreases.

10. Concluding Remarks

In this paper, we propose 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 modular design used 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 building block for constructing practical protocols. The experiments showed our Proof of Assets was about 1.5 times more efficient in runtime than the less flexible design from Provisions. This protocol can be applied to real world problems in Bitcoin, such as auctions, credit checks, and proving solvency for exchanges.

References

[1] A. C. Yao, "Protocols for secure computations," in *IEEE FOCS*, 1982.

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

[3] G. G. Dagher, B. Bünz, J. Bonneau, J. Clark, and D. Boneh, "Provisions: Privacy-preserving proofs of solvency for bitcoin exchanges," in *Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '15, 2015, pp. 720–731.

[4] F. Reid and M. Harrigan, *An Analysis of Anonymity in the Bitcoin System*, 2013, pp. 197–223.

[5] N. Narula, W. Vasquez, and M. Virza, "zkledger: Privacy-preserving auditing for distributed ledgers," in *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, 2018, pp. 65–80.

[6] A. Dutta and S. Vijayakumaran, "Revelio: A mimblewimble proof of reserves protocol," in *2019 Crypto Valley Conference on Blockchain Technology (CVCBT)*, 2019, pp. 7–11.

[7] M. Mohan, M. K. K. Devi, and V. J. Prakash, "Confidential and efficient asset proof for bitcoin exchanges," *Sādhanā*, vol. 43, no. 8, p. 126, Jun 2018.

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

[9] N. Koblitz, "Elliptic curve cryptosystems," *Mathematics of computation*, vol. 48, no. 177, pp. 203–209, 1987.

[10] T. P. Pedersen, "Non-interactive and information-theoretic secure verifiable secret sharing," in *CRYPTO*, 1992.

[11] C. Hazay and Y. Lindell, *Efficient Secure Two-Party Protocols*. Springer, 2010.

[12] R. Cramer, I. Damgård, and B. Schoenmakers, "Proofs of partial knowledge and simplified design of witness hiding protocols," in *CRYPTO*, 1994.

[13] A. Shamir, "How to share a secret," *Commun. ACM*, vol. 22, no. 11, pp. 612–613, Nov. 1979. [Online]. Available: http://doi.acm.org/10.1145/359168.359176

[14] M. Jakobsson and A. Juels, "Mix and match: Secure function evaluation via ciphertexts," in *ASIACRYPT*, 2000.

Appendix

10.1. Zero Knowledge Proofs

Variance requires that base sigma protocols are honest verifier zero knowledge proofs (HVZKP). To be HVZKP, it must be *complete*, *specially sound*, and honest verifier zero knowledge. Completeness is whether an honest run of the protocol will be successful, which is apparent and will be omitted and will be omitted. Special soundness is proven using extractors, which we provide in Protocols 7 and 8.

Extractor for Proof of Equal Discrete Log

1) The Prover and Verifier* run PEDL Protocol, resulting in the transcript $((a_g, a_h), c_1, z_1)$.
2) The Verifier* rewinds the Prover 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 either transcript fails, the extractor aborts. Else, continue.
5) $\frac{z_1 - z_2}{c_1 - c_2} = \frac{(r + c_1 \cdot x) - (r + c_2 \cdot x)}{c_1 - c_2} = \frac{(c_1 - c_2) \cdot x}{c_1 - c_2} = x$.

Protocol 7: Extractor for Proof of Equal Discrete Log

ZK Polynomial Threshold Protocol Extractor

1) The Prover and Verifier* run Protocol 1 (ZK Polynomial Threshold Protocol), getting the Verifier*'s first challenge (c_1) as well as a list of transcripts (T).
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').
4) The Verifier* verifies both lists of transcripts are valid, as well as the polynomials f and f'. If any of the transcripts are invalid, the extractor fails and aborts.
5) For each protocol π_i where $f(i) = f'(i)$, take the transcripts T_i and T'_i and utilize the associated extractor to extract the associated secrets.

Protocol 8: ZK Polynomial Threshold Protocol Extractor