Privacy Focused Proof of Reserves Protocols for Cryptocurrency Exchanges

A thesis submitted in partial fulfillment of
the requirements for the degree of

Doctor of Philosophy

by

Arijit Dutta
(Roll No. 154070004)

Under the guidance of
Prof. Saravanan Vijayakumaran

Department of Electrical Engineering
INDIAN INSTITUTE OF TECHNOLOGY BOMBAY
July 2021

Dissertation Approval

The thesis entitled

Privacy Focused Proof of Reserves Protocols for Cryptocurrency Exchanges

by

Arijit Dutta
(Roll No. 154070004)

is approved for the degree of

Doctor of Philosophy

Prof. Sushmita Ruj
(Examiner)

Prof. Nikhil Karamchandani
(Examiner)

Prof. Saravanan Vijayakumaran
(Supervisor)

Prof. K. S. Mallikarjuna Rao
(Chairman)

Date: 22nd July, 2021

Place: _____

DECLARATION

I declare that this written submission represents my ideas in my own words and where others' ideas or words have been included, I have adequately cited and referenced the original sources. I also declare that I have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in my submission. I understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

Arijit Dutta
Roll Number - 154070004
Date - 22.07.2021

Acknowledgement

I would like to express my immense gratitude to my supervisor Prof. Saravanan Vijayakumaran, who has been an ideal teacher and guide since the beginning of my journey on the path of a Ph.D. student. He gave me an opportunity to work with him, motivated to put more efforts, and supported when things did not work out as planned. He introduced and motivated me to pursue research in this exciting area of applied cryptography for the blockchain technology. Owing to his impeccable leadership qualities, I could work and grow in a healthy, resourceful, and collaborative environment.

I would like to thank Prof. Bikash Kumar Dey, Prof. Nikhil Karamchandani, and Prof. Sachin Patkar who were in the research progress committee. They always motivated me to improve my research and clarified my doubts whenever approached. I am also indebted to every other faculty members of IIT Bombay who taught me various subjects which helped me a lot during research. Their wisdom and philosophy showed me the path during hard times.

I would like to thank my collaborators Suyash Bagad and Arnab Jana. Their insights have enhanced the quality of the research. I shall always cherish the long hours in the lab we spent discussing various aspects of the projects.

I would like to acknowledge the help and support I have received from the Bharti Centre for Communication lab. It provided me a suitable place to work, resources, and funds for research curriculum. I am also indebted to the various staffs and seniors of the lab and the department who helped me in every way they could whenever approached.

Finally, I am grateful to my parents (Shree Surajit Dutta and Smt. Bani Dutta), grandfather (Shree Sushil Mukherjee), and wife (Smt. Sweta Das) for their support, blessings, and well wishes. Without the support of my family, I could not have reached to the place I am today and achieved whatever I could.

Abstract

Cryptocurrency exchanges help to distribute the ownership of cryptocurrencies from the miners to other cryptocurrency users. They buy cryptocurrencies from different miners and provide their customers with various services, namely, possession of cryptocurrencies, safe-keeping the private keys in custodial wallets, and trading of cryptocurrencies. However, there is growing customer concern and distrust towards this popular business model. This is mainly because of the loss of assets by the exchanges due to fraudulent activities and the inability of the exchanges to meet the liabilities towards their customers at some particular instant of time. To regain the trust of the customers, it is proposed that the exchanges should publish periodic proofs that they own more amount of cryptocurrencies than they have sold to their customers i.e. they are solvent. It is desirable that such proof of solvency techniques (a) do not violate the privacy of the exchange, (b) are based on standard cryptographic assumptions, (c) should be publicly verifiable, and (d) need no trusted setup or involvement of a third party.

Provisions [1] is the first proof of solvency scheme proposed for Bitcoin which meets all the above criteria. To prove solvency, the scheme generates an encryption of the reserves amount and obfuscates the exchange-owned addresses in a larger anonymity set. Then the scheme gives a proof of reserves (PoR) without violating the privacy of the exchange. A PoR protocol is an integral part of a proof of solvency protocol. Unlike the remaining steps, the PoR protocol in Provisions is specific to Bitcoin and does not work for other cryptocurrencies. We have proposed the first privacy focused PoR protocols for three different cryptocurrency systems, namely, Monero, MimbleWimble, and Quisquis. Our PoR protocols enable proof of solvency schemes satisfying the above mentioned criteria. For Monero, we have proposed another PoR scheme which provides better privacy than our previous proposal. The simulation results show that all the proposed schemes are practical enough to be adopted by the exchanges in practice.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 A Brief Introduction to Cryptocurrency

The design of a decentralized electronic cash system was an open problem for decades. The problem was solved in 2008 when an anonymous programmer called Satoshi Nakamoto proposed Bitcoin [2]. Bitcoin maintains a peer-to-peer network around the globe and enables any two users to transfer bitcoins without depending on any central financial institution. The security of Bitcoin is unaffected by the malicious behavior or failure of a single node. Figure 1.1 depicts a typical cryptocurrency peer-to-peer network consisting of *full nodes*, *miners*, and a public ledger of transactions called the *blockchain*.

Bitcoin is an open source software. Anybody can run the Bitcoin software and become a node in the peer-to-peer network. However, full nodes in the network maintain and store the blockchain. The blockchain is an append-only ledger containing all the transactions which cannot be corrupted by a computationally bounded adversary. Typically in a cryptocurrency system, each user has multiple public-secret key pairs. The public keys of the users are associated with the *outputs* recorded in the blockchain. These outputs contain coins which can be spent by a digital signature signed by the corresponding secret keys. When Alice wants to transfer some coins to Bob, she specifies one of her outputs as the *input* of the transaction and creates a new output using one of Bob's public keys. Bob can spend from this output later by a digital signature signed by his secret key. To spend from the input, Alice produces a digital signature using her secret key. After constructing the transaction in the prescribed format, Alice sends it to one of the full nodes. The full node checks the validity of the transaction by checking (a) the validity of the

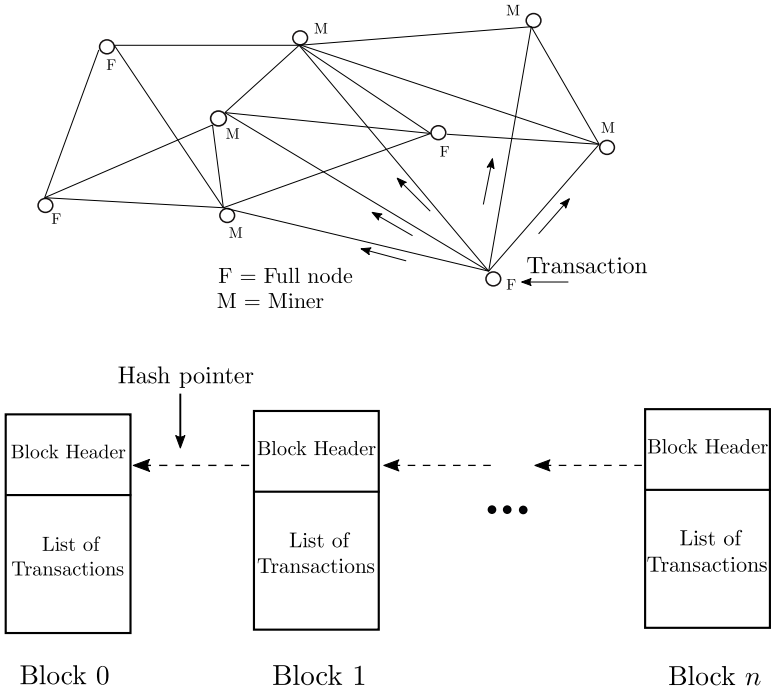

Figure 1.1: Cryptocurrency network and blockchain.

signature, (b) that the input of the transaction contains unspent coins, and (c) that the spent amount is not more than the amount corresponding to the input. If the transaction is valid, the full node broadcasts this transaction to other full nodes and it gets relayed in the entire network.

Miners are some special full nodes which compete with each other to add a new block containing latest transactions to the blockchain. They try to solve a computationally hard problem*. The miner which solves the problem first, gets the chance to add the new block to the blockchain and receives some newly generated coins as a block reward. The successful miner broadcasts the new block across the network and all miners start to compete for the next block. A transaction can only be confirmed when it appears in the blockchain.

Blocks in the blockchain consists of a list of transactions and a block header (Figure 1.1). The block header contains the hash of all transactions of the previous block. One of the major concerns in a digital currency system is *double spending*. A double spending in Bitcoin occurs when a user spends from an already spent output. Suppose an output is spent in a transaction which is recorded in a certain block in the blockchain. In this transaction, the spent output serves as the input. To spend from this output again, an

*Here, we consider only proof-of-work based cryptocurrency technologies.

entity has to replace this transaction in the block with another transaction. The new transaction has the same input but different output than that of the previous transaction. This modification makes all the subsequent blocks invalid as their block headers now have changed. To validate the new transaction, the entity has to produce a chain of blocks longer than the existing one recomputing all the subsequent block headers at once. This is because in a proof of work based cryptocurrency system like Bitcoin, the longest version of the blockchain is agreed to be the real version of the blockchain by the consensus rule. Computing such a chain is hard for a computationally bounded individual adversary as this involves competing with the computation power of the rest of the miners. As long as the majority of the computation power is controlled by the honest miners, this is only possible with a very small probability. The incentive for the miners to be honest lies in the block reward and transaction fees which they continue to get only if the trust of people in the corresponding cryptocurrency system stays alive. More precisely, it is not of a miner's interest to rewrite the block history and execute double spending. Because, double spending reduces the faith in the integrity of the cryptocurrency system, which lowers the price of the coin. As a result, all the earned coins of the miners become worthless. Also the costly computation infrastructure developed by the miners is put to waste.

Privacy focused cryptocurrencies. The above model for a cryptocurrency system (mostly applicable to Bitcoin) suffers from some privacy issues. First of all in Bitcoin, the amount in a transaction appears in the plaintext. Secondly in spite of some efforts[†] to achieve privacy, the real identity of the sender and the receiver of a Bitcoin transaction could be extracted out by several techniques [3–5]. These issues motivated the origin of a new class of privacy focused cryptocurrency systems e.g. Monero [6], ZCash [7], MimbleWimble [8, 9], Quisquis [10]. The goals of these proposals are mainly to provide two privacy features: *anonymity* and *confidentiality*. The anonymity property keeps the sender's and the receiver's identity in a transaction a secret whereas the confidentiality property keeps the amount associated with a transaction a secret. We have considered three privacy focused cryptocurrency systems namely Monero [6], MimbleWimble [8, 9],

[†]In a Bitcoin transaction, the identities of the owners of the inputs or outputs are not revealed. Rather, only the corresponding public keys (P2PK transaction) or the hashes (P2PKH transaction) of the public keys are revealed.

and Quisquis [10]. We have proposed proof of reserves protocols for exchanges possessing these three cryptocurrencies. We discuss these protocols after giving a brief description of a cryptocurrency exchange.

1.2 Cryptocurrency Exchange

When a new cryptocurrency emerges, the miners are the ones who can naturally earn coins of that particular cryptocurrency by adding new blocks to the blockchain. When they add new blocks, the newly generated coins (block reward) get transferred to the public keys owned by them. The other source of their earnings are the transaction fees which are included in every transaction. Later, the miners sell cryptocurrency coins to other parties in return for fiat currency, goods, or services. In this way, a non-miner becomes an owner of that particular cryptocurrency. When Bitcoin became popular and various other cryptocurrencies came into existence, many people intended to possess, use, and invest in cryptocurrencies. To meet the ongoing demand and bridge the gap between the miners and non-miners, a new business platform, namely, the cryptocurrency exchange emerged. A cryptocurrency exchange buys cryptocurrency from the miners and sells it to its customers in exchange of fiat currency. The various services provided by a typical cryptocurrency exchange are listed below.

1. A cryptocurrency exchange enables a non-miner to possess cryptocurrencies. They provide their customer a custodial wallet which displays the amount of coins purchased. A custodial wallet has the following advantages.

    (a) In a typical cryptocurrency system, the knowledge of the secret key implies the ownership of the amount associated with the public key. If this secret key is lost or forgotten, then the entire amount associated with the public key is lost forever. A cryptocurrency secret key is typically a 256 bit number. Though there exists several techniques which convert a secret key into a shorter alphanumeric string, it is still a difficult task for a layman to store multiple secret keys securely. Instead, using a custodial wallet is much convenient. To use her coins, a customer only needs to login to the website of the exchange. In case the customer forgets her password, she can retrieve it back with the help of the customer support and proper authentication. The safety of the

secret keys stored in the custodial wallets is the primary guarantee promised by a cryptocurrency exchange.

(b) After Bitcoin, several other cryptocurrencies with dynamic market values emerged and new cryptocurrencies are still being proposed. As a consequence, the trade of one cryptocurrency with another has become a lucrative investment option. Using the wallet provided by an exchange, a customer can trade between cryptocurrencies. This trade is efficient and faster because the transaction does not need to be recorded in the different underlying blockchains. The exchange possessing several cryptocurrencies can handle such transactions internally. The exchange also provides several charts and trading suggestions to its customers.

In spite of the above advantages, investing in cryptocurrency using a cryptocurrency exchange is not secure. Some of the customers' concerns are listed below.

- There are many examples of loss of customers' funds due to exit scams by the exchange owners, hacking, theft, and internal fraud [11]. According to an anti-money laundering report, an estimated US$1.9 billion worth of cryptocurrencies were reported as stolen from exchanges in the year 2020 [12].

- Another concern is fractional reserves held by the exchanges. It is expected from a cryptocurrency exchange that it should have full reserves i.e. the amount of cryptocurrency it owns should always be more than the amount it has sold to its customers. However, an exchange might falsely claim that it owns more cryptocurrency than it actually holds.

To address the customers' concerns and make cryptocurrency exchanges transparent, it is prescribed that an exchange should publish periodic proofs that it is solvent i.e. it owns more coins than it has sold to its customers. Below, we discuss the several proposals for a proof of solvency scheme.

1.3 Proof of Solvency Protocol

Typically, a proof of solvency protocol consists of three parts.

1. The exchange needs to prove that it owns a certain amount of coins (say a_{res}) in the corresponding blockchain. This proof is often termed as the proof of reserves/assets.

Such proofs are specific to the underlying blockchain technology. They can either be publicly verifiable or verifiable by a designated auditor.

2. The exchange needs to prove that it has sold a specific amount of coins (say a_{liab}) to all its customers. This proof is known as the proof of liabilities. Such proofs in general do not depend on the underlying blockchain technology. Rather the proofs require verification by the customers of the exchange and the auditors.

3. The exchange needs to show $a_{\text{res}} \geq a_{\text{liab}}$ in order to prove that it is solvent.

Decker et al. [13] proposed a protocol for Bitcoin exchanges which only produces a binary output indicating whether the total reserves are more than the total liabilities. This proof of solvency protocol is privacy preserving in the sense that it does not reveal the exchange-owned addresses, a_{res}, or a_{liab}. However, it is based on a trusted platform module. Below we discuss various proposals for each part of a proof of solvency protocol.

Proof of reserves. Historically, proofs of reserves came into existence before proofs of liabilities and proofs of solvency. In 2011, the Mt. Gox cryptocurrency exchange published a transaction on the Bitcoin blockchain transferring 424,242 bitcoins from its wallets to a previously revealed Bitcoin address [14]. This transaction might be considered as the first proof of reserves proving that Mt. Gox indeed possessed a certain amount of bitcoins. In 2019, Blockstream released a tool for Bitcoin exchanges which generates a transaction including all unspent transaction outputs (UTXOs) of an exchange revealing the total reserves amount [15]. It also includes an invalid input to make the transaction invalid. This is to prevent the exchange reserves from being spent. However, these techniques reveal the total reserves amount of the exchange and all the owned Bitcoin addresses. This is crucial business information which an exchange might not want to reveal. To address this issue, Dagher *et al.* [1] proposed a protocol called Provisions for Bitcoin exchanges in the year 2015. Provisions, consisting of a privacy focused proof of reserves protocol followed by a privacy focused proof of liabilities protocol, proves that the exchange is solvent. It was the first scheme which required no trusted setup and was based only on cryptographic assumptions. In the first stage of Provisions, the proof of reserves protocol generates a Pedersen commitment [16] C_{res} to a_{res}. To generate C_{res}, an anonymity set is used which contains all the exchange-owned Bitcoin addresses and some cover addresses. Thus the protocol hides the total reserves amount in a commitment and blends all the exchange-

owned Bitcoin addresses in an anonymity set. The associated zero-knowledge argument of knowledge[‡] proves that only the amounts corresponding to the exchange-owned addresses in the anonymity set are added to generate a commitment to the total reserves i.e. C_{res} is indeed a commitment to the total reserves amount a_{res}. In this way the privacy of the exchange is preserved. However, the proof of reserves protocol in Provisions is specific to the Bitcoin blockchain technology and cannot be used in other cryptocurrencies. We discuss the Provisions' proof of reserves in more detail in Section 2.3.2.

Proof of liabilities. Proof of liabilities is a blockchain independent protocol which can be applied to any cryptocurrency technology. Maxwell proposed a protocol (summarized in [18]) based on Merkle trees which helps an exchange to prove that a_{liab} includes the corresponding amount of a verifying customer. However this protocol reveals the deposit amount of the sibling customer in the tree and a_{liab} to a verifying customer. The proof of liabilities protocol in Provisions improved on it by publishing a commitment to the balance of each customer and a Pedersen commitment to a_{liab}. As mentioned above, in the first stage of Provisions, the proof of reserves protocol generates a Pedersen commitment C_{res} to the total reserves a_{res}. In the next stage, the proof of liabilities generates a Pedersen commitment C_{liab} to the total amount of bitcoins that the exchange has sold to its customers i.e a_{liab}. It also gives range proofs to prove that each customer's amount lies in the set $\{0, 1, 2, \ldots, 2^{51}\}$ where 2^{51} is a bound on the maximum number of satoshis[§] that could exist. This is to show that the exchange is not cheating by adding a negative number while calculating C_{liab}. Any customer can verify that her amount is included in C_{liab} by using some auxiliary information provided by the exchange. To prove solvency in Provisions, another range proof is used to prove that $C_{\text{res}}C_{\text{liab}}^{-1}$ commits[¶] to a non-negative number in the last stage.

As discussed above, the proof of liabilities protocol in Provisions preserves the exchange privacy by not disclosing a_{liab}. However, if a customer fails to check whether her amount is included in a_{liab}, the exchange could possibly omit that amount, effectively reducing its liabilities. Recently, Chalkias et al. [19] proposed a scheme called Distributed Auditing Proofs of Liabilities (DAPOL) which addresses this concern. DAPOL uses a

[‡]We follow the definition of zero-knowledge argument of knowledge as given in [17, Section 2.2].

[§]Smallest unit of bitcoin. 1 bitcoin=10^8 satoshis.

[¶]$C_{\text{res}}C_{\text{liab}}^{-1}$ commits to the amount $a_{\text{res}} - a_{\text{liab}}$ due to the homomorphic property of Pedersen commitment.

deterministic sparse Merkle tree along with range proofs on the commitment to each customer's amount and produces a Pedersen commitment to a_{liab}. The authors proposed to remove the limitation of Provisions' proof of liabilities by using private information retrieval by the customers to view their inclusion proofs. In this way, the exchange cannot have a prior idea about which customer is more likely to check their inclusion proof and hence does not get any advantage if it decides to cheat. Further, they used a key distribution function and a verifiable random function to deterministically shuffle the users in the Merkle tree in every audit.

1.4 Our Contribution

The main contribution of this thesis is to propose proof of reserves protocols which do not reveal sensitive information of an exchange. We summarize the major contributions below.

- For Monero [6], we have proposed MProvisions modifying the proof of reserves protocol in Provisions. Then we propose MProve which performs better than MProvisions.
- Both MProve and MProvisions preserve the privacy of a Monero exchange to some extent. However, the privacy of the exchange as well as the entire Monero network is affected when the exchange spends from a source address used in the reserves proofs in future. We have proposed a privacy enhancing proof of reserves protocol called MProve+ to address this issue.
- For MimbleWimble [8, 9], we have proposed Revelio, another privacy focused PoR protocol. Unlike Bitcoin and Monero, there is no address in a MimbleWimble-based cryptocurrency. The coins are stored in Pedersen commitments which collectively form the UTXO set. The Revelio scheme can be used for MimbleWimble based cryptocurrencies like Grin [20] and Beam [21].
- Lastly, we have considered a recently proposed cryptocurrency scheme called Quisquis [10]. Quisquis is a privacy focused cryptocurrency following account model unlike all other cryptocurrencies discussed so far. For Quisquis, we have proposed Nummatus, another privacy focused proof of reserves protocol.

All the proposed protocols are first such privacy focused proof of reserves protocols for Monero, MimbleWimble, and Quisquis. All of them generate a Pedersen commitment C_{res} to the total reserves amount a_{res}. The Pedersen commitment C_{res} can be used with the Provisions' or DAPOL's proof of liabilities protocol to prove that the exchange is solvent in a privacy preserving manner. The proposed protocols are based on standard cryptographic assumptions and do not require any trusted setup. The simulation results show that they can be used in practice.

1.5 Organization of the Thesis

A brief summary of subsequent chapters is given below.

Chapter 2 starts with a discussion on the basic features of the Monero cryptocurrency. Then we discuss the existing non-private proof of reserves scheme for Monero. Next, we describe the Provisions' proof of reserves scheme and how it can be modified to obtain a privacy focused proof of reserves protocol for Monero. We call this new scheme MProvisions. Later, we propose another protocol, namely, MProve which performs better than MProvisions. We give the performance comparison for both of them. Next, we discuss the security properties and how the protocols behave with the privacy features of Monero. We point out the drawback both the protocols have when the exchange spends from a source address.

In Chapter 3, we address the above drawback using the techniques from Bulletproofs [17] and Omniring [22]. After giving a brief description of both the schemes, we propose the MProve+ scheme which solves the drawback of both MProve and MProvisions schemes. MProve+ gives logarithmic proof size with respect to the size of the anonymity set as compared to the linear proof size in MProve and MProvisions. We give the performance comparison between the MProve+ and MProve schemes. Then we discuss the security properties of the MProve+ scheme and how it behaves with the privacy features of Monero.

In Chapter 4, we have considered Grin, a MimbleWimble-based privacy focused cryptocurrency. We start with a brief overview of Grin. Then we discuss the non-interactive zero-knowledge proof of knowledge (NIZKPoK) signatures proving statements about discrete logarithm relations as summarized by Camenisch and Stadler [23, 24]. These sig-

natures serve as the building block for both Revelio and Nummatus, a proof of reserves scheme discussed in Chapter 5. Then we describe the Revelio scheme for MimbleWimble. Next we discuss the security properties and give the performance analysis.

We start the Chapter 5 with a brief overview of Quisquis, an account based privacy focused cryptocurrency. Then we give the proof generation and verification algorithms for the Nummatus scheme. We conclude the chapter with security and performance analyses.

1.6 Desirable Properties of Proof of Reserves and Challenges

A privacy focused proof of reserves protocol computes an encryption of the total reserves of an exchange and proves that the encryption indeed hides the actual reserves amount using a zero-knowledge proof technique. A privacy focused proof of reserves protocol should possess the following properties.

- *Inflation resistance.* This property roughly implies that it should be infeasible for a computationally bounded exchange to claim that it possess more cryptocurrencies than it actually owns.
- *Collusion resistance.* This property roughly implies that the protocol should prevent two or more exchanges hiding a fractional reserves incident by sharing each other's funds.
- *Privacy.* The protocol should preserve the privacy of the exchange.

Proposing formal definitions of the above properties which are invariant to the underlying cryptocurrency technology is observed to be hard and beyond the scope of the thesis. Some of the major differences between the various cryptocurrency technologies are listed below.

- In Monero, a spent address cannot be identified to be spent by default. Here, compared to the other cryptocurrency technologies we have an additional requirement of showing that the source addresses are not spent without affecting the privacy of the exchange.

- In case of MimbleWimble, we do not have any addresses. Here, coins are stored in *outputs* which are Pedersen commitments.
- Unlike the above two cryptocurrencies, Quisquis follows an account based model instead of the UTXO model.

Because of the above differences, we found it difficult to intertwine the several proposed proof of reserves techniques with some common definitions of the cryptographic properties. Instead, we have discussed this properties in the context of each cryptocurrency technology separately.

Achieving privacy in the ideal sense is also very hard. To be precise, all the proposed proof of reserves protocols reveal that the exchange owns some addresses in the anonymity set. The proof size[‖], generation time, and verification time of the proposed protocols are linear in terms of the anonymity set. Therefore, it is not practical to declare the set of all addresses/outputs/accounts of the corresponding cryptocurrency blockchain as the anonymity set. The different aspects of privacy also vary when the underlying cryptocurrency technology changes.

[‖]The proof size of the MProve+ protocol is logarithmic in terms of the anonymity set size. However, the generation and verification times are linear. As the Monero UTXO set is a very large and monotonically growing set, it is not practical to set the anonymity set as the set of UTXO.

Chapter 2

MProvisions and MProve: Proof of Reserves Protocols for Monero

2.1 Introduction

Monero is a privacy focused cryptocurrency scheme launched in the year 2014 [6, 25]. In a Monero transaction, the scheme hides the identity of the sender, receiver, and also the amount. In 2018, a developer with the moniker Stoffu Noether proposed and implemented a proof of reserves protocol for Monero [26]. It was meant for an individual to reveal her Monero holdings to others. If this protocol is used by a Monero exchange, the privacy of the exchange is violated (as discussed in Section 2.3.1). In this chapter, we propose two novel proof of reserves protocols for Monero, namely, MProvisions and MProve. Both the protocols preserve the privacy of the exchange to some extent. MProvisions is constructed by modifying the proof of reserves scheme for Bitcoin proposed in Provisions [1]. The MProve construction, which outperforms the MProvisions protocol, utilizes ring signatures and linkable ring signatures. We describe them after giving a brief overview of the Monero scheme.

2.2 Overview of Monero

The design of the Monero scheme is based on the CryptoNote protocol [27]. The scheme mainly comprises of three cryptographic primitives. In a Monero transaction, *one-time addresses* are used to preserve the anonymity of the receiver, *linkable ring signatures* are

used to preserve the anonymity of the sender while preventing double spending, and the *ring confidential transaction* scheme is used to preserve the confidentiality of the amount associated with the transaction. We give an overview of these technologies.

2.2.1 Receiver Privacy and the Unlinkability Property of Monero

Monero public keys are points in the prime order subgroup of the Twisted Edwards elliptic curve Ed25519 [25]. Let $\mathbb{G}$ denote this subgroup generated by the base point G. The order q of the subgroup is a 253-bit prime. Monero secret keys are integers in the set $\mathbb{Z}_q = \{0, 1, 2, \ldots, q-1\}$. In this chapter, we will use additive notation for the group operation on the curve. The public key $B \in \mathbb{G}$ corresponding to the secret key $b \in \mathbb{Z}_q$ is given* by

$$B = bG = \underbrace{G + \cdots + G}_{b \text{ times}}.$$

In Monero, every user possesses a public key pair. For example, let $(B_{\text{vk}}, B_{\text{sk}}) \in \mathbb{G}^2$ be the public key pair of Bob. The keys B_{vk} and B_{sk} are called the view public key and spend public key respectively. The corresponding secret keys, $b_{\text{vk}}, b_{\text{sk}} \in \mathbb{Z}_q$ such that $B_{\text{vk}} = b_{\text{vk}}G$ and $B_{\text{sk}} = b_{\text{sk}}G$, are called the secret view key and secret spend key respectively. Bob can share his key pair $(B_{\text{vk}}, B_{\text{sk}})$ with anyone who wants to pay him. Multiple one-time addresses owned by Bob can be created using this public key pair.

Suppose in a Monero transaction *txn*, Alice wants to transfer the coins associated with one of her own one-time addresses to Bob. She creates a one-time address for Bob as follows. First, she chooses a random scalar r from $\mathbb{Z}_q$ (we shall denote this by $r \xleftarrow{\$} \mathbb{Z}_q$ henceforth) and computes the destination one-time address as $P' = H_s(rB_{\text{vk}}, o_{\text{index}})G + B_{\text{sk}}$. Here $H_s : \{0,1\}^* \mapsto \mathbb{Z}_q$ is a hash function which maps its inputs to scalars and o_{index} is the index of the new output in *txn*. Here the comma inside the hash function denotes the bit-wise concatenation $\|$ operation. Alice also computes the group element $R' = rG$ and includes it in *txn*. Subsequently, *txn* containing (P', R') is added to the blockchain.

*In this chapter, we have used additive notation to be consistent with the description of the Monero scheme in the literature [25, 28]. However in Chapter 3, we have used multiplicative notation to be consistent with the Bulletproofs [17] and Omniring [22] papers. This abuse of notation is unavoidable because of the usage of dual notations in the literature.

For every (P, R) in every transaction in the blockchain, Bob computes a group element $P'' = H_s(b_{\text{vk}}R, o_{\text{index}})G + B_{\text{sk}}$. To compute P'', only the knowledge of secret view key b_{vk} is required. For (P', R') in *txn*, P'' will be equal to P' as,

$$P'' = H_s(b_{\text{vk}}R', o_{\text{index}})G + B_{\text{sk}} = H_s(b_{\text{vk}}rG, o_{\text{index}})G + B_{\text{sk}} = H_s(rB_{\text{vk}}, o_{\text{index}})G + B_{\text{sk}} = P'. \tag{2.1}$$

The above equality holds because $rB_{\text{vk}} = b_{\text{vk}}rG = b_{\text{vk}}R'$. This group element is called the *Diffie-Hellman shared secret*. By verifying the equality in equation (2.1), Bob can identify P' as a one-time address which was generated from his public key pair $(B_{\text{vk}}, B_{\text{sk}})$. He calculates the secret key x' of the one-time address P' as $H_s(b_{\text{vk}}R', o_{\text{index}}) + b_{\text{sk}}$ which holds because,

$$P' = H_s(rB_{\text{vk}}, o_{\text{index}})G + B_{\text{sk}} = H_s(b_{\text{vk}}R', o_{\text{index}})G + b_{\text{sk}}G = (H_s(b_{\text{vk}}R', o_{\text{index}}) + b_{\text{sk}})G.$$

So the knowledge of b_{vk} (not b_{sk}) is needed to identify that P' belongs to Bob. In this way, the fact that P' belongs to Bob is hidden. However as discussed in the next section, the knowledge of the secret key x', hence both (b_{vk}, b_{sk}), is needed to spend funds from P'. So, Bob can outsource the task of monitoring the blockchain for fund receipt to a third party by sharing only the secret view key b_{vk}. This third party can identify all the one-time addresses which belong to Bob. However, she cannot spend funds from them without knowing b_{sk}.

The above technique achieves one of the design goals of Monero called *unlinkability* [27]. This property requires that given a one-time address P, a probabilistic polynomial time (PPT) adversary can identify the corresponding public key pair with a probability which is only neglibly better than random guessing.

2.2.2 Sender Privacy and the Untraceability Property of Monero

Let the source one-time address from which Alice pays Bob in *txn* be P and let x be its secret key i.e. $P = xG$. Using a linkable ring signature, Alice hides the fact that P is the source one-time address in *txn*. To prevent double spending, linkable ring signatures proposed in [29] are used in Monero with some modifications [28]. First, Alice constructs a message m by hashing the transaction prefix which consists of all the transaction data except for the signatures. She creates a linkable ring signature σ on message m as follows:

1. She assembles a list of one-time addresses from the Monero blockchain to form the ring $\mathbf{R}(txn) = (P_0, P_1, \ldots, P_{n-1})$ of *txn* such that $P_j = P$ for exactly one $j \in \{0, 1, \ldots, n-1\}$. Here $\{P_i\}_{i \in \{0,1,\ldots,n-1\}, i \neq j}$ are some random one-time addresses taken from the Monero blockchain. They basically serve as cover addresses (a.k.a. decoy addresses or *mixins*) to hide the source of *txn* i.e. $P_j = P$.

2. Let $x_i \in \mathbb{Z}_q$ be the secret key corresponding to P_i, i.e. $P_i = x_i G$. Using the secret key corresponding to P_j, she computes a group element called the *key image* $I \coloneqq x_j H_p(P_j)$ where $H_p : \mathbb{G} \mapsto \mathbb{G}$ is a hash function producing curve points as outputs.

3. She picks α and $s_i, i = 0, 1, \ldots, n-1, i \neq j$, randomly from $\mathbb{Z}_q$. Note that s_j has not been chosen.

4. She computes points $L_j = \alpha G$, $R_j = \alpha H_p(P_j)$, and integer $c_{j+1} = H_s(\mathbf{R}(txn), m, L_j, R_j)$.

5. Increasing j modulo n, she computes points $L_{j+1}, L_{j+2}, \ldots, L_{j-1}, R_{j+1}, R_{j+2}, \ldots, R_{j-1}$ and scalars $c_{j+2}, c_{j+3}, \ldots, c_j$ as

$$\begin{aligned}
L_{j+1} &= s_{j+1} G + c_{j+1} P_{j+1}, \\
R_{j+1} &= s_{j+1} H_p(P_{j+1}) + c_{j+1} I, \\
c_{j+2} &= H_s(\mathbf{R}(txn), m, L_{j+1}, R_{j+1}), \\
&\vdots \\
L_{j-1} &= s_{j-1} G + c_{j-1} P_{j-1}, \\
R_{j-1} &= s_{j-1} H_p(P_{j-1}) + c_{j-1} I, \\
c_j &= H_s(\mathbf{R}(txn), m, L_{j-1}, R_{j-1}).
\end{aligned}$$

6. Finally, she computes $s_j = \alpha - c_j x_j$. As L_j and R_j were computed using α in step 4, this implies that

$$\begin{aligned}
L_j &= \alpha G = (s_j + c_j x_j) G = s_j G + c_j P_j, \\
R_j &= \alpha H_p(P_j) = (s_j + c_j x_j) H_p(P_j) = s_j H_p(P_j) + c_j I.
\end{aligned}$$

7. The linkable ring signature on the message m is given by $\sigma = (I, c_0, s_0, s_1, \ldots, s_{n-1})$.

Alice includes the linkable ring signature σ in *txn* and broadcasts *txn* onto the network for inclusion in the blockchain. Anybody can verify that the signer of σ knows the secret

key of one of the one-time addresses in the ring $\mathbf{R}(txn)$ without knowing which one. The verification of the linkable ring signature proceeds as follows:

1. The message m which was signed is recreated from the transaction prefix.

2. The ring $\mathbf{R}(txn) = (P_0, P_1, \ldots, P_{n-1})$ used to create the linkable ring signature σ is read from the transaction.

3. Using σ, the integers $c_k, k = 1, 2, \ldots, n-1$, are calculated as

$$\begin{aligned} L_0 &= s_0 G + c_0 P_0, \\ R_0 &= s_0 H_p(P_0) + c_0 I, \\ c_1 &= H_s(\mathbf{R}(txn), m, L_0, R_0), \\ &\vdots \\ L_{n-2} &= s_{n-2} G + c_{n-2} P_{n-2}, \\ R_{n-2} &= s_{n-2} H_p(P_{n-2}) + c_{n-2} I, \\ c_{n-1} &= H_s(\mathbf{R}(txn), m, L_{n-2}, R_{n-2}). \end{aligned}$$

4. Finally, c_{n-1} and s_{n-1} are used to calculate c'_0 as

$$\begin{aligned} L_{n-1} &= s_{n-1} G + c_{n-1} P_{n-1}, \\ R_{n-1} &= s_{n-1} H_p(P_{n-1}) + c_{n-1} I, \\ c'_0 &= H_s(\mathbf{R}(txn), m, L_{n-1}, R_{n-1}). \end{aligned}$$

5. The signature σ is accepted if c'_0 equals the c_0 given in σ. Otherwise, it is rejected.

As it is not revealed which address in $\mathbf{R}(txn)$ is spent in txn, Alice might try to spend from P again in some other transaction. The key image I helps to detect this double spending. To do this, the Monero blockchain maintains the set of already appeared key images $\mathcal{I}$. When I appears for the first time in the blockchain, it is included in $\mathcal{I}$. When Alice spends again from P in another Monero transaction, the same I appears in the corresponding linkable ring signature. Then this transaction is rejected by the network after verifying that I is already a member of $\mathcal{I}$.

The signer ambiguity property of the linkable ring signature scheme prevents a PPT adversary from deducing that I is originated from P with a probability non-negligibly

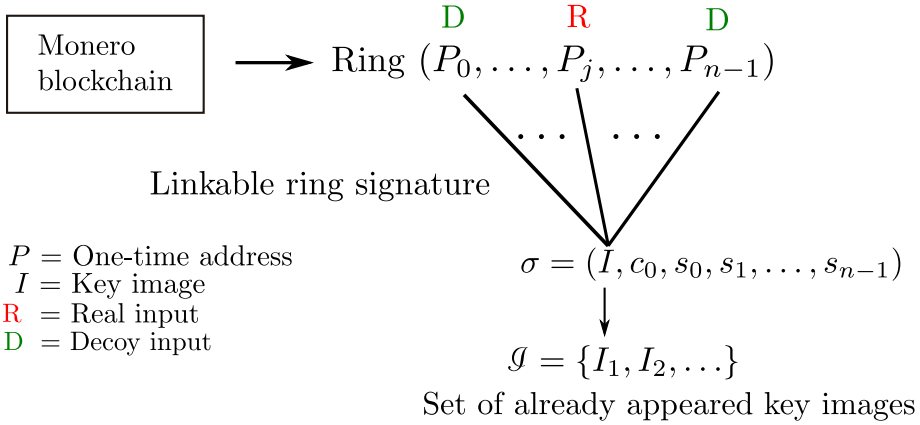

Figure 2.1: Linkable ring signature.

better than that of random guessing. Hence P continues to be a member of the set of unspent one-time addresses, even after being spent in *txn*. The transaction *txn* only reveals that I could have been originated from some one-time address in the ring $\mathbf{R}(txn)$. This information can be represented by a graph having the one-time addresses in the ring and the key image as vertices [30]. This is shown in Figure 2.1. When such graphs from multiple transactions are combined, we will obtain a bipartite graph having the set of one-time addresses as one vertex class and the set of key images as the other vertex class. This bipartite graph will aid us in explaining the effect of MProve/MProvisions and MProve+ (discussed in the next chapter) on the untraceability of Monero.

In this way, linkable ring signatures attain another design goal for Monero, namely *untraceability* while preventing double spending. Roughly speaking, untraceability means that given a transaction ring, no PPT adversary should be able to determine which address in the ring is actually being spent, except with a probability which is negligibly better than that of random guessing [27].

2.2.3 Amount Confidentiality and Public Verification of Monero Transactions

The ring confidential transaction scheme hides the amount in a Monero transaction using Pedersen commitments [16]. Suppose Alice wants to pay a coins[†] of Monero to Bob in *txn*. In *txn*, Alice includes a Pedersen commitment $C = yG + aH$ where $y \xleftarrow{\$} \mathbb{Z}_q$ is a random

[†]Coins in Monero are integers in the range $\{0, 1, 2, \ldots, 2^{64} - 1\}$. This is because the maximum number of piconeros (smallest unit of currency in Monero) that could exist is $2^{64} - 1$.

blinding factor and H is another curve point with unknown discrete logarithm relationship with G. In Monero, every one-time address is associated with a Pedersen commitment hiding the corresponding amount[‡]. One might wonder how miners can validate Monero transactions if the amounts are hidden. This is done as follows.

An interesting feature of Pedersen commitments is that a digital signature can prove that the amount hidden in a Pedersen commitment is zero. This is because a Pedersen commitment hiding the zero amount is of the form $C = zG$ for some $z \in \mathbb{Z}_q$. The person making the commitment can prove that C commits to a zero amount by providing a elliptic curve digital signature algorithm (ECDSA) signature using C as the public key and z as the secret key. Suppose a Pedersen commitment C is of the form $C = zG + aH$ for some $a \neq 0$. For such C, no PPT adversary can compute an ECDSA signature by computing $z' \in \mathbb{Z}_q$ such that $C = z'G = zG + aH$. This is because the discrete logarithm relation of H with respect to G is unknown. This property along with the homomorphic property of the Pedersen commitments is used to provide transaction verification as described below.

Suppose, *txn* has one input and two outputs. Let a_{in} be the input amount, $a^1_{\text{out}}, a^2_{\text{out}}$ be the output amounts, and f be the transaction fees. In particular, Alice wants to pay $a^1_{\text{out}} = a$ amount to Bob from her one-time address P containing amount $a_{\text{in}} > a$. From this amount, Alice pays Bob and the transaction fees f, and creates another one-time address for herself to transfer the remaining amount a^2_{out}. Hence these amounts satisfy the relation,

$$a_{\text{in}} = a^1_{\text{out}} + a^2_{\text{out}} + f.$$

Let $C(y, a)$ denote the commitment $yG + aH$ for $y, a \in \mathbb{Z}_q$. The commitment to the input amount $C(y_{\text{in}}, a_{\text{in}})$ will be recorded in the blockchain with the blinding factor y_{in} known to Alice. Alice will randomly choose blinding factors $y^1_{\text{out}}, y^2_{\text{out}}$ and create the output commitments

$$C(y^1_{\text{out}}, a^1_{\text{out}}) = y^1_{\text{out}}G + a^1_{\text{out}}H,$$
$$C(y^2_{\text{out}}, a^2_{\text{out}}) = y^2_{\text{out}}G + a^2_{\text{out}}H.$$

The transaction will contain $C(y^1_{\text{out}}, a^1_{\text{out}})$, $C(y^2_{\text{out}}, a^2_{\text{out}})$, and the transaction fees f. It will

[‡]This is true for RingCT transactions which were introduced in Monero in 2017 and made mandatory soon after. In this thesis, we have only considered RingCT transactions.

also contain an ECDSA signature verifiable by the public key

$$\begin{aligned} & C(y_{\text{in}}, a_{\text{in}}) - C(y^1_{\text{out}}, a^1_{\text{out}}) - C(y^2_{\text{out}}, a^2_{\text{out}}) - fH \\ &= \left(y_{\text{in}} - y^1_{\text{out}} - y^2_{\text{out}}\right) G + \left(a_{\text{in}} - a^1_{\text{out}} - a^2_{\text{out}} - f\right) H \\ &= zG + 0H = C(z, 0), \end{aligned}$$

where only Alice knows the secret key z. By calculating the public key $C(z, 0)$ and performing ECDSA signature verification, the miners and other full nodes are convinced that the difference between the commitments and the fees term is a commitment to zero.

For simplicity, we denote $a^1_{\text{out}}, y^1_{\text{out}}$, and $C(y^1_{\text{out}}, a^1_{\text{out}})$ by a, y, and C respectively. Bob needs to know a, y to open the commitment C and spend from P' in future. To communicate a and y to Bob, Alice stores the following quantities in *txn*,

$$a' = a \oplus H_K(H_K(rB_{\text{vk}})), \tag{2.2}$$

$$y' = y \oplus H_K(rB_{\text{vk}}), \tag{2.3}$$

where H_K is the Keccak hash function which maps group elements to scalars. Apart from Alice, only entities having access to the Bob's secret view key b_{vk} can recover a and y from a' and y' as follows.

$$a = a' \oplus H_K(H_K(b_{\text{vk}}R')), \tag{2.4}$$

$$y = y' \oplus H_K(b_{\text{vk}}R'). \tag{2.5}$$

The above equations hold again because $rB_{\text{vk}} = b_{\text{vk}}rG = b_{\text{vk}}R'$.

Alice also has to give a proof that each output amount lies in the range $\{0, 1, 2, \ldots, 2^{64} - 1\}$. Presently, Monero uses Bulletproofs [17] for range proofs which is discussed in Chapter 3.

Summary. Bob needs to know the secret key pair $(b_{\text{vk}}, b_{\text{sk}})$ to generate the secret key $x' = H(b_{\text{vk}}R', o_{\text{index}}) + b_{\text{sk}}$ of the one-time address P' (R' can be obtained from the Monero blockchain using b_{vk} as discussed above). To spend from P', Bob can sign a linkable ring signature with x'. Whereas, the knowledge of the Diffie-Hellman shared secret $rB_{\text{vk}} = b_{\text{vk}}R'$ is needed to recover a, y from *txn*. Hence, the ability to generate x' implies the ability to generate a, y. From the above discussion, it is clear that proving knowledge of x such that $P = xG$ is enough to prove the ownership of the amounts corresponding to the one-time address P. We have built our proof of reserves protocols on this principle.

2.3 Existing Proof of Reserves Protocols

We start this section with an overview of the proof of reserves protocol proposed by Stoffu Noether [26]. We discuss how the privacy of a Monero exchange is affected if it uses this protocol. Then we discuss the proof of reserves protocol for Bitcoin which is proposed in the Provisions [1] paper. This description serves as the precursor of the description of the MProvisions scheme in the next section.

2.3.1 Existing Reserves Proof for Monero

As described in Section 2.2.2, a one-time address cannot be identified as spent even if it is spent in a transaction[§]. Hence in a Monero proof of reserves protocol, the exchange needs to prove that its source one-time addresses contributing to the total reserves are not spent already. As mentioned above, the first proof of reserves technique for Monero was added to the official Monero client in 2018 by Stoffu Noether [26]. This tool helps a user to prove that she holds more coins than a target amount. It can be invoked via a command line tool or a remote procedure call to the Monero client. It proceeds as follows.

This tool takes the public key pair of a user and some target amount as inputs. It attempts to find the smallest set of one-time addresses owned by the public key pair whose amount sum exceeds the target amount. Once such a set is identified, the one-time addresses in this set along with their corresponding key images and the Diffie-Hellman shared secrets used in address generation are revealed as part of the proof of reserves. For each address in the set, signatures proving that the Diffie-Hellman shared secret and key image were correctly generated are included in the proof. Upon receiving the proof, an auditor will mark an address as spent if its key image has already appeared in the set of already appeared key images $\mathcal{I}$. As the amount corresponding to an address can be recovered from the Diffie-Hellman shared secret, the auditor can calculate the sum of the amounts in unspent addresses as the reserve amount corresponding to a particular public key pair.

By repeating the proof generation process over all the public key pairs owned by it, an exchange could generate a proof of reserves. While the above technique serves the purpose of proof of reserves, the exchange adopting this technique will eventually leak

[§]Unless it is traced by the methods given in [31–34].

all the one-time addresses it owns and the total reserves amount. This technique also violates the privacy that a Monero exchange enjoys by default in the following ways.

1. As the verifier comes to know about all the one-time addresses generated from the public key pairs of the exchange, the unlinkability property for the exchange-owned one-time addresses is violated.

2. The exchange also has to reveal the key images of all the unspent one-time addresses to prove that they are not spent. When the exchange spends from these one-time addresses in some future Monero transactions, the same key images that have appeared in the reserves proof, appear again. As a result, the one-time addresses corresponding to these key images are revealed. Hence the untraceability property for these Monero transactions is violated.

3. By sharing the Diffie-Hellman shared secrets, the exchange also reveals the amounts corresponding to all of its unspent one-time addresses. Hence the amount confidentiality for these addresses is also violated.

Because of these privacy issues, the above proof of reserves protocol is not suitable for an exchange which is concerned about its privacy.

2.3.2 Provisions' Proof of Reserves Protocol for Bitcoin

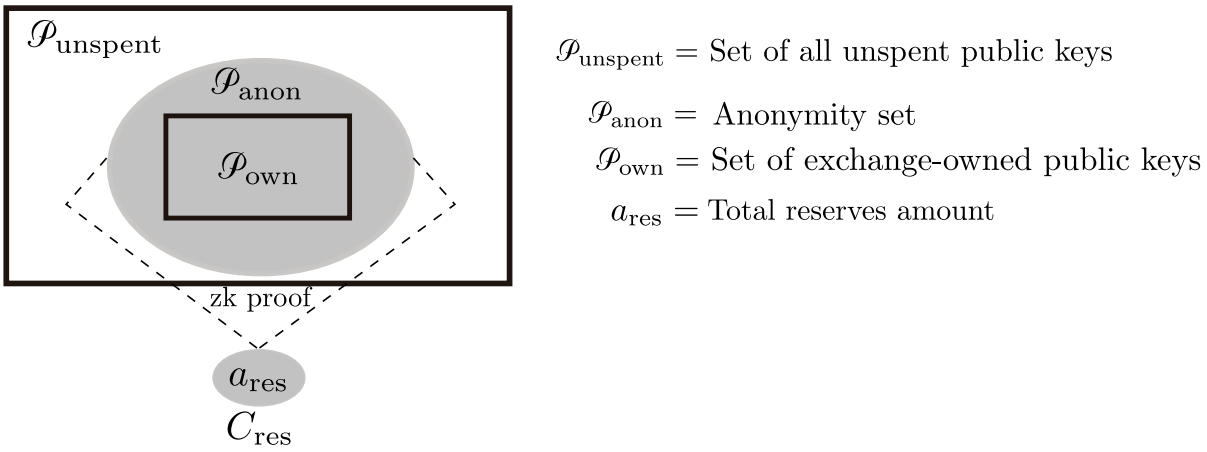

Figure 2.2: Provisions' proof of reserves protocol.

The Provisions scheme [1] proposed the first privacy focused proof of reserves protocol for Bitcoin exchanges based on standard cryptographic assumptions and without

relying on any trusted setup. Figure 2.2 gives a schematic diagram presenting the main idea behind the Provisions' proof of reserves protocol (PPoRP). In Figure 2.2, the set $\mathscr{P}_{\text{unspent}}$ gives the set of all Bitcoin public keys containing unspent amounts at a particular state of the blockchain. The Provisions scheme supports only those address types which correspond to a single public key. These include all pay-to-pubkey (P2PK) addresses and those pay-to-pubkey-hash (P2PKH) addresses which have been appeared as transaction inputs at least once [1, Section 4.2]. The set $\mathscr{P}_{\text{unspent}}$ is formed by assembling all such unspent addresses in the Bitcoin unspent transaction outputs (UTXO) set. The set $\mathscr{P}_{\text{own}}$ denotes the set of all exchange-owned public keys containing unspent amounts. Let a_{res} be the total reserves amount stored in all the public keys in the set $\mathscr{P}_{\text{own}}$. To preserve the privacy of the exchange, PPoRP takes the following steps.

1. To obfuscate the owned public keys, it publishes a set $\mathscr{P}_{\text{anon}}$ such that $\mathscr{P}_{\text{own}} \subseteq \mathscr{P}_{\text{anon}} \subseteq \mathscr{P}_{\text{unspent}}$.

2. To hide the amount a_{res}, it publishes a Pedersen commitment C_{res} to the amount a_{res} computed using all the public keys in $\mathscr{P}_{\text{anon}}$.

3. The associated zero-knowledge argument of knowledge proves that only the amounts corresponding to public keys in $\mathscr{P}_{\text{own}}$ are added to compute C_{res}.

To achieve the above, PPoRP uses sigma protocols which are converted into non-interactive arguments of knowledge. The intuition behind PPoRP is given below.

Intuition behind PPoRP

We consider the Bitcoin elliptic curve group $\mathbb{G}$ with prime order q and generator G. Let there be n public keys in the set $\mathscr{P}_{\text{anon}} = \{P_1, P_2, \ldots, P_n\}$. Let $i_j, j \in \{1, 2, \ldots, |\mathscr{P}_{\text{own}}|\}$ be the secret index of the jth owned public key in $\mathscr{P}_{\text{anon}}$. The exchange proceeds as follows.

- To prove ownership of the jth owned public key $Y_{i_j} \in \mathscr{P}_{\text{anon}}$, the exchange has to prove the knowledge of the secret key x_{i_j} such that $Y_{i_j} = x_{i_j} G$. To do this without revealing the secret indices, the exchange computes and publishes n group elements as follows.

$$L_i = \begin{cases} Y_i + t_i H & \text{if } Y_i \in \mathscr{P}_{\text{own}}, \\ t_i H & \text{if } Y_i \notin \mathscr{P}_{\text{own}}, \end{cases} \tag{2.6}$$

where H is another generator of $\mathbb{G}$ having unknown discrete logarithm relationship with G and $t_i \xleftarrow{\$} \mathbb{Z}_q$ is a randomly chosen scalar. Then for each public key $Y_i \in \mathscr{P}_{\text{anon}}$, the exchange needs to prove the following disjunctive statement,

$$\textit{Either it knows } x_i, t_i \textit{ such that } L_i = x_i G + t_i H \textit{ OR it knows } t_i \textit{ such that } L_i = t_i H. \tag{2.7}$$

- For each public key $Y_i \in \mathscr{P}_{\text{anon}}$, the corresponding balance $bal(Y_i)$ can be obtained by anybody querying the blockchain. In spite of this, the PPoRP enables the exchange to hide its total reserves a_{res} in a Pedersen commitment C_{res}. The total reserves of the exchange is given by,

$$a_{\text{res}} = \sum_{j=1}^{|\mathscr{P}_{\text{own}}|} bal(Y_{i_j}). \tag{2.8}$$

First, for each $Y_i \in \mathscr{P}_{\text{anon}}$, the exchange computes and publishes a group element $B_i = bal(Y_i)G$. Then, to obfuscate which amounts are added in C_{res}, the exchange defines and publishes n group elements as,

$$P_i = \begin{cases} B_i + r_i H & \text{if } Y_i \in \mathscr{P}_{\text{own}}, \\ r_i H & \text{if } Y_i \notin \mathscr{P}_{\text{own}}, \end{cases} \tag{2.9}$$

where $r_i \xleftarrow{\$} \mathbb{Z}_q$ is a randomly chosen scalar. If the definition in equation (2.9) is true, then the commitment[¶] to the total reserves can be calculated as,

$$\begin{aligned} C_{\text{res}} = \sum_{i=1}^{n} P_i = \sum_{j=1}^{|\mathscr{P}_{\text{own}}|} B_{i_j} + \left(\sum_{i=1}^{n} r_i\right) H &= \left(\sum_{j=1}^{|\mathscr{P}_{\text{own}}|} bal(Y_{i_j})\right) G + \left(\sum_{i=1}^{n} r_i\right) H \\ &= a_{\text{res}} G + r_{\text{res}} H, \end{aligned} \tag{2.10}$$

where $r_{\text{res}} \coloneqq \sum_{i=1}^{n} r_i$.

The PPoRP gives the disjunctive proofs for the statement in equation (2.7) and also proves that the definition given in equation (2.9) holds for each P_i. Below we discuss how it is done.

[¶]In the description of PPoRP, the amount and the blinding factor in a Pedersen commitment are associated with the bases G and H respectively. However in both MProvisions and MProve, the amount and the blinding factor are associated with the bases H and G respectively to be consistent with the convention of Monero.

Overview of PPoRP

For each public key $Y_i \in \mathscr{P}_{\text{anon}}, i \in \{1, 2, \ldots, n\}$, the exchange chooses a binary variable $s_i \in \{0, 1\}$ such that $s_i = 1$ when $Y_i \in \mathscr{P}_{\text{own}}$. When $Y_i \notin \mathscr{P}_{\text{own}}$, s_i is set to 0. Using these binary variables, equations in equation (2.6) and (2.9) can be alternatively expressed as,

$$L_i = s_i Y_i + t_i H, \quad i \in \{1, 2, \ldots, n\}, \tag{2.11}$$

$$P_i = s_i B_i + r_i H, \quad i \in \{1, 2, \ldots, n\}. \tag{2.12}$$

The exchange needs to prove knowledge of $\{s_i, r_i, t_i\}_{i=1}^n$ such that equations in equation (2.11) and (2.12) hold. It additionally needs to prove when $s_i = 1$, it knows x_i such that $Y_i = x_i G$. To do so, the exchange proves the knowledge of representation of L_i with respect to the bases G and H as,

$$L_i = s_i x_i G + t_i H = \hat{x}_i G + t_i H, \quad i \in \{1, 2, \ldots, n\}, \tag{2.13}$$

where $\hat{x}_i \coloneqq s_i x_i$. Notice that the exchange knows x_{i_j} for $j \in \{1, 2, \ldots, |\mathscr{P}_{\text{own}}|\}$. However, it knows $\hat{x}_i$ for all $i \in \{1, 2, \ldots, n\}$ i.e. $\hat{x}_i = x_i$ when $s_i = 1$, $\hat{x}_i = 0$ when $s_i = 0$. Moreover subtracting an equation in equation (2.11) from an equation in equation (2.13) for a particular i, we get

$$s_i Y_i = \hat{x}_i G. \tag{2.14}$$

Equation (2.14) implies when $s_i = 1$, the exchange knows $\hat{x}_i$ such that $Y_i = \hat{x}_i G$, i.e. the exchange owns the public key Y_i.

For each $i \in \{1, 2, \ldots, n\}$, the exchange proves knowledge of quantities $s_i \in \{0, 1\}$, and $r_i, t_i, \hat{x}_i \in \mathbb{Z}_q$ such that the following statement holds.

$$P_i = s_i B_i + r_i H \ \wedge \ L_i = s_i Y_i + t_i H \ \wedge \ L_i = \hat{x}_i G + t_i H, \tag{2.15}$$

where $\wedge$ denotes the logical AND operator. For proving $s_i \in \{0, 1\}$, a sigma protocol is used which utilizes the following fact,

$$x \in \{0, 1\} \iff x(x-1) = 0 \iff x^2 = x.$$

In the next section, we describe the design of MProvisions which is based on the above idea.

2.4 Proposed Privacy Focused Proof of Reserves Protocols for Monero

In the last section, we discussed the privacy issues of the existing proof of reserves scheme for Monero. We have also discussed PPoRP, a privacy focused proof of reserves scheme for Bitcoin. In this section, we describe MProvisions, the proposed privacy focused proof of reserves scheme for Monero which is obtained by modifying PPoRP. Then we describe MProve, another proposed privacy focused proof of reserves scheme for Monero. Unlike MProvisions which uses sigma protocols, MProve uses linkable ring signatures and ring signatures as primitives. It also performs better than MProvisions.

2.4.1 MProvisions Protocol

First, let us try to use PPoRP directly for Monero. We consider the Monero elliptic curve group $\mathbb{G}$ with prime order q and generators G and H having unknown discrete logarithm relationship with each other. We assume that the decisional Diffie-Hellman problem is hard in $\mathbb{G}$.

Suppose a Monero exchange chooses n one-time addresses from the Monero blockchain to construct the anonymity set $\mathscr{P}_{\text{anon}} = \{P_1, P_2, \ldots, P_n\}$. The set $\mathscr{P}_{\text{anon}}$ includes all its owned one-time addresses which collectively form the set $\mathscr{P}_{\text{own}}$. The set $\mathscr{P}_{\text{anon}}$ also includes some randomly chosen one-time addresses which will work as *cover addresses*. Let $i_j, j \in \{1, 2, \ldots, |\mathscr{P}_{\text{own}}|\}$ be the secret index of the jth owned one-time address in $\mathscr{P}_{\text{anon}}$. The exchange proceeds as follows.

- To prove ownership of the jth owned one-time address $P_{i_j} \in \mathscr{P}_{\text{anon}}$, the exchange has to prove the knowledge of the secret key x_{i_j} such that $P_{i_j} = x_{i_j}G$. To do this without revealing the secret indices, the exchange publishes n group elements

$$N_i = s_i P_i + f_i H, \quad i \in \{1, 2, \ldots, n\}, \tag{2.16}$$

  similar to PPoRP, and proves knowledge of representation of each N_i with respect to base pairs P_i, H and G, H. Here, $s_i \in \{0, 1\}, f_i \xleftarrow{\$} \mathbb{Z}_q$ are the secret bit and scalar chosen for each i respectively. The bit s_i is set to 1 when P_i is an exchange-owned address. It is set to 0 when P_i is not an exchange-owned address. The alternative

representation of N_i with respect to bases G and H is given by,

$$N_i = s_i x_i G + f_i H = \hat{x}_i G + f_i H, \quad i \in \{1, 2, \ldots, n\}. \tag{2.17}$$

As in PPoRP, the difference between these two alternative representations of N_i (given in equation (2.16) and (2.17)) for each i gives,

$$s_i P_i = \hat{x}_i G. \tag{2.18}$$

Equation (2.18) implies when $s_i = 1$, the exchange knows $\hat{x}_i$ such that $P_i = \hat{x}_i G$, i.e. the exchange owns the one-time address P_i.

- In Monero, each one-time address $P_i \in \mathcal{P}_{\text{anon}}$ is already associated with a commitment C_i to the corresponding amount a_i i.e. $C_i = y_i G + a_i H$. To construct C_{res}, the exchange selectively and homomorphically adds these commitments similar to PPoRP. In particular for each $i \in \{1, 2, \ldots, n\}$, the exchange chooses a blinding factor $k_i \xleftarrow{\$} \mathbb{Z}_q$ and publishes the following quantities.

$$B_i = s_i C_i + k_i G. \tag{2.19}$$

Let a_{res} denote the total reserves amount owned by the exchange. Let $C_{\text{res}} = bG + a_{\text{res}} H$, where $b \in \mathbb{Z}_q$ is some blinding factor. We calculate C_{res} as,

$$C_{\text{res}} = \sum_{i=1}^{n} B_i. \tag{2.20}$$

The amount corresponding to C_i is included in C_{res} only if $s_i = 1$, otherwise some randomness (k_i) is included with zero amount i.e. no H component is added.

Finally, the exchange proves knowledge of quantities $s_i \in \{0, 1\}$, and $k_i, t_i, \hat{x}_i \in \mathbb{Z}_q$ such that the following statement holds for each $i \in \{1, 2, \ldots, n\}$.

$$B_i = s_i C_i + k_i G \ \wedge \ N_i = s_i P_i + f_i H \ \wedge \ N_i = \hat{x}_i G + f_i H. \tag{2.21}$$

The above protocol does not work for Monero because of the following reason. In Monero, it is not revealed whether a one-time address in the Monero blockchain is spent or not (as discussed in Section 2.2.2). As a result, a malicious exchange could use the amount of an already spent address while calculating the total reserves. Hence in a Monero proof of reserves scheme, an exchange needs to prove that only the amounts corresponding to the unspent one-time addresses are added to the total reserves amount a_{res} besides proving the ownership of these addresses.

Construction of MProvisions

The MProvisions scheme publishes the key images of the owned addresses to prove that they are not spent. Suppose the exchange only publishes the key images I_{i_j} corresponding to the exchange-owned addresses P_{i_j} for all $j \in \{1, 2, \ldots, |\mathscr{P}_{\text{own}}|\}$. It has to prove that the published key image I_{i_j} is valid by showing knowledge of x_{i_j} for each j such that the following statements hold.

$$P_{i_j} = x_{i_j} G \ \wedge \ I_i = x_{i_j} H_p(P_{i_j}), \quad j \in \{1, 2, \ldots, |\mathscr{P}_{\text{own}}|\}. \tag{2.22}$$

The verifier can check that P_{i_j} is an unspent address by checking that I_{i_j} is not a member of the set of already appeared key images $\mathscr{I}$. However as P_{i_j} is used to prove the validity of I_{i_j}, the fact that P_{i_j} is an exchange-owned address is also revealed.

To obfuscate the source addresses, the exchange publishes a *dummy* key image I_i for each $P_i \in \mathscr{P}_{\text{anon}} \setminus \mathscr{P}_{\text{own}}$. First, note that for such a cover address $P_i = x_i G$, the exchange does not know the secret key x_i and cannot calculate the corresponding key image $I_i = x_i H_p(P_i)$. Instead, the exchange chooses a long term[‖] secret key $k_{\text{exch}} \xleftarrow{\$} \mathbb{Z}_q$. For a cover address $P_i \notin \mathscr{P}_{\text{own}}$, the exchange generates a random scalar $x'_i = H_s(P_i, k_{\text{exch}})$. Then it computes the dummy key image as $I_i = x'_i H_p(P_i)$. For a one-time address $P_i = x_i G$, let its hash $H_p(P_i)$ be equal to $y_i G$ for some $y_i \in \mathbb{Z}_q$. Then if $P_i \in \mathscr{P}_{\text{own}}$, the triple $(P_i = x_i G, H_p(P_i) = y_i G, I_i = x_i H_p(P_i) = x_i y_i G)$ is a Decisional Diffie-Hellman (DDH) triple. However if $P_i \notin \mathscr{P}_{\text{own}}$, the dummy key image is $I_i = x'_i H_p(P) = x'_i y_i G$. In this case, the triple $(P_i, H_p(P_i), I_i)$ is not a DDH triple as $x'_i \neq x_i$. However, no PPT distinguisher can distinguish between a real key image and a dummy key image owing to the DDH assumption.

To prove the validity of the real key images without revealing them, the exchange publishes n group elements,

$$M_i = s_i I_i + e_i H, \quad i \in \{1, 2, \ldots, n\}, \tag{2.23}$$

where $e_i \xleftarrow{\$} \mathbb{Z}_q$ is a randomly chosen scalar. The alternative representation of M_i with respect to bases $H_p(P_i)$ and H is given by,

$$M_i = \hat{x}_i H_p(P_i) + e_i H, \quad i \in \{1, 2, \ldots, n\}, \tag{2.24}$$

[‖] The usage of the long term secret key k_{exch} helps to preserve the privacy of the exchange even when multiple MProvisions proofs are published. This is discussed in detail in Section 2.6.3.

where $\hat{x}_i = s_i x_i$ if $P_i \in \mathscr{P}_{\text{own}}$, else $\hat{x}_i = s_i x'_i$. By the similar argument as above, the difference between these two alternative representations of M_i (given in equation (2.23) and equation (2.24)) for each i gives,

$$s_i I_i = \hat{x}_i H_p(P_i). \tag{2.25}$$

Equation (2.25) implies when $s_i = 1$, the exchange knows $\hat{x}_i$ such that $I_i = \hat{x}_i H_p(P_i)$.

Instead of statement in equation (2.22), the exchange proves knowledge of $s_i \in \{0, 1\}$, and $k_i, e_i, f_i, \hat{x}_i \in \mathbb{Z}_q$ such that the following statement holds for each $i \in \{1, 2, \ldots, n\}$.

$$\begin{aligned} B_i = s_i C_i + k_i G \ \wedge \ M_i = s_i I_i + e_i H \ \wedge \ N_i = s_i P_i + f_i H \\ \wedge \ M_i = \hat{x}_i H_p(P_i) + e_i H \ \wedge \ N_i = \hat{x}_i G + f_i H. \end{aligned} \tag{2.26}$$

As $\hat{x}_i$ is the same for M_i and N_i, $I_i = \hat{x}_i H_p(P_i) \ \wedge \ P_i = \hat{x}_i G$ holds for the same $\hat{x}_i$ when $s_i = 1$. Hence the validity of the real key images is verified. The MProvisions scheme uses sigma protocols which can be made non-interactive using public coin challenges from the verifier and the Fiat-Shamir heuristic [35]. The argument of knowledge for the MProvisions scheme is given below.

Argument of knowledge for MProvisions

Public data from blockchain: (P_i, C_i) for $i \in \{1, 2, \ldots, n\}$.

Verifier's input from prover: (B_i, I_i, M_i, N_i) for $i \in \{1, 2, \ldots, n\}$.

Prover's (the exchange) input: $s_i \in \{0, 1\}, k_i, e_i, f_i, \hat{x}_i \in \mathbb{Z}_q$ for $i \in \{1, 2, \ldots, n\}$.

---

1. For $i \in \{1, 2, \ldots, n\}$

   (a) Prover chooses:

   $$u_i^{(1)}, u_i^{(2)}, u_i^{(3)}, u_i^{(4)}, u_i^{(5)} \xleftarrow{\$} \mathbb{Z}_q.$$

   (b) The prover sends to the verifier:

   $$\begin{aligned} A_i^{(1)} &= u_i^{(1)} C_i + u_i^{(2)} G, \ A_i^{(2)} = u_i^{(1)} I_i + u_i^{(3)} H, \\ A_i^{(3)} &= u_i^{(1)} P_i + u_i^{(4)} H, A_i^{(4)} = u_i^{(5)} H_p(P_i) + u_i^{(3)} H \\ A_i^{(5)} &= u_i^{(5)} G + u_i^{(4)} H. \end{aligned}$$

(c) The verifier replies with a challenge $c_i \xleftarrow{\$} \mathbb{Z}_q$.

(d) Prover replies with :

$$\begin{aligned} r_{s_i} &= u_i^{(1)} + c_i \cdot s_i, \\ r_{k_i} &= u_i^{(2)} + c_i \cdot k_i, \\ r_{e_i} &= u_i^{(3)} + c_i \cdot e_i, \\ r_{f_i} &= u_i^{(4)} + c_i \cdot f_i, \\ r_{\hat{x}_i} &= u_i^{(5)} + c_i \cdot \hat{x}_i. \end{aligned}$$

(e) The verifier accepts if:

$$\begin{aligned} r_{s_i} C_i + r_{k_i} G &\stackrel{?}{=} c_i B_i + A_i^{(1)}, \\ r_{s_i} I_i + r_{e_i} H &\stackrel{?}{=} c_i M_i + A_i^{(2)}, \\ r_{s_i} P_i + r_{f_i} H &\stackrel{?}{=} c_i N_i + A_i^{(3)}, \\ r_{\hat{x}_i}\ H_p(P_i) + r_{e_i} H &\stackrel{?}{=} c_i M_i + A_i^{(4)}, \\ r_{\hat{x}_i} G + r_{f_i} H &\stackrel{?}{=} c_i N_i + A_i^{(5)}. \end{aligned}$$

(f) Run the protocol in Appendix B of Provisions [1] (with B_i as verifier's input) to prove knowledge of $s'_i \in \{0, 1\}$ and $k'_i \in \mathbb{Z}_q$ satisfying equation (2.19) (the binding property of Pedersen commitments ensures that $s_i = s'_i$ and $k_i = k'_i$)

2. Finally the verifier computes $C_{\texttt{res}} \coloneqq \sum_{i=1}^{n} B_i$ as in equation (2.20) .

Before starting the verification process as described above, the verifier first reads all the key images (I_is) published by the exchange. If any of them has already appeared in the Monero blockchain i.e. a member of the set $\mathcal{I}$, the verifier rejects the proof. Otherwise the verifier continues with the proof.

The proof that the MProvisions scheme is a perfect special-honest-verifier-zero-knowledge (SHVZK) protocol is given in Appendix A.1. The privacy features and the other security properties are discussed in Section 2.6.

2.4.2 MProve Protocol

In this section, we describe MProve, which is another privacy focused proof of reserves protocol for a Monero exchange giving better performance than MProvisions. MProve uses ring signatures and linkable ring signatures as primitives. As the linkable ring signature is discussed in Section 2.2.2, we give the ring signature generation and verification algorithm before describing the protocol.

Ring Signatures

Given n public keys, a ring signature [36] on a message m proves that one of the corresponding secret keys was used to sign the message without revealing which one. Suppose Alice wants to sign a message m using a ring signature involving the public keys $\mathscr{P} = (P_0, P_1, \ldots, P_{n-1})$. She knows the secret key x_j corresponding to P_j i.e. $P_j = x_j G$, for some $j \in \{0, 1, \ldots, n-1\}$. She generates the ring signature γ as follows:

1. She samples α and $s_i, i \in \{0, 1, \ldots, n-1\}, i \neq j$, randomly from $\mathbb{Z}_q$.

2. She computes a group element $L_j = \alpha G$ and an integer $c_{j+1} = H_s(\mathscr{P}, m, L_j)$ where $H_s : \{0,1\}^* \mapsto \mathbb{Z}_q$ is a hash function.

3. Increasing j modulo n, she computes points $L_{j+1}, L_{j+2}, \ldots, L_{j-1}$ and scalars c_{j+2}, $c_{j+3}, \ldots, c_j$ as

$$\begin{aligned} L_{j+1} &= s_{j+1}G + c_{j+1}P_{j+1}, \\ c_{j+2} &= H_s(\mathscr{P}, m, L_{j+1}), \\ &\vdots \\ L_{j-1} &= s_{j-1}G + c_{j-1}P_{j-1}, \\ c_j &= H_s(\mathscr{P}, m, L_{j-1}). \end{aligned}$$

4. Finally, she computes $s_j = \alpha - c_j x_j$. As L_j was computed using α in step 2, this implies that

$$L_j = \alpha G = (s_j + c_j x_j)G = s_j G + c_j P_j.$$

5. The ring signature on the message m is given by $\gamma = (c_0, s_0, s_1, \ldots, s_{n-1})$.

To check the validity of a ring signature, a verifier does the following:

1. Using the public key list $\mathscr{P} = (P_0, P_1, \ldots, P_{n-1})$, the message m, and the ring signature $\gamma = (c_0, s_0, s_1, \ldots, s_{n-1})$, the verifier calculates the scalars $c_k, k = 1, 2, \ldots, n-1$, as

$$\begin{aligned} L_0 &= s_0 G + c_0 P_0, \\ c_1 &= H_s(\mathscr{P}, m, L_0), \\ &\vdots \\ L_{n-2} &= s_{n-2} G + c_{n-2} P_{n-2}, \\ c_{n-1} &= H_s(\mathscr{P}, m, L_{n-2}). \end{aligned}$$

2. Finally, c_{n-1} and s_{n-1} are used to calculate c_0' as

$$\begin{aligned} L_{n-1} &= s_{n-1} G + c_{n-1} P_{n-1}, \\ c_0' &= H_s(\mathscr{P}, m, L_{n-1}). \end{aligned}$$

3. The signature γ is accepted if c_0' equals the c_0 given in γ. Otherwise, it is rejected.

Note that $c_0' = c_0$ holds only if the signer knows at least one secret key corresponding to the public keys in the ring and computes γ according to the above signature generation algorithm. However, the verifier does not get to know the secret index j while verifying γ.

Motivating the MProve Protocol

In this section, we motivate the MProve protocol. Suppose an exchange owns a set of one-time addresses $\mathscr{P}_{\text{own}}$ on the Monero blockchain, i.e. it knows the secret key corresponding to each address in $\mathscr{P}_{\text{own}}$. Like MProvisions, the exchange publishes a set of one-time addresses $\mathscr{P}_{\text{anon}} = \{P_1, P_2, \ldots, P_n\}$ such that $\mathscr{P}_{\text{own}}$ is a subset of $\mathscr{P}_{\text{anon}}$. Let $\mathscr{I}_{\text{own}} = \{i \mid P_i \in \mathscr{P}_{\text{own}}\}$ be the set of indices[**] i such that the exchange knows the secret key corresponding to P_i. Each address $P_i \in \mathscr{P}_{\text{anon}}$ is associated with a unique Pedersen commitment C_i on the Monero blockchain. Suppose C_i is a commitment to an amount a_i,

[**]In MProvisions, these indices were denoted by i_j for $j \in \{1, 2, \ldots, |\mathscr{P}_{\text{own}}|\}$. Here, we have changed the notation for the sake of convenience.

that is $C_i = y_i G + a_i H$ for a blinding factor $y_i \in \mathbb{Z}_q$. Then the Monero reserves amount of the exchange is equal to $a_{\text{res}} = \sum_{i \in \mathcal{I}_{\text{own}}} a_i$. The goal of the MProve protocol is to ensure that the exchange generates a Pedersen commitment C_{res} to an amount which is at most a_{res}. Using MProve, an exchange can generate C_{res} as a commitment to any amount of the form $\sum_{i \in \mathcal{I}} a_i$ where $\mathcal{I} \subseteq \mathcal{I}_{\text{own}}$. So C_{res} cannot be a commitment to an amount larger than a_{res}.

For each $P_i \in \mathcal{P}_{\text{anon}}$, the exchange publishes a Pedersen commitment C'_i and claims that the commitment $C_{\text{res}} = \sum_{i=1}^{n} (C_i - C'_i)$ is a commitment to an amount of Monero it owns. To support this claim, the exchange publishes n ring signatures and n linkable ring signatures.

- Let us consider the linkable ring signatures first. The ith linkable ring signature is over the ring $(P_i, C'_i - C_i)$. Now there are two scenarios: $P_i \notin \mathcal{P}_{\text{own}}$ and $P_i \in \mathcal{P}_{\text{own}}$.
  - If $P_i \notin \mathcal{P}_{\text{own}}$, then the exchange does not know the secret key corresponding to P_i. To generate a linkable ring signature over the ring $(P_i, C'_i - C_i)$, the exchange is forced to use the secret key corresponding to the public key $C'_i - C_i$. This in turn forces the exchange to choose C'_i such that $C'_i - C_i$ is of the form $z_i G$ where $z_i \in \mathbb{Z}_q$. On the contrary, if the exchange were to choose C'_i such that $C'_i - C_i$ is of the form $z_i G + aH$ for some nonzero $a \in \mathbb{Z}_q$, then to create the linkable ring signature it would need to calculate $z \in \mathbb{Z}_q$ such that $zG = z_i G + aH$. But this is beyond the capability of the computationally bounded exchange as the discrete logarithm of H with respect to G is unknown. Consequently, $C'_i - C_i$ is forced to be a commitment to zero whenever $P_i \notin \mathcal{P}_{\text{own}}$. Thus the contribution of such $C_i - C'_i$ in $C_{\text{res}} = \sum_{j=1}^{n} \left(C_j - C'_j\right)$ is a commitment to the zero amount.
  - If $P_i \in \mathcal{P}_{\text{own}}$, the exchange knows the secret key corresponding to P_i. It can generate the linkable ring signature using either this secret key or using z_i when C'_i is chosen such that $C'_i - C_i = z_i G$. Doing the latter is not in the interest of the exchange as it makes the contribution of $C_i - C'_i$ in $C_{\text{res}} = \sum_{j=1}^{n} \left(C_j - C'_j\right)$ a commitment to the zero amount. So the exchange is forced to use the secret key corresponding to P_i to generate the linkable ring signature whenever it wants the amount in C_i to be included in C_{res}. An important consequence is that the key images of all P_is used by the exchange as sources of funds are revealed (via

the linkable ring signature). This prevents the exchange from using an already spent address as a source of funds as the key images of spent addresses appear on the Monero blockchain. The verifier of an MProve proof rejects it if any of the n key images generated by the linkable ring signatures has already appeared on the Monero blockchain i.e. is a member of the set $\mathcal{I}$. Note that the key images of $C'_i - C_i$ generated in the case when $P_i \notin \mathcal{P}_{\text{own}}$ can appear on the blockchain only with a negligible probability if the z_is are randomly chosen from $\mathbb{Z}_q$.

- Now let us motivate the need for the ring signatures. The ith ring signature is over the ring $(C'_i, C'_i - C_i)$. Note that the linkable ring signature over $(P_i, C'_i - C_i)$ does not impose any restriction on C'_i when $P_i \in \mathcal{P}_{\text{own}}$, as the exchange can use the secret key corresponding to P_i to generate the signature. Without any further constraint on C'_i, an exchange can inflate the amount committed in C_{res} to a value larger than a_{res}. For example, suppose $P_1 \in \mathcal{P}_{\text{own}}$. For any $b \in \mathbb{Z}_q$ such that $b > a_{\text{res}}$ and $z_i \in \mathbb{Z}_q$, the exchange can set

$$\begin{aligned} C'_1 &= z_1 G + C_1 + (q-b)H, \\ C'_i &= z_i G + C_i \qquad \text{for } i = 2, 3, \ldots, n, \end{aligned} \tag{2.27}$$

and claim that the commitment to its reserves is given by

$$C_{\text{res}} = \sum_{i=1}^{n} C_i - C'_i = -\sum_{i=1}^{n} z_i G + bH. \tag{2.28}$$

To prevent this, we require the exchange to publish a ring signature over the ring $(C'_i, C'_i - C_i)$ for all $i = 1, 2, \ldots, n$. Let us once again consider the two scenarios: $P_i \notin \mathcal{P}_{\text{own}}$ and $P_i \in \mathcal{P}_{\text{own}}$.

  - If $P_i \notin \mathcal{P}_{\text{own}}$, then the exchange uses the secret key corresponding to $C'_i - C_i$ to generate the linkable ring signature over the ring $(P_i, C'_i - C_i)$. The exchange can then use the same secret key to generate the ring signature over the ring $(C'_i, C'_i - C_i)$.
  - If $P_i \in \mathcal{P}_{\text{own}}$, then the exchange uses the secret key corresponding to P_i to generate the linkable ring signature over the ring $(P_i, C'_i - C_i)$, to avoid making the contribution of $C_i - C'_i$ to C_{res} a commitment to zero. To generate the ring signature over the ring $(C'_i, C'_i - C_i)$ while keeping the contribution of $C_i - C'_i$ to C_{res}

nonzero, the exchange is forced to use the secret key corresponding to C'_i. This in turn forces the exchange to choose C'_i to be of the form $z_i G$ where $z_i \in \mathbb{Z}_q$, which is a commitment to the zero amount. Consequently, the contribution of $C_i - C'_i$ to $C_{\text{res}} = \sum_{j=1}^{n} \left(C_j - C'_j\right)$ will be a commitment which has the same amount as C_i. Note that a computationally bounded exchange would not be able to generate the ring signature using the secret key corresponding to the C'_1 in equation (2.27) as this would require calculating the discrete logarithm of H with respect to G.

To summarize, the linkable ring signature over $(P_i, C'_i - C_i)$ forces the exchange to reveal the key images of all $P_i \in \mathscr{P}_{\text{own}}$ it wants to use as sources of funds and also forces the exchange to make $C_i - C'_i$ a commitment to the zero amount whenever $P_i \notin \mathscr{P}_{\text{own}}$. The ring signature over $(C'_i, C'_i - C_i)$ forces the exchange to generate C'_i as a commitment to either the zero amount or to the same amount as C_i. As the exchange claims that $C_{\text{res}} = \sum_{j=1}^{n}(C_j - C'_j)$ is a commitment to its reserves, it can only include amounts in C_i and that too only when $P_i \in \mathscr{P}_{\text{own}}$. A precise description of the MProve protocol is given next.

Proof Generation of the MProve Protocol

The MProve proof of reserves protocol proceeds as follows:

1. The exchange chooses a set of one-time addresses $\mathscr{P}_{\text{anon}} = (P_1, P_2, \ldots, P_n)$ from the Monero blockchain such that it knows the secret keys corresponding to a subset $\mathscr{P}_{\text{own}}$ of $\mathscr{P}_{\text{anon}}$. The set $\mathscr{P}_{\text{anon}}$ is made public by the exchange.

2. For each $P_i \in \mathscr{P}_{\text{anon}}$, anybody can read the corresponding Pedersen commitment C_i from the blockchain. Let C_i be the commitment to an amount a_i with blinding factor y_i, i.e.

$$C_i = C(y_i, a_i) = y_i G + a_i H. \tag{2.29}$$

   For $P_i \in \mathscr{P}_{\text{own}}$, the exchange knows y_i and a_i. For $P_i \notin \mathscr{P}_{\text{own}}$, the exchange may know y_i and a_i if it was the party which sent funds to P_i. In general, the exchange will not know y_i and a_i for $P_i \notin \mathscr{P}_{\text{own}}$.

3. The exchange chooses a long term key[††] $k_{\text{exch}} \xleftarrow{\$} \mathbb{Z}_q$. For each $P_i \in \mathscr{P}_{\text{anon}}$, the

[††]The reason for choosing this long term key is to ensure privacy for multiple proofs. This is discussed in detail in Section 2.6.3.

exchange sets $z_i = H_s(k_{\text{exch}}, P_i)$ and generates C'_i as

$$C'_i = \begin{cases} z_i G & \text{if } P_i \in \mathscr{P}_{\text{own}}, \\ z_i G + C_i & \text{if } P_i \notin \mathscr{P}_{\text{own}}. \end{cases} \tag{2.30}$$

4. For each $i = 1, 2, \ldots, n$, the exchange publishes a regular ring signature γ_i on a message m verifiable by the pair of public keys $(C'_i, C'_i - C_i)$. The calculation of γ_i is described in Appendix A.2.

5. For each $i = 1, 2, \ldots, n$, the exchange publishes a linkable ring signature σ_i on a message m verifiable by the pair of public keys $(P_i, C'_i - C_i)$. The calculation of σ_i is described in Appendix A.3.

6. The exchange publishes a commitment C_{res} which satisfies the equation

$$\sum_{i=1}^{n} C_i = C_{\text{res}} + \sum_{i=1}^{n} C'_i. \tag{2.31}$$

   The exchange claims that C_{res} is a Pedersen commitment to the amount of Monero it owns.

The following theorem assures us that an honest exchange can correctly generate a commitment to its total reserves using MProve.

Theorem 2.1. *If an exchange follows the MProve protocol honestly, then C_{res} will be a commitment to the amount*

$$a_{res} = \sum_{i \in \mathcal{I}_{own}} a_i. \tag{2.32}$$

Proof. Consider the definition of C'_i given in equation (2.30).

- If $P_i \in \mathscr{P}_{\text{own}}$, C'_i is a commitment to zero. Hence $\sum_{i \in \mathcal{I}_{\text{own}}} C'_i$ is a commitment to the zero amount.

- If $P_i \notin \mathscr{P}_{\text{own}}$, $C'_i - C_i$ is a commitment to zero. Let $\mathcal{I}_{\text{unknown}} = \{i \mid 1 \leq i \leq n, P_i \notin \mathscr{P}_{\text{own}}\}$ denote the set of indices i such that the exchange does not know the secret key corresponding to P_i. Then $\sum_{i \in \mathcal{I}_{\text{unknown}}} (C'_i - C_i)$ is a commitment to the zero amount.

Rearranging equation (2.31), we get

$$\sum_{i \in \mathcal{J}_{\text{own}}} C_i = C_{\text{res}} + \sum_{i \in \mathcal{J}_{\text{own}}} C'_i + \sum_{i \in \mathcal{J}_{\text{unknown}}} (C'_i - C_i). \tag{2.33}$$

As the last two sums on the right hand side are commitments to zero, C_{res} and $\sum_{i \in \mathcal{J}_{\text{own}}} C_i$ must be commitments to the same amount. Since

$$\begin{aligned} \sum_{i \in \mathcal{J}_{\text{own}}} C_i &= \sum_{i \in \mathcal{J}_{\text{own}}} (y_i G + a_i H) \\ &= a_{\text{res}} H + \sum_{i \in \mathcal{J}_{\text{own}}} y_i G, \end{aligned} \tag{2.34}$$

C_{res} is a commitment to a_{res}. □

Proof Verification

The output of an exchange in the MProve protocol consists of the following:

- A set of one-time addresses $(P_1, P_2, \ldots, P_n)$.
- The commitments $C'_1, C'_2, \ldots, C'_n$ created by the exchange.
- The regular ring signatures $\gamma_i = (d_0^i, t_0^i, t_1^i)$ for $i = 1, 2, \ldots, n$.
- The linkable ring signatures $\sigma_i = (I_i, c_0^i, s_0^i, s_1^i)$ for $i = 1, 2, \ldots, n$.
- The message m used to create γ_i and σ_i.
- The commitment C_{res} which the exchange claims to be a commitment to its total reserves.

Verification involves the following operations:

1. The verifier checks that none of the key images I_i published by the exchange as part of the signatures σ_i appear in the set of already appeared key images $\mathcal{J}$. If a key image appears in $\mathcal{J}$, the verifier rejects the proof of reserves as it implies that the funds in the corresponding one-time address P_i have already been spent. If none of the key images have appeared in $\mathcal{J}$, the verifier continues with proof verification.
2. The verifier reads the commitments C_i corresponding to the P_is from the blockchain.
3. The public key $C'_i - C_i$ is computed for each i.

4. The public key pair $(C'_i, C'_i - C_i)$ is used to verify the regular ring signatures γ_i.

5. The public key pair $(P_i, C'_i - C_i)$ is used to verify the linkable ring signatures σ_i.

6. Equality in equation (2.31) is verified using the C_is, C'_is, and C_{res}.

Before discussing the security properties for both MProve and MProvisions, we describe a privacy issue both the schemes suffer from.

2.5 Drawback of MProve and MProvisions

When a source address is spent by the exchange in a future Monero transaction, then that transaction becomes traceable because of an MProve/MProvisions proof. In this section, we explain how this happens. First, let us consider a scenario which affects any Monero proof of reserves protocol that reveals the key images of exchange-owned addresses.

Example 2.1. *Consider a Monero transaction txn where a Monero exchange Ex is spending from a one-time address P. Before this transaction, the exchange has published some reserves proofs where P has been used as a source address. As a result, the corresponding key image (say I) of P has appeared in those reserves proofs. When P is being spent in txn, the same key image I will appear again in txn. As the same I has appeared in the reserves proofs published by Ex and in txn, the fact that Ex is spending in txn is revealed.*

The drawback shown in Example 2.1 exists in every proof of reserves protocol which has to reveal the key images of the source addresses explicitly to prove that they are not spent. The proof of reserves protocol proposed by Stoffu Noether [26], MProve, MProvisions, and MProve+ (discussed in the next chapter) suffer from this drawback. We discuss the challenges involved in proving that the source addresses are not spent without revealing their key images in the next chapter. For now, we discuss the case when the MProve scheme is used as the proof of reserves protocol in Example 2.1.

Effect of an MProve Proof on Monero Transactions

In Figure 2.3, we consider a single MProve proof where the size of the anonymity set is n. There are n key images $I_1, I_2, \ldots, I_n$ corresponding to the n linkable ring signatures. Among these key images, some are real key images (originated from a one-time address)

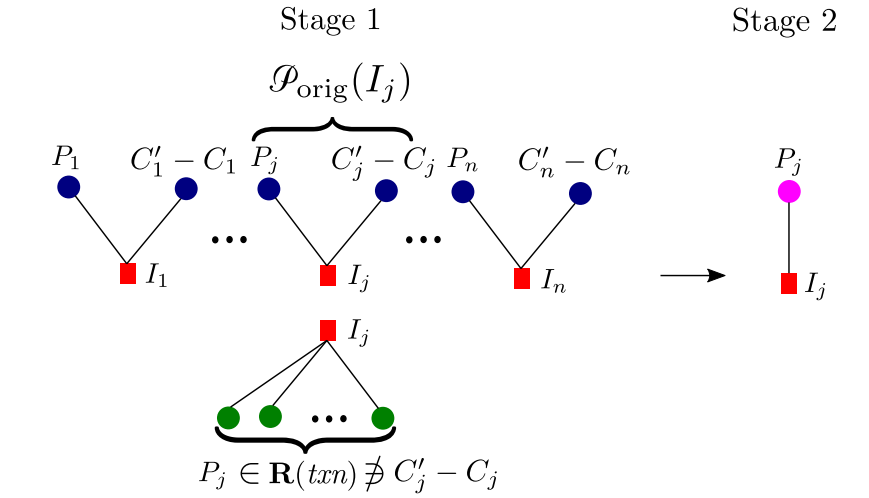

Figure 2.3: Linking key image for MProve when a source address is spent.

and some are dummy key images (originated from a group element which is not a one-time address). Consider the key image I_j which is generated from a one-time address P_j. Suppose there is a PPT adversary $\mathcal{A}$ who wants to obtain the real originator of I_j. When $\mathcal{A}$ observes this MProve proof, she infers that I_j could have originated either from P_j or from the group element $C'_j - C_j$. Hence I_j is connected to both P_j and $C'_j - C_j$ in Figure 2.3. Defining $\mathscr{P}_{\text{orig}}(I_j)$ as the set of possible originators of I_j, we have $\mathscr{P}_{\text{orig}}(I_j) = \{P_j, C'_j - C_j\}$. When the exchange spends from P_j in a future Monero transaction txn, I_j appears again. Let the ring of txn be $\mathbf{R}(txn)$. The set $\mathbf{R}(txn)$ must contain P_j as P_j is the source address being spent in txn. However, the group element $C'_j - C_j$ cannot be an element in the set $\mathbf{R}(txn)$ as it is unlikely to be a valid one-time address. The view of $\mathcal{A}$ in this situation is shown in terms of bipartite graphs in stage 1 of Figure 2.3. As the intersection between the sets $\mathscr{P}_{\text{orig}}(I_j)$ and $\mathbf{R}(txn)$ is a singleton set $\{P_j\}$, $\mathcal{A}$ successfully links I_j with P_j. This has been shown in stage 2 of Figure 2.3. So the probability that $\mathcal{A}$ successfully outputs P_j as the originator for I_j is,

$$\Pr[\mathcal{A}(I_j, \mathscr{P}_{\text{orig}}(I_j), \mathbf{R}(txn)) = P_j] = 1. \tag{2.35}$$

Effect of an MProvisions Proof on Monero Transactions

Next we consider the scenario when the MProvisions scheme is used in Example 2.1. In Figure 2.4, we consider a single MProvisions proof where there are n key images

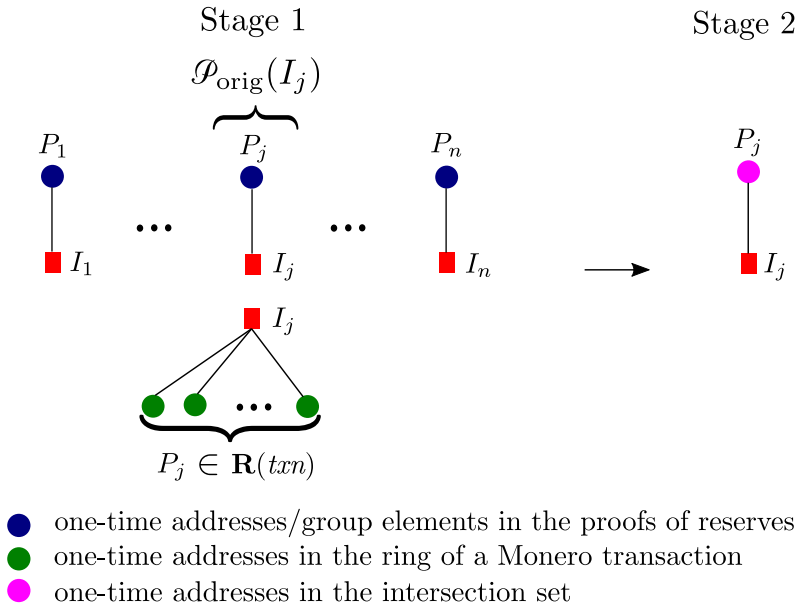

Figure 2.4: Linking key image for MProvisions when a source address is spent.

$I_1, I_2, \ldots, I_n$ corresponding to the n one-time addresses in the anonymity set. If a one-time address is a source, then the key image is the real key image of it. For a cover one-time address, the associated key image is not its real key image. Now consider the same PPT adversary $\mathcal{A}$ who wants to obtain the real originator of some $I_j \in \{I_1, I_2, \ldots, I_n\}$. By viewing the MProvisions proof she infers that either P_j is the real originator of I_j or I_j is not at all a key image. In any case the set $\mathscr{P}_{\text{orig}}(I_j)$ is the singleton set $\{P_j\}$. Next consider the situation when txn appears in the blockchain containing P_j in $\mathbf{R}(txn)$ and I_j as its key image. Now $\mathcal{A}$ is assured that the key image I_j is a real key image and P_j originates it. In this case also we have,

$$\Pr[\mathcal{A}(I_j, \mathscr{P}_{\text{orig}}(I_j), \mathbf{R}(txn)) = P_j] = 1. \tag{2.36}$$

Implication. For both MProve and MProvisions, txn loses the untraceability property (discussed in Section 2.2.2). In particular, it is revealed that the exchange is spending P_j in txn and the corresponding key image is I_j. As P_j is known to be spent, it cannot be used as a cover address in any transaction henceforth. Moreover, the effective ring sizes reduce by 1 for all transactions which have used P_j as a cover address so far. In spite of the above drawback, both the schemes exhibit the following security and privacy properties.

2.6 Security Properties of MProve and MProvisions

In this section, we describe the various security properties of both MProve and MProvisions. We shall sometimes find it convenient to describe a property of the MProve protocol first. Then we show that the MProvisions protocol also exhibits a similar attribute. Both the protocols have three main properties, namely, *collusion resistance*, *inflation resistance*, and *pre-spend privacy*. The collusion resistance property prevents two or more exchanges from sharing a common one-time address as a source of funds while generating a reserves proof. The inflation resistance property prevents an exchange from publishing a commitment C_{res} to an amount which exceeds the actual reserves amount. The pre-spend privacy property provides privacy to the exchange given that it has not spent from a source one-time address used in the reserves proofs.

2.6.1 Collusion Resistance

In both MProve and MProvisions protocols, the exchange has to publish a key image I for each address P in the anonymity set. If the amount corresponding to P contributes to the total reserves, then the following relation holds,

$$P = xG \wedge I = xH_p(P), \tag{2.37}$$

where x is the secret key corresponding to P. Note that for a given P, the key image I in equation (2.37) is unique. Hence if two PPT exchanges use a common one-time address as a source address to generate reserves proofs, the key image corresponding to that one-time address will appear in both the reserves proofs. Thus a verifier can easily detect collusion between exchanges. If we assume the exchanges are PPT, then they can generate different key images for the same source address only with a negligible probability. For MProve, this follows from the unforgeability of the linkable ring signatures. For MProvisions, it follows from the unforgeability of the argument of knowledge. Therefore, both the schemes are collusion resistant.

2.6.2 Inflation Resistance

For the MProve scheme, we shall prove that a PPT exchange cannot generate a Pedersen commitment to an amount which is more than what it owns. It can always omit an

amount corresponding to some owned address $P_j, j \in \mathcal{I}_{\text{own}}$, by setting $C'_j - C_j = w_j G$ for some $w_j \in \mathbb{Z}_q$ and signing the ring signature γ_j using w_j. Note that the linkable ring signature σ_j will always need to be generated by the secret key corresponding to P_j for $j \in \mathcal{I}_{\text{own}}$. Generating σ_j using the secret key corresponding to $C'_j - C_j$ for $j \in \mathcal{I}_{\text{own}}$ can reveal that the address P_j is owned by the exchange (as discussed in Section 2.6.3). If P_j is not owned by the exchange, it cannot include the corresponding amount in C_{res}. To show this, we give the following theorem which is proved in Appendix A.4.

Theorem 2.2. *Suppose an exchange creates a proof of reserves with commitment $C_{res} = C(y_{res}, a_{res})$ such that the reserves proof is accepted by the verification procedure in Section 2.4.2. Then the amount a_{res} must be of the form $\sum_{i \in \mathcal{I}_1} a_i$ for any set $\mathcal{I}_1 \subseteq \mathcal{I}_{own}$ i.e. $a_{res} = \sum_{i \in \mathcal{I}_1} a_i$.*

For an accepted MProvisions proof, the statement in equation (2.26) holds for each $i \in \{1, 2, \ldots, n\}$ which is given as,

$$B_i = s_i C_i + k_i G \ \wedge \ M_i = s_i I_i + e_i H \ \wedge \ N_i = s_i P_i + f_i H$$
$$\wedge \ M_i = \hat{x}_i H_p(P_i) + e_i H \ \wedge \ N_i = \hat{x}_i G + f_i H.$$

The above statement in turn implies that the amount in C_i is added to B_i, only if the exchange knows $\hat{x}_i$ such that $P_i = \hat{x}_i G \wedge I_i = \hat{x}_i H_p(P_i)$ holds i.e. the exchange owns P_i. As C_{res} is computed by adding[‡‡] B_i for each $i \in \{1, 2, \ldots, n\}$, a PPT exchange cannot inflate the total reserves amount.

2.6.3 Pre-spend Privacy

We have discussed the privacy issue when the exchange spends from a one-time address which is used as a source address in an MProve/MProvisions proof in Section 2.5. In this section, we consider the scenario when the exchange has not spent from any of the source one-time addresses used in MProve/MProvisions proofs. We also consider the general scenario when multiple reserves proofs are published. First, we discuss the necessity of the long term keys when multiple proofs are generated for both the protocols.

[‡‡] Checked in Step 2 of the MProvisions protocol.

Multiple Proofs and the Long Term Keys

First, we shall consider the MProve protocol. Suppose in equation (2.30) instead of using the long term key k_{exch}, the exchange randomly picks $z_i \in \mathbb{Z}_q$ and generates C'_i as

$$C'_i = \begin{cases} z_i G & \text{if } P_i \in \mathscr{P}_{\text{own}}, \\ z_i G + C_i & \text{if } P_i \notin \mathscr{P}_{\text{own}}. \end{cases}$$

Suppose a Monero exchange has published two MProve proofs with anonymity sets $\mathscr{P}^{(1)}_{\text{anon}}$ and $\mathscr{P}^{(2)}_{\text{anon}}$. Let $\mathscr{P}^{(1)}_{\text{own}}$ and $\mathscr{P}^{(2)}_{\text{own}}$ be the corresponding subsets of exchange-owned addresses. Suppose a one-time address P_i belongs to both the anonymity sets, i.e. $P_i \in \mathscr{P}^{(1)}_{\text{anon}} \cap \mathscr{P}^{(1)}_{\text{anon}}$. Let C_i be the Pedersen commitment corresponding to P_i on the Monero blockchain. Let $C'_{i,1}, C'_{i,2}$ be the commitments generated by the exchange for such a P_i in the two MProve proofs. If P_i belongs to $\mathscr{P}^{(1)}_{\text{own}} \cap \mathscr{P}^{(2)}_{\text{own}}$, then the key images revealed in the linkable ring signatures over the rings $(P_i, C'_{i,1} - C_i)$ and $(P_i, C'_{i,2} - C_i)$ in the two MProve proofs will be the same. On the other hand, suppose P_i does not belong to $\mathscr{P}^{(1)}_{\text{own}} \cap \mathscr{P}^{(2)}_{\text{own}}$. Now suppose the exchange chooses $z_{i,1}$ and $z_{i,2}$ independently and randomly from $\mathbb{Z}_q$ to generate $C'_{i,1} = z_{i,1}G + C_i$ and $C'_{i,2} = z_{i,2}G + C_i$. Then the key images revealed in the linkable ring signatures over the rings $(P_i, C'_{i,1} - C_i)$ and $(P_i, C'_{i,2} - C_i)$ in the two MProve proofs will be different. This difference will lead an observer to conclude that P_i is not an exchange-owned address. To avoid this leakage of information, the MProve protocol chooses the scalars z_i for the commitments C'_i corresponding to cover addresses in the anonymity set as deterministic functions of the addresses themselves. It chooses a long term key k_{exch} and sets $z_i = H_s(k_{\text{exch}}, P_i)$ where $H_s(\cdot)$ is a cryptographic hash function with outputs in $\mathbb{Z}_q$.

Now consider the MProvisions scheme. The long term key k_{exch} is again used for a similar reason. If the key images of cover addresses revealed by multiple MProvisions proofs change from one proof to another, then an observer would be able to identify them as addresses not owned by the exchange.

Omission of Amounts Corresponding to Exchange-Owned Addresses

If an exchange wants to omit the amount corresponding to an owned address P_i in an MProve/MProvisions proof, it cannot treat P_i as a cover address. It must reveal the true key image of P_i while nullifying the contribution of C_i to C_{res} by using the disjunctive

nature of the proofs. Otherwise, the fact that P_i is owned by the exchange will be revealed. To see this, consider once again the scenario of two MProve proofs where $P_i \in \mathscr{P}_{\text{own}}^{(1)} \cap \mathscr{P}_{\text{own}}^{(2)}$. Suppose while publishing the two MProve proofs, the exchange wants to use P_i as a cover address in the first proof and use it as a source address in the second proof. To do so, suppose the exchange generates the first linkable ring signatures $\sigma_{i,1}$ using the secret key corresponding to $C'_{i,1} - C_i$ and the second linkable ring signature $\sigma_{i,2}$ using the secret key corresponding to P_i. Then the key images in the two linkable ring signatures corresponding to P_i will be different and an adversary can identify that P_i is an exchange-owned address. Similarly, in MProvisions, the exchange should publish the real key image of P_i in both the proofs. When it wants to use P_i as a source address, it should set the corresponding $s_i = 1$. And when it wants to use P_i as a cover address, it should set the corresponding $s_i = 0$.

Next, we discuss the privacy of both the protocols given that the exchange has not spent from any source addresses in the reserves proofs.

Pre-spend Privacy for MProve

We consider the scenario when a Monero exchange publishes $f(\lambda)$ MProve proofs where $f(\lambda)$ denotes a polynomial of the security parameter λ. Let the size of the ith anonymity set be n_i. We denote the ith MProve proof by $\left(\mathbf{P}^{(i)}, \mathbf{C}^{(i)}, \mathbf{C'}^{(i)}, \mathbf{\Gamma}^{(i)}, \mathbf{\Sigma}^{(i)}, m_i, C_{\text{res}}^{(i)}\right)$ where,

$$\mathbf{P}^{(i)} = (P_{i,1}, P_{i,2}, \ldots, P_{i,n_i}), \tag{2.38}$$

$$\mathbf{C}^{(i)} = (C_{i,1}, C_{i,2}, \ldots, C_{i,n_i}), \tag{2.39}$$

$$\mathbf{C'}^{(i)} = (C'_{i,1}, C'_{i,2}, \ldots, C'_{i,n_i}), \tag{2.40}$$

$$\mathbf{\Gamma}^{(i)} = (\gamma_{i,1}, \gamma_{i,2}, \ldots, \gamma_{i,n_i}), \tag{2.41}$$

$$\mathbf{\Sigma}^{(i)} = (\sigma_{i,1}, \sigma_{i,2}, \ldots, \sigma_{i,n_i}), \tag{2.42}$$

and m_i denotes the message to be signed by the linkable ring signatures and ring signatures of the ith anonymity set. We denote the $f(\lambda)$ MProve proofs as $\mathsf{MPe}_{\mathsf{act}} = \left(\mathbf{P}^{(i)}, \mathbf{C}^{(i)}, \mathbf{C'}^{(i)}, \mathbf{\Gamma}^{(i)}, \mathbf{\Sigma}^{(i)}, m_i, C_{\text{res}}^{(i)}\right)_{i=1}^{f(\lambda)}$. We want to show that $\mathsf{MPe}_{\mathsf{act}}$ does not reveal any information of the exchange apart from the fact that the commitments to the total reserves are valid. For this, we construct a simulator $\mathcal{S}_{\mathsf{MPe}}$ which has access to $\mathsf{MPe}_{\mathsf{act}}$ but does not know any secret information of the exchange. The simulator proceeds as follows.

She chooses a long term key $k_{\text{sim}} \xleftarrow{\$} \mathbb{Z}_q$. To construct the ith simulated MProve proof, she reads the ith MProve proof $\left(\mathbf{P}^{(i)}, \mathbf{C}^{(i)}, \mathbf{C'}^{(i)}, \mathbf{\Gamma}^{(i)}, \mathbf{\Sigma}^{(i)}, m_i, C_{\text{res}}^{(i)}\right)$. In the ith simulated proof, she keeps $\left(\mathbf{P}^{(i)}, \mathbf{C}^{(i)}, m_i\right)$ as it is and changes the other elements in the following manner. She sets $\hat{C'}_{i,k} = C_{i,k} + y_{i,k}G$, where $y_{i,k} = H_s(k_{\text{sim}}, P_{i,k})$ for all $k \in \{1, 2, \ldots, n_i\}$. She computes the signatures $\hat{\gamma}_{i,k}$ and $\hat{\sigma}_{i,k}$ in the simulated proof using the knowledge of $y_{i,k}$ for all $k \in \{1, 2, \ldots, n_i\}$. In particular, the set of public keys in the signatures $\hat{\gamma}_{i,k}$ and $\hat{\sigma}_{i,k}$ are $(\hat{C'}_{i,k}, \hat{C'}_{i,k} - C_{i,k})$ and $(P_{i,k}, \hat{C'}_{i,k} - C_{i,k})$ respectively. Each of them are signed with $y_{i,k}$ where $\hat{C'}_{i,k} - C_{i,k} = y_{i,k}G$. She constructs the vectors $\hat{\mathbf{C}}'^{(i)}, \hat{\mathbf{\Gamma}}^{(i)}, \hat{\mathbf{\Sigma}}^{(i)}$ with the computed quantities $\hat{C'}_{i,k}$, $\hat{\gamma}_{i,k}$, and $\hat{\sigma}_{i,k}$ respectively for all $k \in \{1, 2, \ldots, n_i\}$. She computes $\hat{C}_{\text{res}}^{(i)}$ as,

$$\hat{C}_{\text{res}}^{(i)} = \sum_{j=1}^{n_i} C_{i,j} - \sum_{j=1}^{n_i} \hat{C'}_{i,j}. \tag{2.43}$$

She sets $\mathsf{MPe}_{\mathsf{sim}} = \left(\mathbf{P}^{(i)}, \mathbf{C}^{(i)}, \hat{\mathbf{C}}'^{(i)}, \hat{\mathbf{\Gamma}}^{(i)}, \hat{\mathbf{\Sigma}}^{(i)}, m_i, \hat{C}_{\text{res}}^{(i)}\right)_{i=1}^{f(\lambda)}$.

If there exists no PPT distinguisher $\mathscr{D}_{\mathsf{MPe}}$ that can distinguish between $\mathsf{MPe}_{\mathsf{act}}$ and $\mathsf{MPe}_{\mathsf{sim}}$ (except with a negligible probability of success), then we can say $\mathsf{MPe}_{\mathsf{act}}$ does not reveal any information about the exchange. In particular, we define the privacy experiment `MProvePriv` for the MProve scheme as follows.

1. $\mathcal{S}_{\mathsf{MPe}}$ sets $\mathsf{MPe}_0 = \mathsf{MPe}_{\mathsf{sim}}$ and $\mathsf{MPe}_1 = \mathsf{MPe}_{\mathsf{act}}$.
2. $\mathcal{S}_{\mathsf{MPe}}$ chooses a bit $b \xleftarrow{\$} \{0, 1\}$ randomly.
3. $\mathcal{S}_{\mathsf{MPe}}$ sends MPe_b to $\mathscr{D}_{\mathsf{MPe}}$.
4. $\mathscr{D}_{\mathsf{MPe}}$ outputs a bit $\mathscr{D}_{\mathsf{MPe}}(\mathsf{MPe}_b)$ as a prediction of b.

Notice that $\Pr[\mathscr{D}_{\mathsf{MPe}}(\mathsf{MPe}_b) = b] = \frac{1}{2}$ can be obtained by random guessing. We give the following definition.

Definition 2.1. *The MProve protocol is said to provide pre-spend privacy if for every PPT $\mathscr{D}_{\mathsf{MPe}}$ in the `MProvePriv` experiment, there exists a negligible function $\mathsf{negl}(\lambda)$ of the security parameter λ such that,*

$$\left| \Pr[\mathscr{D}_{\mathsf{MPe}}(\mathsf{MPe}_b) = b] - \frac{1}{2} \right| \leq \mathsf{negl}(\lambda). \tag{2.44}$$

Here we are considering that the exchange has not spent from any source addresses after publishing the reserves proofs.

Notice that if the exchange spends from any of the source addresses, then it becomes easy to distinguish between MPe_1 and MPe_0. This is because the proofs in MPe_1 will contain one or more key images (corresponding to the source addresses) which have appeared in the blockchain (Section 2.5). On the other hand, the proofs in MPe_0 will have no such key images. However this is not the case when the cover addresses in the proofs are spent. The key images of cover addresses match neither those in MPe_1 nor those in MPe_0. We have the following theorem.

Theorem 2.3. *The MProve protocol provides pre-spend privacy in the random oracle model under the DDH assumption.*

The proof of Theorem 2.3 uses a hybrid argument [37, Section 11.2.2] and is given in Appendix A.5.

Pre-spend Privacy for MProvisions

Next we consider the scenario when an exchange has published $f(\lambda)$ MProvisions proofs where $f(\lambda)$ is a polynomial of the security parameter λ. We again assume that the exchange has not spent from any of the one-time addresses used as sources in those published proofs. We denote the ith MProvisions proof as

$$\mathsf{MPs}_{\mathsf{act}i} = (P_{i,j}, C_{i,j}, B_{i,j}, I_{i,j}, M_{i,j}, N_{i,j}, \mathsf{Tx}_{i,j})_{j=1}^{n_i},$$

where $\mathsf{Tx}_{i,j} = \left(A_{i,j}^{(1)}, A_{i,j}^{(2)}, A_{i,j}^{(3)}, A_{i,j}^{(4)}, A_{i,j}^{(5)}, c_{i,j}, r_{s_{i,j}}, r_{k_{i,j}}, r_{e_{i,j}}, r_{f_{i,j}}, r_{\hat{x}_{i,j}}\right)$ is the transcript and n_i is the size of the ith anonymity set. We denote the $f(\lambda)$ MProvisions proofs by $\mathsf{MPs}_{\mathsf{act}} = (\mathsf{MPs}_{\mathsf{act}i})_{i=1}^{f(\lambda)}$. Similar to MProve, we consider a PPT simulator $\mathcal{S}_{\mathsf{MPs}}$ and a PPT distinguisher $\mathcal{D}_{\mathsf{MPs}}$. The simulator $\mathcal{S}_{\mathsf{MPs}}$ has access to $\mathsf{MPs}_{\mathsf{act}}$. To construct the simulated proof, the simulator $\mathcal{S}_{\mathsf{MPs}}$ proceeds as follows.

She reads the ith MProvisions proof $\mathsf{MPs}_{\mathsf{act}i}$. In $\mathsf{MPs}_{\mathsf{act}i}$, she keeps $(P_{i,j}, C_{i,j})_{j=1}^{n_i}$ as it is and changes the other elements in the following manner. She chooses a long term key $k_{\text{sim}} \xleftarrow{\$} \mathbb{Z}_q$. She computes a random scalar $x''_{i,j} = H_s(k_{\text{sim}}, P_{i,j})$ and sets $\hat{I}_{i,j} = x''_{i,j} H_p(P_{i,j})$ for all $j \in \{1, 2, \ldots, n_i\}$. She chooses $\hat{k}_{i,j}, \hat{e}_{i,j}, \hat{f}_{i,j} \xleftarrow{\$} \mathbb{Z}_q$ and sets $\hat{B}_{i,j} = \hat{k}_{i,j} G, \hat{M}_{i,j} = \hat{e}_{i,j} H, \hat{N}_{i,j} = \hat{f}_{i,j} H$. She computes the ith simulated MProvisions proof as $\mathsf{MPs}_{\mathsf{sim}i} = \left(P_{i,j}, C_{i,j}, \hat{B}_{i,j}, \hat{I}_{i,j}, \hat{M}_{i,j}, \hat{N}_{i,j}, \hat{\mathsf{Tx}}_{i,j}\right)_{j=1}^{n_i}$ following the MProvisions protocol, where $\hat{\mathsf{Tx}}_{i,j} = (\hat{A}_{i,j}^{(1)}, \hat{A}_{i,j}^{(2)}, \hat{A}_{i,j}^{(3)}, \hat{A}_{i,j}^{(4)}, \hat{A}_{i,j}^{(5)}, \hat{c}_{i,j}, \hat{r}_{s_{i,j}}, \hat{r}_{k_{i,j}}, \hat{r}_{e_{i,j}}, \hat{r}_{f_{i,j}}, \hat{r}_{\hat{x}_{i,j}})$ is the transcript. She

performs the same operation for all $i \in \{1, 2, \ldots, f(\lambda)\}$ and sets the $f(\lambda)$ simulated proof as $\mathsf{MPs}_{\mathsf{sim}} = (\mathsf{MPs}_{\mathsf{sim}_i})_{i=1}^{f(\lambda)}$.

We define the privacy experiment `MProvisionsPriv` for the MProvisions scheme as follows.

1. $\mathcal{S}_{\mathsf{MPs}}$ sets $\mathsf{MPs}_0 = \mathsf{MPs}_{\mathsf{sim}}$ and $\mathsf{MPs}_1 = \mathsf{MPs}_{\mathsf{act}}$.
2. $\mathcal{S}_{\mathsf{MPs}}$ chooses a bit $b \xleftarrow{\$} \{0, 1\}$ randomly.
3. $\mathcal{S}_{\mathsf{MPs}}$ sends MPs_b to $\mathcal{D}_{\mathsf{MPs}}$.
4. $\mathcal{D}_{\mathsf{MPs}}$ outputs a bit $\mathcal{D}_{\mathsf{MPs}}(\mathsf{MPs}_b)$ as a prediction of b.

Now we give the following definition.

Definition 2.2. *The MProvisions protocol is said to provide privacy if for every PPT $\mathcal{D}_{MPs}$ in the `MProvisionsPriv` experiment, there exists a negligible function $\mathsf{negl}(\lambda)$ of the security parameter λ such that,*

$$\left| \Pr[\mathcal{D}_{MPs}(MPs_b) = b] - \frac{1}{2} \right| \leq \mathsf{negl}(\lambda). \tag{2.45}$$

Here we are considering that the exchange has not spent from any source addresses after publishing the reserves proofs.

The intuition behind Definition 2.2 is similar to that of Definition 2.1, i.e. MPs_0 is constructed without the knowledge of which addresses in the anonymity sets are exchange-owned, their secret keys or the amounts. If no PPT distinguisher can distinguish between MPs_0 and MPs_1, the real MProvisions proofs, then it is verified that the MProvisions protocol preserves the privacy of the exchange. Like MProve, if the exchange spends from some of the source addresses then it is trivial to distinguish between MPs_1 and MPs_0 by matching the key images of the source addresses (Section 2.5). However, this is not the case if the cover addresses are spent by their owners. This is because the key images of the cover addresses match neither the dummy key images in MPs_1 (generated from $H_s(k_{\text{exch}}, P_{i,j})$) nor the dummy key images in MPs_0 (generated from $H_s(k_{\text{sim}}, P_{i,j})$). Now we give the following theorem.

Theorem 2.4. *The MProvisions protocol provides privacy in the random oracle model under the DDH assumption.*

The proof of Theorem 2.4 is given in Appendix A.6.

2.7 Effect of Proposed Protocols on Monero Privacy

The major privacy properties for the Monero scheme are amount confidentiality, unlinkability, and untraceability. We described these properties in Section 2.2. A privacy focused proof of reserves protocol should not violate the privacy features of the underlying cryptocurrency scheme. Below we discuss how the proposed protocols MProve and MProvisions affect the privacy features of Monero.

2.7.1 Effect on the Untraceability Property of Monero

As discussed in Section 2.2.2, a Monero transaction is called untraceable when a PPT adversary cannot determine which one-time address in the ring is actually being spent i.e. originates the associated key image. Here by saying that the one-time address P originates the key image I, we mean that there exists a scalar $x \in \mathbb{Z}_q$ such that $P = xG \wedge I = xH_p(P)$ holds. As discussed in Section 2.5, a Monero transaction becomes traceable when the exchange spends from a one-time address which has already been used as a source address in an MProve or MProvisions proof.

2.7.2 Effect on the Amount Confidentiality Property of Monero

From Theorems 2.3 and 2.4, it is verified that the MProve and MProvisions protocols do not reveal the amount corresponding to any address in the anonymity set or the total reserves amount. However, the amount confidentiality of Monero is affected by exchange-owned addresses that become traceable when they are spent in a Monero transaction. Suppose an exchange-owned address P becomes traceable after being spent in a transaction *txn*. Any other Monero transaction *txn*$'$ which has included P in its ring will have its effective ring size reduced by one. This in turn reveals information about the amounts in the outputs created in *txn*$'$. To make the discussion concrete, suppose the transaction *txn*$'$ spends from a single input using the ring of m one-time addresses $(P, P_1, P_2, \ldots, P_{m-1})$ with corresponding Pedersen commitments $(C, C_1, C_2, \ldots, C_{m-1})$. For simplicity, suppose *txn*$'$ has a single output represented by a one-time address P' and Pedersen commitment C'. Before it was revealed that P was spent in transaction *txn*, the upper bound on the amount in the commitment C' is given by the maximum of the amounts in the commitments $(C, C_1, C_2, \ldots, C_{m-1})$ minus the transaction fees. This is because any of the ring

members could have been the true source of funds in the transaction. Once P has been revealed as spent in txn, it cannot be the source of funds in txn'. Thus the upper bound on the amount in the commitment C' is given by the maximum of the amounts in the commitments $(C_1, C_2, \ldots, C_{m-1})$. As this list of commitments is smaller, the upper bound can only be more restrictive.

One might argue that the amounts in the Pedersen commitments are not known. But the amounts in coinbase commitments in Monero are revealed to ensure that miners are creating valid blocks. While the amounts in non-coinbase commitments are not known exactly, they can be upper bounded by identifying the coinbase commitments which could have potentially contributed funds to them. Such an analysis has been demonstrated for the Pedersen commitments in Grin [38].

To conclude, the MProve and MProvisions reveal information about the amounts in transactions where exchange-owned addresses appear as mixins. Therefore both the schemes do not preserve the amount confidentiality of Monero.

2.7.3 Effect on the Unlinkability Property of Monero

The unlinkability property of Monero implies that a PPT adversary can link a one-time address with its corresponding public key pair only with a negligible probability (Section 2.2.1). Let $\{\mathcal{M}_\lambda\}$ denote a sequence of Monero-like systems indexed by the security parameter λ. We consider one such particular system $\mathcal{M}_\lambda$ from the sequence and define the following `MoneroLink` experiment to precisely characterize the unlinkability property of Monero.

1. An experimenter chooses some scalars $x_0, y_0, x_1, y_1, r \xleftarrow{\$} \mathbb{Z}_q$. She sets two public key pairs $(X_0 = x_0G, Y_0 = y_0G), (X_1 = x_1G, Y_1 = y_1G)$, and a random point $R = rG$.

2. The experimenter selects a bit $b \xleftarrow{\$} \{0, 1\}$. Then she generates a one-time address $P = H(rX_b)G + Y_b$.

3. The experimenter sends $(X_0, Y_0, X_1, Y_1, R, P)$ to a PPT adversary $\mathcal{A}$. The adversary $\mathcal{A}$ outputs $\hat{b}$ as a prediction of b. $\mathcal{A}$ wins if $\hat{b} = b$.

Owing to the unlinkability property of Monero, we have the following lemma.

Lemma 2.1. *For every PPT adversary $\mathcal{A}$ in the* `MoneroLink` *experiment, there exists a negligible function $\mathsf{negl}(\lambda)$ of the security parameter λ such that the following inequality holds.*

$$\left| \Pr[\mathcal{A}(X_0, Y_0, X_1, Y_1, R, P) = b] - \frac{1}{2} \right| \leq \mathsf{negl}(\lambda). \tag{2.46}$$

Next, we propose the following `MProveLink` experiment to show that MProve preserves the unlinkability property of Monero.

1. An experimenter chooses some scalars $x_0, y_0, x_1, y_1, r \xleftarrow{\$} \mathbb{Z}_q$. She sets two public key pairs $(X_0 = x_0G, Y_0 = y_0G), (X_1 = x_1G, Y_1 = y_1G)$ and a random point $R = rG$.

2. The experimenter selects a bit $\mathfrak{b} \xleftarrow{\$} \{0, 1\}$. Then she generates a one-time address $P = H(rX_b)G + Y_b$. The secret key is $x = H(rX_b) + y_b$.

3. The experimenter produces $f(\lambda)$ MProve proofs $\mathsf{MPe}_{\mathsf{act}}$ using the singleton set $\{P\}$ as the anonymity set in all of them. The linkable ring signatures contain the key image $I = xH_p(P)$.

4. The experimenter sends $(X_0, Y_0, X_1, Y_1, R, P, \mathsf{MPe}_{\mathsf{act}})$ to a PPT adversary $\mathcal{B}$. $\mathcal{B}$ outputs $\hat{\mathfrak{b}}$ as a prediction of $\mathfrak{b}$. $\mathcal{B}$ wins if $\hat{\mathfrak{b}} = \mathfrak{b}$.

Now we give the following definition.

Definition 2.3. *The MProve protocol is said to preserve the unlinkability property of Monero, if for every PPT adversary $\mathcal{B}$ in the* `MProveLink` *experiment, there exists a negligible function $\mathsf{negl}_1(\lambda)$ of the security parameter λ such that the following inequality holds.*

$$\left| \Pr[\mathcal{B}(X_0, Y_0, X_1, Y_1, R, P, \mathit{MPe}_{act}) = b] - \frac{1}{2} \right| \leq \mathsf{negl}_1(\lambda). \tag{2.47}$$

We give the following theorem and its proof.

Theorem 2.5. *The MProve protocol preserves the unlinkability property of Monero in the random oracle model under the DDH assumption and given that the Lemma 3.1 holds.*

Proof. We prove the theorem by contradiction. Suppose, there exists a PPT adversary $\mathcal{B}$ in the `MProveLink` experiment for which the following inequality holds,

$$\left| \Pr[\mathcal{B}(X_0, Y_0, X_1, Y_1, R, P, \mathsf{MPe}_{\mathsf{act}}) = \mathfrak{b}] - \frac{1}{2} \right| \geq \frac{1}{p(\lambda)}, \tag{2.48}$$

where $p(\lambda)$ is a polynomial of the security parameter λ. From Theorem 2.3, no PPT adversary can distinguish between $\mathsf{MPe}_{\mathsf{act}}$ and $\mathsf{MPe}_{\mathsf{sim}}$ with a probability non-negligibly better than $\frac{1}{2}$. So for the adversary $\mathcal{B}$, there exists another PPT adversary $\mathfrak{B}$ such that the following inequality holds,

$$\left| \Pr[\mathfrak{B}(X_0, Y_0, X_1, Y_1, R, P, \mathsf{MPe}_{\mathsf{sim}}) = \mathfrak{b}] - \frac{1}{2} \right| \geq \frac{1}{p(\lambda)}, \tag{2.49}$$

given that inequality (2.48) is true for $\mathcal{B}$. Next, we construct a PPT adversary $\mathcal{A}$ for the `MoneroLink` experiment using $\mathfrak{B}$ as a subroutine. The construction of $\mathcal{A}(X_0, Y_0, X_1, Y_1, R, P)$ is given below.

1. $\mathcal{A}$ generates $f(\lambda)$ simulated MProve proofs $\mathsf{MPe}_{\mathsf{sim}}$ using the singleton set $\{P\}$ as the anonymity set in all of them, following the same steps of the simulator $\mathcal{S}_{\mathsf{MPe}}$ given in Section 2.6.3.

2. $\mathcal{A}$ sends $(X_0, Y_0, X_1, Y_1, R, P, \mathsf{MPe}_{\mathsf{sim}})$ to $\mathfrak{B}$ and receives $\hat{\mathfrak{b}}$.

3. It outputs $\hat{\mathfrak{b}}$ as the estimation of b.

Now we have,

$$\begin{aligned} &\left| \Pr[\mathcal{A}(X_0, Y_0, X_1, Y_1, R, P) = b] - \frac{1}{2} \right| \\ &= \left| \Pr[\mathfrak{B}(X_0, Y_0, X_1, Y_1, R, P, \mathsf{MPe}_{\mathsf{sim}}) = b] - \frac{1}{2} \right| \\ &\geq \frac{1}{p(\lambda)}. \end{aligned} \tag{2.50}$$

This contradicts Lemma 3.1. Hence there cannot be a PPT adversary $\mathcal{B}$ for which the inequality (2.48) holds. □

The statement that the MProvisions protocol preserves the unlinkability property of Monero can be proved by a similar analysis as above.

2.8 Implementation and Performance

An MProve prover needs to compute $11n$ group exponentiations and $\mathcal{O}(n)$ point additions and field operations. An MProve verifier needs to compute $12n$ group exponentiations and $\mathcal{O}(n)$ point additions and field operations. The proof size is $(3n + 2)$ group elements and $6n$ scalars.

Whereas, an An MProvisions prover needs to compute $15n$ group exponentiations and $\mathcal{O}(n)$ point additions and field operations. An MProvisions verifier needs to compute $19n$ group exponentiations and $\mathcal{O}(n)$ point additions and field operations. The proof size is $6n$ group elements and $9n$ scalars.

The generation and verification (except for the key image blockchain query) algorithms for both MProve and MProvisions were implemented in C++ as tests in the Monero codebase [39, 40]. The implementation of MProvisions is similar to Provisions [1] and is based on non-interactive Schnorr signatures.

The simulation performance of MProve and MProvisions is given in Table 2.1 for anonymity set $\mathscr{P}_{\text{anon}}$ having sizes 1000, 10000, and 100000. The program to check whether the key images in the proof appear on the blockchain was implemented as a Python script which sends an RPC call to the Monero daemon (script available at [39]). The query time column in Table 2.1 gives the execution times of this script after the LMDB database was fully loaded into RAM (it consumes about 2 GB). The execution times were measured on a 3.6 GHz CPU/8 GB RAM desktop PC. For each case, the percentage of known addresses is either 10%, 50%, or 90%. Both generation/verification times and proof sizes increase linearly with the size of $\mathscr{P}_{\text{anon}}$, with the proof generation time having a small dependence on the size of $\mathscr{P}_{\text{own}}$. From Table 2.1, we observe that the proof sizes for the MProvisions protocol are about 1.5 times that of the MProve protocol. Also the generation and verification times of MProvisions are about 2.5 times that of MProve. We, therefore, infer that the MProve protocol is better than the MProvisions protocol in every respect.

2.9 Conclusion

In this chapter, we describe two proof of reserves protocols for Monero. Both of them are first to provide some privacy to a Monero exchange. They also prevent the exchanges from colluding with each other while generating reserves proofs. By describing MProvisions, we have explained how the proof of reserves protocol in Provisions [1] can be modified for Monero. Then we described MProve which outperforms MProvisions. MProve's proof generation and verification algorithms are efficient taking only about a minute to complete for an anonymity set having 100,000 addresses. However, we foresee two areas for

Table 2.1: MProve and MProvisions Proof Generation and Verification Performance.

$\lvert\mathscr{P}_{\text{anon}}\rvert$	$\lvert\mathscr{P}_{\text{own}}\rvert$	MProve Proof Size	MProve Generat. Time	MProve Verif. Time	MProvisions Proof Size	MProvisions Generat. Time	MProvisions Verif. Time	Query Time
100	100	0.32 MB	1.15 s	1.07 s	0.48 MB	2.56 s	2.85 s	0.048 s
1000	500	0.32 MB	1.14 s	1.07 s	0.48 MB	2.70 s	2.84 s	0.048 s
1000	900	0.32 MB	1.16 s	1.11 s	0.48 MB	2.82 s	2.85 s	0.048 s
10000	1000	3.2 MB	11.54 s	10.74 s	4.8 MB	25.93 s	28.78 s	0.087 s
10000	5000	3.2 MB	11.43 s	10.72 s	4.8 MB	27.36 s	28.76 s	0.087 s
10000	9000	3.2 MB	11.28 s	10.73 s	4.8 MB	28.53 s	28.71 s	0.087 s
100000	10000	32 MB	116.22 s	108.15 s	48 MB	258.78 s	287.57 s	0.545 s
100000	50000	32 MB	114.97 s	108.02 s	48 MB	271.37 s	301.55 s	0.545 s
100000	90000	32 MB	114.43 s	108.08 s	48 MB	284.60 s	286.96 s	0.545 s

improvement.

- For both the protocols, a source spending transaction becomes traceable. Preventing this will provide better privacy to the exchange as well as the entire Monero network.
- The proof sizes scale linearly with the size of the anonymity set $\mathscr{P}_{\text{anon}}$. Reducing the proof size is important especially if the exchanges are asked to publish the reserves proofs frequently and store them for later audits.

We address both the aspects mentioned above in the next chapter.

Chapter 3

MProve+: Privacy Enhancing Proof of Reserves Protocol for Monero

In the last chapter, we described MProvisions and MProve. We saw that when an exchange spends from a one-time address which has already been used as a source address in an MProve or MProvisions proof, the spending transaction becomes traceable (Section 2.5). In particular, the source of the transaction and its linkage to the key image are revealed. A zero-mixin transaction in Monero is a transaction which does not have any decoy addresses in the ring. For MProve/MProvisions, a transaction spending from a source address is effectively a zero-mixin transaction as the other decoy addresses in the ring fail to obfuscate the source of the transaction. This can lead to traceability in other transactions via the cascade effect [31, 32]. This is a significant privacy limitation since it not only affects the exchange's privacy but also affects the privacy of other Monero users. Another issue with the MProvisions/MProve protocols is that the proof size scales linearly with the size of the anonymity set.

To address the above concerns, we propose MProve+, a privacy enhancing proof of reserves scheme for Monero*. The MProve+ protocol is motivated from Bulletproofs [17] and Omniring [41]. In particular, the MProve+ protocol is constructed by modifying a transaction scheme for Monero given in the Omniring paper [22]. The MProve+ scheme solves the drawback of the MProve scheme and makes the proof size logarithmic in terms of the anonymity set size. After describing the protocol, we discuss its security properties. To compare the performance of MProve+ to MProve, we have implemented both of them

*MProve+ is based on joint work with Suyash Bagad.

in Rust.

3.1 Background

In this section, we present notation and the preliminary concepts required to describe the MProve+ protocol.

3.1.1 Notation

We consider the same cyclic group $\mathbb{G}$ of prime order q with generator G used in Monero where the decisional Diffie Hellman (DDH) problem is assumed to be hard. In this chapter, we have followed multiplicative notation for group operation to be consistent with the Omniring [22] paper. All group elements are denoted by upper case letters. All scalars in $\mathbb{Z}_q$ are denoted by lower case letters. As $\mathbb{G}$ is of prime order, every non-identity element of $\mathbb{G}$ is a generator. Let $H \in \mathbb{G}$ be another random generator of $\mathbb{G}$ such that the discrete logarithm relation between G and H is not known i.e. x is not known where $H = G^x$. A Pedersen commitment [16] C to an amount a is defined as $G^y H^a$, where $y \in \mathbb{Z}_q$ is a randomly sampled blinding factor.

Let $\mathbb{G}^n$ and $\mathbb{Z}_q^n$ be the n-ary Cartesian products of sets $\mathbb{G}$ and $\mathbb{Z}_q$ respectively. Bold fonts denote vectors. Inner product of two scalar vectors $\mathbf{a}, \mathbf{b} \in \mathbb{Z}_q^n$ is defined as $\langle \mathbf{a}, \mathbf{b} \rangle \coloneqq \sum_{i=1}^{n} a_i \cdot b_i$ where $\mathbf{a} = (a_1, \ldots, a_n), \mathbf{b} = (b_1, \ldots, b_n)$. Hadamard and Kronecker products are defined respectively as, $\mathbf{a} \circ \mathbf{b} \coloneqq (a_1 \cdot b_1, \ldots, a_n \cdot b_n) \in \mathbb{Z}_q^n$, $\mathbf{a} \otimes \mathbf{c} \coloneqq (a_1\mathbf{c}, \ldots, a_n\mathbf{c}) \in \mathbb{Z}_q^{nm}$ where $\mathbf{c} \in \mathbb{Z}_q^m$. The concatenation of vectors $\mathbf{a}$ and $\mathbf{b}$ is denoted as $\mathbf{a} \| \mathbf{b} \coloneqq (a_1, \ldots, a_n, b_1, \ldots, b_n)$. For a base vector $\mathbf{G} = (G_1, \ldots, G_n) \in \mathbb{G}^n$, vector exponentiation is defined as $\mathbf{G}^{\mathbf{a}} \coloneqq \prod_{i=1}^{n} G_i^{a_i} \in \mathbb{G}$. For a scalar $u \in \mathbb{Z}_q \setminus \{0\}$, we denote its consecutive powers in the form of a vector $\boldsymbol{u}^n \coloneqq (1, u, u^2, \ldots, u^{n-1})$. We represent exponentiation of all components of a vector $\mathbf{a}$ by the same scalar $k \in \mathbb{Z}_q$ by $\mathbf{a}^{\circ k} \coloneqq (a_1^k, a_2^k, \ldots, a_n^k)$. Hadamard inverse of a vector is defined as $\mathbf{a}^{\circ -1} \coloneqq (b_1, b_2, \ldots, b_n)$ where $b_i = a_i^{-1}$ if $a_i \neq 0$ and $b_i = 1$ otherwise. If an element a is chosen uniformly from a set A, such a choice is denoted by $a \xleftarrow{\$} A$. For a positive integer N, $[N]$ denotes the set $\{1, 2, \ldots, N\}$.

3.1.2 Drawback of MProve

In Section 2.2 and 2.4.2, we have discussed various features of Monero and MProve respectively. In this subsection, we recall the drawback of the MProve scheme discussed in Section 2.5 to make this chapter self-contained. Suppose a Monero exchange Ex uses a owned one-time address P_j to generate an MProve proof. Then Ex has to publish the key image of P_j i.e. I_j in the proof as a part of the linkable ring signature σ_j. Suppose at a later point of time, Ex creates a transaction txn to spend from P_j. In txn, Ex forms the ring of the linkable ring signature containing P_j and some other cover one-time addresses to obfuscate the source of txn. However in the linkable ring signature of txn, I_j appears again. When txn appears in the blockchain, an adversary can match I_j as a key image of txn and a part of the MProve proof published by Ex. Essentially she gains the following information.

1. As I_j appearing in txn has already appeared in an Ex generated MProve proof, Ex is spending in txn.

2. As P_j is the only common one-time address in the ring of the linkable ring signature σ_j and the ring of one-time addresses used in txn, P_j is owned by Ex and it has I_j as its key image.

3. P_j is the source of the transaction txn.

All the three statements affect the privacy of the exchange. However, the statements 2 and 3 are more crucial towards the privacy of the exchange as well as the entire Monero network because of the following reason. P_j is revealed as exchange-owned and txn effectively becomes a zero-mixin transaction. This increases traceability of transactions in the Monero blockchain [31, 32]. To avoid the cascade effect, P_j should be pruned from the set of UTXOs. Even if this is done, the ring sizes reduce by 1 for all the transaction which have used P_j as a cover address so far. The main reason for this drawback is the association of I_j with P_j through σ_j. Note that a similar association also occurs in the MProvisions protocol. The MProve+ scheme breaks this association using techniques from Bulletproofs [17] and Omniring [22] which are discussed next.

3.1.3 Bulletproofs and Omniring

The current Monero implementation suffers from the fact that the linkable ring signature size scales linearly with the size of the ring. This is crucial because these signatures are part of the transaction stored in the blockchain. As a consequence, it is expensive to use a large ring size (higher transaction size costs more transaction fees). Omniring [22] proposes a technique where the proof of validity of the transaction is logarithmic in the size of the ring. Omniring is motivated from Bulletproofs [17] and does not require any trusted setup. Currently, for Monero transactions with multiple sources, a separate ring is chosen for each source one-time address. Omniring proposes to use a single large ring for all source one-time addresses of a transaction, hence the name.

Bulletproofs [17] gives a state-of-the-art range proof system with logarithmic proof size. Here, given a Pedersen commitment[†] $C = G^v H^\gamma$, a prover can prove that $v \in \{0, 1, \ldots, N-1\}$ for some $N = 2^n \in \mathbb{Z}_q$ without revealing v. Currently, Bulletproofs are used in a Monero transaction to prove that all the output amounts in a transaction are in the right range. In the following, we discuss some aspects of Bulletproofs and Omniring that are relevant to us.

Range Proof Using Bulletproofs

In a range proof, a prover needs to prove that $v \in \{0, 1, \ldots, N-1\}$ for some $N = 2^n \in \mathbb{Z}_q$ where the verifier only knows C which is equal to $G^v H^\gamma$. To do so, v is represented in binary bits (say by binary vector $\mathbf{a}_L \in \mathbb{Z}_2^n$). The complement vector of $\mathbf{a}_L$, i.e. vector $\mathbf{1}^n - \mathbf{a}_L$, is denoted by $\mathbf{a}_R$. The condition $v \in \{0, 1, \ldots, N-1\}$ is then equivalently represented by following three constraint equations which use $\mathbf{a}_L$ and $\mathbf{a}_R$.

$$\langle \mathbf{a}_L, \mathbf{2}^n \rangle = v \tag{3.1}$$

$$\langle \mathbf{a}_L, \mathbf{a}_R \circ \mathbf{y}^n \rangle = 0 \tag{3.2}$$

$$\langle \mathbf{a}_L - \mathbf{1}^n - \mathbf{a}_R, \mathbf{y}^n \rangle = 0, \tag{3.3}$$

where the vector $\mathbf{y}^n = (1, y, y^2, \ldots, y^{n-1})$ is constructed using the consecutive powers of $y \xleftarrow{\$} \mathbb{Z}_q$, a random challenge sent by the verifier. Here equation (3.1) ensures that $\mathbf{a}_L$ is

[†]In the Monero literature, the amount is placed in the exponent of H and the blinding factor is placed in the exponent of G. In case of Bulletproofs [17], it is the opposite. However, this is just a difference in notation.

the binary representation of v, equation (3.2) ensures that the component-wise product of $\mathbf{a}_L$ with $\mathbf{a}_R$ is always a zero vector, and equation (3.3) ensures that $\mathbf{a}_R$ is obtained by subtracting the elements of $\mathbf{a}_L$ from $\mathbf{1}^n$ vector. Both equations (3.2) and (3.3) ensure that the elements of $\mathbf{a}_L$ are either 0 or 1. Here the idea is that if a polynomial evaluates to zero at a random evaluation point chosen from a large set, then with high probability, the polynomial is a zero polynomial. These constraint equations are multiplied with powers of another random challenge $z \xleftarrow{\$} \mathbb{Z}_q$ sent by the verifier and added to form a single inner product as follows.

$$\langle \mathbf{a}_L - z{\cdot}\mathbf{1}^n, \mathbf{y}^n \circ (\mathbf{a}_R + z{\cdot}\mathbf{1}^n) + z^2{\cdot}\mathbf{2}^n \rangle = z^2{\cdot}v + \delta(y, z), \tag{3.4}$$

where $\delta(y, z)$ is a function of y, z and can be calculated by the verifier. Bulletproofs proposes an optimized inner product proof with logarithmic proof size. However this inner product proof is not zero-knowledge. As $\mathbf{a}_L$, $\mathbf{a}_R$ are secret quantities, this inner product proof cannot be applied directly to prove equation (3.4). Thus the prover chooses two blinding vectors $\mathbf{s}_L, \mathbf{s}_R \xleftarrow{\$} \mathbb{Z}_q^n$ and computes the following polynomials and their inner product.

$$\begin{aligned} l(X) &= \mathbf{a}_L - z{\cdot}\mathbf{1}^n + \mathbf{s}_L{\cdot}X && \in \mathbb{Z}_q^n[X] \\ r(X) &= \mathbf{y}^n \circ (\mathbf{a}_R + z{\cdot}\mathbf{1}^n + \mathbf{s}_R{\cdot}X) + z^2{\cdot}\mathbf{2}^n && \in \mathbb{Z}_q^n[X] \\ t(X) &= \langle l(X), r(X) \rangle = t_0 + t_1{\cdot}X + t_2{\cdot}X^2 && \in \mathbb{Z}_q[X], \end{aligned}$$

where $t_0 = z^2{\cdot}v + \delta(y, z)$. Then the prover and the verifier engage in an interactive protocol. The prover sends a commitment to $\mathbf{a}_L$, $\mathbf{a}_R$ as $A = H^{\alpha}\mathbf{G}^{\mathbf{a}_L}\mathbf{H}^{\mathbf{a}_R}$, a commitment to $\mathbf{s}_L$, $\mathbf{s}_R$ as $S = H^{\rho}\mathbf{G}^{\mathbf{s}_L}\mathbf{H}^{\mathbf{s}_R}$, and commitments to t_1 and t_2 as $T_1 = G^{t_1}H^{\tau_1}$, $T_2 = G^{t_2}H^{\tau_2}$ to the verifier where $\alpha, \rho, \tau_1, \tau_2 \xleftarrow{\$} \mathbb{Z}_q$ are random scalars and $\mathbf{G}, \mathbf{H} \xleftarrow{\$} \mathbb{G}^n$ are random base vectors. The verifier sends a random evaluation point $x \xleftarrow{\$} \mathbb{Z}_q$ to the prover. Prover then evaluates $\mathbf{l} = l(x)$, $\mathbf{r} = r(x)$, and $\hat{t} = \langle \mathbf{l}, \mathbf{r} \rangle$. Because of blinding vectors $\mathbf{s}_L$ and $\mathbf{s}_R$, the prover can use $\mathbf{l}$, $\mathbf{r}$ in the inner product proof to prove that $\hat{t} = \langle \mathbf{l}, \mathbf{r} \rangle$, without revealing $\mathbf{a}_L$ and $\mathbf{a}_R$. Using C, A, S, T_1, T_2, $\mathbf{l}$, $\mathbf{r}$, $\hat{t}$, and other quantities sent by the prover, the verifier verifies the following conditions.

i. $\hat{t} \stackrel{?}{=} t_0 + t_1 x + t_2 x^2$.

ii. $\mathbf{l} \stackrel{?}{=} \mathbf{a}_L - z{\cdot}\mathbf{1}^n + \mathbf{s}_L{\cdot}x$ and $\mathbf{r} \stackrel{?}{=} \mathbf{y}^n \circ (\mathbf{a}_R + z{\cdot}\mathbf{1}^n + \mathbf{s}_R{\cdot}x) + z^2{\cdot}\mathbf{2}^n$.

iii. $\hat{t} \stackrel{?}{=} \langle \mathbf{l}, \mathbf{r} \rangle$.

As x is chosen randomly, this is equivalent to checking equation (3.4). However instead of sending $\mathbf{l}$, $\mathbf{r}$ (size $2n$) directly, the prover uses the optimized inner product proof of $\log_2 n$ size to prove that $\hat{t} = \langle \mathbf{l}, \mathbf{r} \rangle$. Hence the range proof is a logarithmic size range proof. Omniring and the MProve+ schemes follow a similar idea as discussed above.

Omniring

For a single source transaction in Omniring [22], we can prove the knowledge of the secret key corresponding to one element in the ring $\mathbf{P}$ (represented by a vector) by proving knowledge of a secret key ($x \in \mathbb{Z}_q$) and one secret unit vector $\mathbf{e}$ such that $\mathbf{P}^{\mathbf{e}} = G^x$. The unit vector $\mathbf{e}$ has zeros in $n-1$ places and 1 in the location corresponding to the source one-time address location in the ring. Therefore $\mathbf{e}$ selects only the source address in the ring vector $\mathbf{P}$. For a multiple source transaction, separate unit vectors are needed. The discrete logarithm relation can be alternatively represented as

$$1_g = G^{-x}\mathbf{P}^{\mathbf{e}}, \tag{3.5}$$

where 1_g is the identity element of the group $\mathbb{G}$. The Omniring authors called this equation the *main equality*. For a multiple source transaction in Omniring, the secret vector is formed by concatenating all the secret keys, unit vectors, output amounts, and blinding factors. The constraint equations are formed to ensure that the unit vectors contain zeros in all places except a single 1 in the source address location, the output amounts are in the right range, and the sum of input amounts is equal to the sum of output amounts and transaction fees for the transaction. The equations are added with blinding factors to form a single inner product like Bulletproofs. Then a technique similar to the Bulletproofs-based range proof is followed except with the following difference.

Let us define the secret vector as $\mathbf{a} = (-x \| \mathbf{e})$. Then the main equality (3.5) can be alternatively represented as

$$(G \| \mathbf{P})^{\mathbf{a}} = 1_g. \tag{3.6}$$

In the Bulletproofs-based range proof, to generate commitment A to the secret vectors $\mathbf{a}_L$ and $\mathbf{a}_R$, random base vectors $\mathbf{G}$ and $\mathbf{H}$ are chosen by the prover. As they are randomly generated, discrete logarithm relation between elements of the base vectors are

not known. This is necessary and used in the extraction of the witnesses. In Omniring, from equation (3.6) we observe that the base vectors to generate A must include $\mathbf{P}$ to show that the main equality (3.5) holds. However, the prover might know the discrete logarithm relation between the elements of $\mathbf{P}$ especially when some of them are owned by the prover. The Omniring scheme mitigates this issue by replacing the base vector $\mathbf{G}$ with $\mathbf{G}_w := ((G\|\mathbf{P})^w \circ \mathbf{Q})$ where w, $\mathbf{Q}$ are randomly chosen from $\mathbb{Z}_q$ and $\mathbb{G}^{n+1}$ respectively. Even if a discrete logarithm relation between elements of $\mathbf{P}$ is known, it is computationally infeasible to compute a discrete logarithm relation between elements of $\mathbf{G}_w$. Further, for $w' \neq w$, it holds that $\mathbf{G}_w^{\mathbf{a}} = \mathbf{G}_{w'}^{\mathbf{a}}$ if the main equality (3.5) holds. Recall that the same base $\mathbf{G}$ is used to generate A and S in the Bulletproofs-based range proof. In Omniring, $\mathbf{G}_0$ is used to generate A and $\mathbf{G}_w$ is used to generate S, where $w \xleftarrow{\$} \mathbb{Z}_q$ is sent by the verifier after receiving A. The rest of the protocol will work only if $\mathbf{G}_0^{\mathbf{a}} = \mathbf{G}_w^{\mathbf{a}}$ holds. In this way the main equality (3.5) is implicitly verified. The MProve+ scheme uses this technique.

3.2 MProve+: An Improvement over MProve

In this section, we describe MProve+ which helps to remove the drawback of MProve using the techniques of Bulletproofs and Omniring.

3.2.1 Intuition

In both MProve and MProve+ schemes, a Monero exchange Ex reveals a list of one-time addresses $\mathscr{P}_{\text{anon}} = \{P_1, P_2, \ldots, P_n\}$ as the anonymity set. Suppose in the set $\mathscr{P}_{\text{anon}}$, Ex owns s one-time addresses which are to be used as source addresses. In both the schemes, the key images corresponding to the source addresses are published to show that the source addresses are not spent yet. In the MProve scheme, a key image (real or dummy) is published for each one-time address in $\mathscr{P}_{\text{anon}}$. This is to hide the fact that a particular address in $\mathscr{P}_{\text{anon}}$ is a source address. However, this creates the association of a key image with a unique address in the anonymity set $\mathscr{P}_{\text{anon}}$ and introduces the privacy issue discussed in Section 3.1.2. In the MProve+ scheme, we publish the key images corresponding to only the source addresses in $\mathscr{P}_{\text{anon}}$, without revealing the association between the key images and their actual source addresses. An observer will be only able

to infer that each key image can be the key image of any address in the set $\mathscr{P}_{\text{anon}}$. While this reveals the number of source addresses s, the association of a key image with multiple one-time addresses helps to remove the drawback of the MProve scheme (as discussed in Section 3.3.3). Below we give an overview of the MProve+ scheme.

- In the MProve+ scheme, *Ex* publishes a vector of one-time addresses $\mathbf{P} = (P_1, P_2, \ldots, P_n)$ which have Pedersen commitments $\mathbf{C} = (C_1, C_2, \ldots, C_n)$ associated with them. *Ex* also reveals a key image vector $\mathbf{I} = (I_1, I_2, \ldots, I_s)$ and a Pedersen commitment C_{res} to the total reserves.

- First, *Ex* wants to prove that it knows the s secret keys corresponding to some s of the n addresses in $\mathbf{P}$. In other words, it wants to prove that there are s distinct indices $\{i_1, i_2, \ldots, i_s\} \subset \{1, 2, \ldots, n\}$ such that it knows $\{x_1, x_2, \ldots, x_s\}$ where $P_{i_j} = G^{x_j}$ for all $j = 1, 2, \ldots, s$. *Ex* does not want to reveal the indices.

- Second, *Ex* wants to prove that the key images $\mathbf{I} = (I_1, I_2, \ldots, I_s)$ correspond to the same s indices. In other words, $I_j = \left(H_p(P_{i_j})\right)^{x_j}$ for $j = 1, 2, \ldots, s$.

- Third, *Ex* wants to prove that for the same s indices it knows[‡] the blinding factor $r_j \in \mathbb{Z}_q$ and the amount $a_j \in \{0, 1, \ldots, 2^{\beta} - 1\}$ corresponding to the Pedersen commitments $C_{i_1}, C_{i_2}, \ldots, C_{i_s}$.

- Finally, *Ex* wants to prove that the amount in C_{res} is the same as the sum of the amounts in the Pedersen commitments at the same s indices. In other words, if $C_{\text{res}} = G^{r_{\text{res}}} H^{a_{\text{res}}}$ and $C_{i_j} = G^{r_j} H^{a_j}$, then *Ex* wants to prove that $a_{\text{res}} = \sum_{j=1}^{s} a_j$.

To prove the above statements, *Ex* proceeds as follows.

- *Ex* proves knowledge of s secret keys, amounts, and blinding factors by proving knowledge of s unit vectors $\mathbf{e}_1, \mathbf{e}_2, \ldots, \mathbf{e}_s$, secret vectors $\boldsymbol{x} = (x_1, x_2, \ldots, x_s)$, $\mathbf{a} = (a_1, a_2, \ldots, a_s)$, and $\mathbf{r} = (r_1, r_2, \ldots, r_s)$ such that $\mathbf{P}^{\mathbf{e}_j} = G^{x_j} \wedge \mathbf{C}^{\mathbf{e}_j} = G^{r_j} H^{a_j}$ holds for $j = 1, 2, \ldots, s$. As discussed in Section 3.1.3, the jth unit vector $\mathbf{e}_j$ is used to choose the jth source address and its corresponding Pedersen commitment in the vectors $\mathbf{P}$ and $\mathbf{C}$ respectively.

[‡]To prove ownership, proving knowledge of the secret key is enough (Section 2.2.3). However, if we show the knowledge of the source amounts and the blinding factors, then the commitment to the total reserves is more efficiently computed giving better performance.

- As proving $I_j = \left(H_p(P_{i_j})\right)^{x_j}$ is the same as proving $I_j^{x_j^{-1}} = H_p(P_{i_j})$, *Ex* proves that the unit vectors $\mathbf{e}_1, \mathbf{e}_2, \ldots, \mathbf{e}_s$ and secret keys in the vector $\boldsymbol{x} = (x_1, x_2, \ldots, x_s)$ also satisfy $\mathbf{H}_p^{\mathbf{e}_j} = I_j^{x_j^{-1}}$ for all $j = 1, 2, \ldots, s$, where $\mathbf{H}_p = (H_p(P_1), H_p(P_2), \ldots, H_p(P_n))$ is the vector of hashed one-time addresses.

- Finally, *Ex* shows that $a_{\text{res}} = \sum_{j=1}^{s} a_j$ by proving that there exists a binary[§] vector $\mathbf{b} = (b_0, b_1, \ldots, b_{\beta-1})$ such that $\sum_{i=0}^{\beta-1} b_i 2^i = a_{\text{res}} \wedge \sum_{i=0}^{\beta-1} b_i 2^i = \sum_{j=1}^{s} a_j$ holds.

All the above mentioned conditions for a proof of reserves are accumulated in a main equality similar to Omniring [22]. All the elements in the exponents of the main equality form the secret vector. To show that these elements of the secret vector satisfy all the necessary conditions, some constraint equations are formed. These equations collectively form a single inner product. We use Bulletproofs to prove that this inner product holds in zero-knowledge as discussed in Section 3.1.3. The main equality is also implicitly verified during this inner product verification. This is similar to the technique that Omniring uses and is discussed in Section 3.1.3. Below, we describe the MProve+ scheme in detail.

3.2.2 Construction of MProve+

The MProve+ protocol is constructed by modifying the scheme given in Appendix F of the Omniring [22] paper. Roughly speaking, a MProve+ proof is a giant Omniring transaction with a single output commitment, namely, a commitment to the total reserves. The differences between the MProve+ protocol and the protocol given in Appendix F of the Omniring paper [22] are as follows.

1. A one-time address is denoted as $P = H^x$ in the Omniring scheme. However it is denoted as $P = G^x$ in the MProve+ scheme.

2. A commitment is denoted as $C = G^a H^r$ in the Omniring scheme where a, r denote the amount and the blinding factor respectively. However a commitment in the MProve+ scheme is denoted by $C = G^r H^a$.

[§]This binary representation basically gives a range proof on a_{res} and is motivated from Omniring. In our case, ranges of $a_1, a_2, \ldots, a_s$ are already verified in the blockchain. Therefore proving that a_{res} is the sum of them implicitly verifies its range. However it is observed that, using the binary representation $\mathbf{b}$ instead of a_{res} in the secret vector gives better performance.

3. The number of outputs $|\tau|$ in the Omniring scheme is 1 in the MProve+ scheme. Hence the binary vector $\mathsf{vec}(\mathbf{B})$ of length $\beta|\tau|$ in $\mathbf{c}_L$ is replaced by a binary vector $\mathbf{b}$ of length β in the MProve+ scheme.

Below we give the language for the MProve+ protocol satisfying the requirements mentioned in Section 3.2.1.

$$\mathcal{L}^{\mathsf{crs}}_{\mathsf{MP+}} = \left\{ \begin{pmatrix} \mathbf{P},\, \mathbf{C},\, \mathbf{H}_p, \\ \{I_j\}_{j=1}^{s},\, C_{\text{res}} \end{pmatrix} \;\middle|\; \begin{array}{l} \exists(\boldsymbol{x}, \mathbf{e}_1, \ldots, \mathbf{e}_s, \mathbf{b}, \mathbf{a}, \mathbf{r}, a_{\text{res}}, r_{\text{res}}) \\ \text{such that each } \mathbf{e}_j \text{ is a unit vector,} \\ \mathbf{P}^{\mathbf{e}_j} = G^{x_j}, x_j \in \boldsymbol{x}, \\ \mathbf{C}^{\mathbf{e}_j} = G^{r_j} H^{a_j}, r_j \in \mathbf{r}, a_j \in \mathbf{a}, \\ \mathbf{H}_p^{\mathbf{e}_j} = I_j^{x_j^{-1}} \; \forall j \in [s], \\ \mathbf{b} \text{ is the binary representation of } a_{\text{res}} \text{ of length } \beta, \\ C_{\text{res}} = G^{a_{\text{res}}} H^{r_{\text{res}}}, \\ \sum_{a_j \in \mathbf{a}} a_j = a_{res}. \end{array} \right\} \tag{3.7}$$

Here the common reference string crs specifies the necessary details like description of the group, its generators, the hash function $H_p(\cdot)$ to be used. We define $(\mathbf{P}, \mathbf{C}, \mathbf{H}_p, \{I_j\}_{j=1}^{s}, C_{\text{res}})$ as the statement stmt of the language. We also define $(\boldsymbol{x}, \mathbf{e}_1, \ldots, \mathbf{e}_s, \mathbf{b}, \mathbf{a}, \mathbf{r}, a_{\text{res}}, r_{\text{res}})$ as the witness wit of the language.

3.2.3 Forming the Main Equality

We define $\boldsymbol{\mathcal{E}}$ as the $s \times n$ matrix containing $\mathbf{e}_1, \ldots, \mathbf{e}_s$ as the rows. We give the following definitions similar to the definitions in Appendix F of the Omniring paper [22]. Here u, v are public coin challenges sent by the verifier.

$$\hat{\mathbf{Y}} \coloneqq \mathbf{P} \circ \mathbf{C}^{\circ u} \circ \mathbf{H}_p^{\circ u^2}, \tag{3.8}$$

$$\hat{\mathbf{I}} \coloneqq \mathbf{I}^{\circ -u^2 \boldsymbol{v}^s}, \tag{3.9}$$

$$\hat{\mathbf{e}} \coloneqq \boldsymbol{v}^s \boldsymbol{\mathcal{E}}, \tag{3.10}$$

$$\xi \coloneqq -\langle \boldsymbol{v}^s, u \cdot \mathbf{a} \rangle, \tag{3.11}$$

$$\eta \coloneqq -\langle \boldsymbol{v}^s, \boldsymbol{x} + u \cdot \mathbf{r} \rangle. \tag{3.12}$$

The main equality is formed to verify the following representations from the language given in (3.7).

$$G^{-x_j}\mathbf{P}^{\mathbf{e}_j} = 1_g, \forall j \in [s], \tag{3.13}$$

$$G^{-r_j}H^{-a_j}\mathbf{C}^{\mathbf{e}_j} = 1_g, \forall j \in [s], \tag{3.14}$$

$$I_j^{-x_j^{-1}}\mathbf{H}_p^{\mathbf{e}_j} = 1_g, \forall j \in [s], \tag{3.15}$$

where 1_g is the identity element of $\mathbb{G}$. Equations (3.13), (3.14), and (3.15) represent s equations each. We exponentiate the jth equation ($j \in [s]$) of (3.13), (3.14), and (3.15) with v^{j-1}, uv^{j-1}, and u^2v^{j-1} respectively. Multiplying all these modified equations together gives us the following equation.

$$H^{\xi}G^{\eta}\hat{\mathbf{Y}}^{\hat{\mathbf{e}}}\hat{\mathbf{I}}^{\boldsymbol{x}^{\circ -1}} = 1_g. \tag{3.16}$$

Equation (3.16) is called the main equality for MProve+.

$$\mathbf{c}_L \coloneqq (\ \xi \parallel \eta \parallel \hat{\mathbf{e}} \parallel \boldsymbol{x}^{\circ -1} \parallel \quad \mathsf{vec}(\mathscr{E}) \quad \parallel \quad \mathbf{b} \quad \parallel \mathbf{a} \parallel \mathbf{r}\ )$$
$$\mathbf{c}_R \coloneqq (\quad \mathbf{0}^{2+n} \quad \parallel \quad \boldsymbol{x} \quad \parallel \mathbf{1}^{sn} - \mathsf{vec}(\mathscr{E}) \parallel \mathbf{1}^{\beta} - \mathbf{b} \parallel \quad \mathbf{0}^{2s} \quad)$$

Figure 3.1: Honest encoding of witness in MProve+.

$$\begin{bmatrix} \mathbf{v}_0 \\ \mathbf{v}_1 \\ \mathbf{v}_2 \\ \mathbf{v}_3 \\ \mathbf{v}_4 \\ \mathbf{v}_5 \\ \mathbf{v}_6 \\ \mathbf{v}_7 \\ \mathbf{v}_8 \\ \mathbf{u}_5 \end{bmatrix} \coloneqq \begin{bmatrix} \cdot & \cdot & \cdot & \cdot & \boldsymbol{y}^{sn+\beta} & & \cdot & \cdot \\ \cdot & \cdot & \cdot & \boldsymbol{y}^{s} & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \mathbf{2}^{\beta} & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \boldsymbol{y}^{s} \otimes \mathbf{1}^{n} & \cdot & \cdot & \cdot \\ 1 & \cdot & \cdot & \cdot & \cdot & \cdot & u\boldsymbol{v}^{s} & \cdot \\ \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & u\boldsymbol{v}^{s} \\ \cdot & \cdot & -\boldsymbol{y}^{n} & \cdot & \boldsymbol{v}^{s} \otimes \boldsymbol{y}^{n} & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \mathbf{2}^{\beta} & -\mathbf{1}^{s} & \cdot \\ \cdot & \cdot & \cdot & \cdot & \boldsymbol{y}^{sn+\beta} & & \cdot & \cdot \\ \cdot & \cdot & \cdot & \boldsymbol{v}^{s} & \cdot & \cdot & \cdot & \cdot \end{bmatrix}$$

Figure 3.2: Definitions of constraint vectors (dots mean zero vectors) for MProve+.

3.2.4 Defining Secret Vectors and the Constraint Equations

Now we construct the secret vectors given in Figure 3.1. Here, we essentially need to consider all the exponents in the main equality (3.16). Additionally we need to consider

$\mathsf{vec}(\mathcal{E})$ which is a vector of length sn, formed by concatenating all the rows of matrix $\mathcal{E}$. Vector $\mathsf{vec}(\mathcal{E})$ is used to ensure that all the rows of $\mathcal{E}$ are indeed unit vectors i.e. they contain a single 1 and rest of the elements are 0. The vector $\mathbf{c}_L$ has 2 scalars (ξ, η), 3 vectors $(\boldsymbol{x}^{\circ -1}, \mathbf{a}, \mathbf{r})$ of length s, and 3 vectors $(\hat{\mathbf{e}}, \mathsf{vec}(\mathcal{E}), \mathbf{b})$ of length n, sn, and β respectively. Hence the length of $\mathbf{c}_L$ is $m = sn + n + 3s + 2 + \beta$. Vector $\mathbf{c}_R$ is an auxiliary vector of the same length m used to prove the constraints on the witnesses.

Figure 3.2 gives some constraint vectors which are used to select various parts of the secret vector and give a constraint in terms of an inner product. All these inner product constraint equations are given in Figure 3.4. Here equations (3.17) and (3.25) verify that all elements of $\mathsf{vec}(\mathcal{E})$ and $\mathbf{b}$ are either 0 or 1. Equation (3.18) verifies that the $(n+2+1)$th to $(n+2+s)$th elements of $\mathbf{c}_L$ and $\mathbf{c}_R$ are inverses of each other. Equation (3.19) verifies that $\mathbf{b}$ is the binary representation of a_{res}. Equation (3.20) verifies that each block of n elements of the vector $\mathsf{vec}(\mathcal{E})$ contains a single 1 and the rest of the elements are 0 i.e. each such block is a unit vector. Equations (3.21) and (3.22) verify the definitions of ξ and η given in equations (3.11) and (3.12) respectively. Equation (3.23) verifies the definition of $\hat{\mathbf{e}}$ given in equation (3.10). Lastly, equation (3.24) verifies the equality $\sum_{i=0}^{\beta-1} b_i 2^i = \sum_{j=1}^{s} a_j$. Notice that a_{res} is not a member of the secret vector and the verification $C_{\text{res}} \stackrel{?}{=} G^{r_{\text{res}}} H^{a_{\text{res}}}$ is not done in the set of constraint equations. As we shall see, this verification is done in the final verification equation (V3) similar to the Omniring scheme.

$$\boldsymbol{\theta} = \mathbf{v}_0 + z\mathbf{v}_1, \qquad \boldsymbol{\mu} \coloneqq \sum_{i=2}^{8} z^i \mathbf{v}_i,$$

$$\boldsymbol{\nu} = z^8 \mathbf{v}_8, \qquad \boldsymbol{\omega} = z^5 \mathbf{u}_5,$$

$$\kappa = z\langle \mathbf{1}^s, \boldsymbol{y}^s \rangle + z^3 \langle \mathbf{1}^s, \boldsymbol{y}^s \rangle + \langle \mathbf{1}^m, \boldsymbol{\nu} \rangle$$

$$\delta = z\langle \mathbf{1}^s, \boldsymbol{y}^s \rangle + z^3 \langle \mathbf{1}^s, \boldsymbol{y}^s \rangle + \langle \mathbf{1}^m, \boldsymbol{\nu} \rangle + \langle \boldsymbol{\alpha}, \boldsymbol{\mu} \rangle$$

$$\boldsymbol{\alpha} = \boldsymbol{\theta}^{\circ -1} \circ (\boldsymbol{\omega} + \boldsymbol{\nu}), \qquad \boldsymbol{\beta} = \boldsymbol{\theta}^{\circ -1} \circ \boldsymbol{\mu},$$

Figure 3.3: Definitions of constraint vectors (continued) for MProve+.

$$\mathsf{EQ}(\boldsymbol{\gamma}_L, \boldsymbol{\gamma}_R) = 0 \iff$$

$$\langle \boldsymbol{\gamma}_L, \boldsymbol{\gamma}_R \circ \mathbf{v}_0 \rangle = 0 \tag{3.17}$$

$$\langle \boldsymbol{\gamma}_L, \boldsymbol{\gamma}_R \circ \mathbf{v}_1 \rangle = \langle \mathbf{1}^s, \boldsymbol{y}^s \rangle \tag{3.18}$$

$$\langle \boldsymbol{\gamma}_L, \mathbf{v}_2 \rangle = a_{\text{res}} \tag{3.19}$$

$$\langle \boldsymbol{\gamma}_L, \mathbf{v}_3 \rangle = \langle \mathbf{1}^s, \boldsymbol{y}^s \rangle \tag{3.20}$$

$$\langle \boldsymbol{\gamma}_L, \mathbf{v}_4 \rangle = 0 \tag{3.21}$$

$$\langle \boldsymbol{\gamma}_L, \mathbf{v}_5 \rangle + \langle \boldsymbol{\gamma}_R, \mathbf{u}_5 \rangle = 0 \tag{3.22}$$

$$\langle \boldsymbol{\gamma}_L, \mathbf{v}_6 \rangle = 0 \tag{3.23}$$

$$\langle \boldsymbol{\gamma}_L, \mathbf{v}_7 \rangle = 0 \tag{3.24}$$

$$\langle \boldsymbol{\gamma}_L + \boldsymbol{\gamma}_R - \mathbf{1}^m, \mathbf{v}_8 \rangle = 0 \tag{3.25}$$

Figure 3.4: A system of constraint equations guaranteeing integrity of encoding of witness for MProve+.

3.2.5 Combining All Constraint Equations in a Single Inner Product

For a random scalar $z \in \mathbb{Z}_q$ sent by the verifier, multiplying equations (3.17) to (3.25) by consecutive powers of z namely $1, z, z^2, \ldots, z^8$ and adding them gives,

$$\langle \boldsymbol{\gamma}_L, \boldsymbol{\gamma}_R \circ \boldsymbol{\theta} + \boldsymbol{\mu} \rangle + \langle \boldsymbol{\omega} + \boldsymbol{\nu}, \boldsymbol{\gamma}_R \rangle = \kappa + z^2 a_{\text{res}}, \tag{3.26}$$

where $\boldsymbol{\theta}, \boldsymbol{\nu}$, and κ are defined in Figure 3.3. To get a single inner product, we modify (3.26) as follows,

$$\langle \boldsymbol{\gamma}_L, \boldsymbol{\gamma}_R \circ \boldsymbol{\theta} + \boldsymbol{\mu} \rangle + \langle (\boldsymbol{\omega} + \boldsymbol{\nu}) \circ \boldsymbol{\theta}^{\circ -1}, \boldsymbol{\gamma}_R \circ \boldsymbol{\theta} + \boldsymbol{\mu} \rangle \tag{3.27}$$

$$= \kappa + \langle (\boldsymbol{\omega} + \boldsymbol{\nu}) \circ \boldsymbol{\theta}^{\circ -1}, \boldsymbol{\mu} \rangle + z^2 a_{\text{res}}$$

$$\implies \langle \boldsymbol{\gamma}_L + \boldsymbol{\alpha}, \boldsymbol{\gamma}_R \circ \boldsymbol{\theta} + \boldsymbol{\mu} \rangle = \delta + z^2 a_{\text{res}}, \tag{3.28}$$

where $\boldsymbol{\alpha}$ and δ are defined in Figure 3.3.

In the following protocol, we prove the inner product given in equation (3.28) using the Bulletproofs technique as discussed in Section 3.1.3. The main equality (3.16) is implicitly proved using the technique followed in Omniring as discussed in Section 3.1.3.

MProve+ Protocol (Π_{MPP}): Argument of knowledge for $\mathcal{L}^{\mathsf{crs}}_{\mathsf{MP+}}$.

$\mathsf{Setup}(\lambda)$:

$\mathsf{crs} = (\mathbb{G}, q, G, H, H_p(\cdot))$.

Generate: $r_{\text{res}} \xleftarrow{\$} \mathbb{Z}_q, a_{\text{res}} = \sum_{a_j \in \mathbf{a}} a_j, C_{\text{res}} = G^{r_{\text{res}}} H^{a_{\text{res}}}$,

$\mathbf{b} = (b_0, b_1, \ldots, b_{\beta-1})$, such that $\sum_{k=0}^{\beta-1} b_k 2^k = a_{\text{res}}, \mathbf{P} = \{P_j\}_{j=1}^n, \mathbf{C} = \{C_j\}_{j=1}^n$,

$\mathbf{H}_p = \{H_P(P_j)\}_{j=1}^n, \mathbf{I} = \{I_j\}_{j=1}^s = \{H_p(P_j)^{x_j}\}_{j=1}^s$

Output: $\mathsf{stmt} = (\mathbf{P}, \mathbf{C}, \mathbf{H}_p, \mathbf{I}, C_{\text{res}})$, $\mathsf{wit} = (\mathbf{x}, \mathbf{e}_1, \ldots, \mathbf{e}_s, \mathbf{b}, \mathbf{a}, \mathbf{r}, a_{\text{res}}, r_{\text{res}})$

$\langle \mathcal{P}(\mathsf{crs}, \mathsf{stmt}, \mathsf{wit}), \mathcal{V}(\mathsf{crs}, \mathsf{stmt}) \rangle$:

$\mathcal{V}$: $u, v \xleftarrow{\$} \mathbb{Z}, F \xleftarrow{\$} \mathbb{G}, \mathbf{Q} \xleftarrow{\$} \mathbb{G}^{2+n+s}, \mathbf{G}' \xleftarrow{\$} \mathbb{G}^{m-n-s-2}, \mathbf{H} \xleftarrow{\$} \mathbb{G}^m$

$\mathcal{V} \longrightarrow \mathcal{P}$: $u, v, F, \mathbf{Q}, \mathbf{G}', \mathbf{H}$

$\mathcal{P}$, $\mathcal{V}$:

1. Compute $\hat{\mathbf{Y}} = \mathbf{P} \circ \mathbf{C}^{\circ u} \circ \mathbf{H}_p^{\circ u^2}$ and $\hat{\mathbf{I}} = \mathbf{I}^{\circ -u^2 \boldsymbol{v}^s}$
2. For $w \in \mathbb{Z}_q$, denote

$$\mathbf{G}_w := \left[\left((H \| G \| \hat{\mathbf{Y}} \| \hat{\mathbf{I}})^{\circ w} \circ \mathbf{Q} \right) \| \mathbf{G}' \right] \tag{3.29}$$

$\mathcal{P}$:

1. $r_A \xleftarrow{\$} \mathbb{Z}_q$
2. $A := F^{r_A} \mathbf{G}_0^{\mathbf{c}_L} \mathbf{H}^{\mathbf{c}_R}$

   Note: $\mathbf{G}_w^{\mathbf{c}_L} = \mathbf{G}_{w'}^{\mathbf{c}_L}$ $\forall w, w' \in \mathbb{Z}_q$ since $H^{\xi} G^{\eta} \hat{\mathbf{Y}}^{\hat{\mathbf{e}}} \hat{\mathbf{I}}^{\boldsymbol{x}^{\circ -1}} = 1_g$ by the main equality 3.16. Thus $A = F^{r_A} \mathbf{G}_w^{\mathbf{c}_L} \mathbf{H}^{\mathbf{c}_R}$ $\forall w \in \mathbb{Z}_q$

$\mathcal{P} \longrightarrow \mathcal{V}$: A

$\mathcal{V}$: $w \xleftarrow{\$} \mathbb{Z}_q$

$\mathcal{V} \longrightarrow \mathcal{P}$: w

$\mathcal{P}$:

1. $r_S \xleftarrow{\$} \mathbb{Z}_q, \mathbf{s}_L \xleftarrow{\$} \mathbb{Z}_q^m$, for $\mathbf{s}_R \in \mathbb{Z}_q^m$ s.t. for $j \in [m]$

$$\mathbf{s}_R[j] = \begin{cases} s_j \xleftarrow{\$} \mathbb{Z}_q, & \text{for } \mathbf{c}_R[j] \neq 0 \\ 0, & \text{for } \mathbf{c}_R[j] = 0 \end{cases}$$

2. $S = F^{r_S} \mathbf{G}_w^{\mathbf{s}_L} \mathbf{H}^{\mathbf{s}_R}$

$\mathcal{P} \longrightarrow \mathcal{V}$: S

$\mathcal{V}$: $y, z \xleftarrow{\$} \mathbb{Z}_q$

$\mathcal{V} \longrightarrow \mathcal{P}$: y, z

$\mathcal{P}$:

1. Define the following polynomials (in X):

$$\begin{aligned} l(X) &\coloneqq \mathbf{c}_L + \boldsymbol{\alpha} + \mathbf{s}_L \cdot X && \in \mathbb{Z}_q^m[X] \\ r(X) &\coloneqq \boldsymbol{\theta} \circ (\mathbf{c}_R + \mathbf{s}_R \cdot X) + \boldsymbol{\mu} && \in \mathbb{Z}_q^m[X] \\ t(X) &\coloneqq \langle l(X), r(X) \rangle = t_2 X^2 + t_1 X + t_0 && \in \mathbb{Z}_q^N[X] \end{aligned}$$

   for some $t_2, t_1, t_0 \in \mathbb{Z}_q$. In particular, $t_0 = z^2 a_{\text{res}} + \delta$

2. $\tau_1, \tau_2 \xleftarrow{\$} \mathbb{Z}_q$

3. $T_1 = H^{t_1} G^{\tau_1}, T_2 = H^{t_2} G^{\tau_2}$

$\mathcal{P} \longrightarrow \mathcal{V}$: T_1, T_2

$\mathcal{V}$: $x \xleftarrow{\$} \mathbb{Z}_q$

$\mathcal{V} \longrightarrow \mathcal{P}$: x

$\mathcal{P}$:

1. $\boldsymbol{\ell} \coloneqq l(x) = \mathbf{c}_L + \boldsymbol{\alpha} + \mathbf{s}_L \cdot x \in \mathbb{Z}_q^m$
2. $\boldsymbol{r} \coloneqq r(x) = \boldsymbol{\theta} \circ (\mathbf{c}_R + \mathbf{s}_R \cdot x) + \boldsymbol{\mu} \in \mathbb{Z}_q^m$
3. $\hat{t} \coloneqq \langle \boldsymbol{\ell}, \boldsymbol{r} \rangle \in \mathbb{Z}_q$
4. $\tau \coloneqq z^2 r_{\text{res}} + \tau_2 x^2 + \tau_1 x$
5. $r \coloneqq r_A + r_S x$

$\mathcal{P} \longrightarrow \mathcal{V}$: $\boldsymbol{\ell}, \boldsymbol{\imath}, \hat{t}, \tau, r$

$\mathcal{V}$: Checks if the following relations hold:

(V1) $\hat{t} \stackrel{?}{=} \langle \boldsymbol{\ell}, \boldsymbol{\imath} \rangle$

(V2) $F^r \mathbf{G}_w^{\boldsymbol{\ell}} \mathbf{H}^{\boldsymbol{\theta}^{\circ -1} \circ \boldsymbol{\imath}} \stackrel{?}{=} AS^x \mathbf{G}_w^{\boldsymbol{\alpha}} \mathbf{H}^{\boldsymbol{\beta}}$

(V3) $H^{\hat{t}} G^{\tau} \stackrel{?}{=} H^{\delta} C_{\text{res}}^{z^2} T_1^x T_2^{x^2}$

Verification equations (V1) and (V2) need $\boldsymbol{\ell}, \boldsymbol{\imath} \in \mathbb{Z}_q^m$ which requires $\mathcal{O}(m)$ size communication from the prover. Instead, we can use the inner product protocol which is used in Bulletproofs [17] and Omniring [22]. The inner product argument is expressed by the following language.

$$\mathcal{L}_{IP} = \left\{ P \in \mathbb{G}, c \in \mathbb{Z}_q \;\middle|\; \begin{array}{c} \exists (\mathbf{a}, \mathbf{b}) \text{ such that} \\ P = U^c \mathbf{G}^{\mathbf{a}} \mathbf{H}^{\mathbf{b}} \wedge c = \langle \mathbf{a}, \mathbf{b} \rangle. \end{array} \right\} \tag{3.30}$$

where $\mathbf{a}, \mathbf{b} \in \mathbb{Z}_q^{|\mathbf{a}|}$, $\mathbf{G}, \mathbf{H} \xleftarrow{\$} \mathbb{G}^{|\mathbf{a}|}$, $U \xleftarrow{\$} \mathbb{G}$. In our case, the verifier sets $c = \hat{t}$. Apart from the prover, the verifier can also compute the Pedersen commitment P to $\boldsymbol{\ell}$ and $\boldsymbol{\imath}$ without knowing $\boldsymbol{\ell}$ and $\boldsymbol{\imath}$ as

$$P = U^{\hat{t}} \mathbf{G}_w^{\boldsymbol{\ell}} (\mathbf{H}')^{\boldsymbol{\imath}} = U^{\hat{t}} (F)^{-r} AS^x \mathbf{G}_w^{\boldsymbol{\alpha}} \mathbf{H}^{\boldsymbol{\beta}}, \tag{3.31}$$

by verification equation (V2), where $\mathbf{H}' = \mathbf{H}^{\boldsymbol{\theta}^{\circ -1}}$. With this, the prover and the verifier engage in the inner product argument to prove verification equations (V1) and (V2). So the prover does not send $\boldsymbol{\ell}$ and $\boldsymbol{\imath}$ in the previous step reducing the communication cost to $\mathcal{O}(\log_2(m))$. The inner product argument is public coin, so can be done by only one interaction between the prover and the verifier using the Fiat-Shamir heuristic. We have the following theorems which come directly from Theorem F.2 and F.3 of the Omniring paper [22], hence their proofs are omitted.

Theorem 3.1. *The argument presented in* Π_{MPP} *is public-coin, constant-round, perfectly complete and perfect special honest-verifier zero-knowledge.*

Theorem 3.2. *Assuming the discrete logarithm assumption holds over* $\mathbb{G}$, Π_{MPP} *has computational witness-extended-emulation for extracting a valid witness* wit.

3.2.6 Proof Generation and Verification

The exchange follows the Π_{MPP} protocol and publishes $(\mathbf{P}, \mathbf{I}, C_{\text{res}})$ and a Π_{MPP} proof. The verifier of Π_{MPP} protocol does the following verification steps.

1. Computes $\mathbf{H}_p$ using the hash function $H_p(\cdot)$. Reads $\mathbf{C}$ by looking at the Monero blockchain and using $\mathbf{P}$.

2. Checks that no element in $\mathbf{I}$ appears in the set of key images $\mathcal{I}$. If this is not the case then double spending is detected.

3. Checks that all the elements in $\mathbf{I}$ are distinct. This is to ensure that no source amount is used more than once in calculating the total reserves.

4. Checks the proof of Π_{MPP} as discussed above.

5. Checks that no element in $\mathbf{I}$ appears in the MProve+ proofs generated by another Monero exchange. If this is not the case then address sharing collusion is detected.

The verifier rejects the proof if any of the above steps fails. Otherwise she accepts the proof. For faster verification, we have done some optimization as discussed below.

Faster Verification

The cost of verifying an MProve+ proof is largely determined by the verification of the argument of knowledge Π_{MPP}. A verifier checks the validity of Π_{MPP} by checking the verification equation (V3) and the inner product argument. As noted in [17], an inner product argument $\Pi_{\mathsf{IP}} = \left(\{L_j, R_j\}_{j=1}^{\log_2 m} \in \mathbb{G}, a, b \in \mathbb{Z}_q\right)$ for the language in (3.30) can be verified in a single multi-exponentiation check as

$$\mathbf{G}^{a\cdot\mathbf{s}} \cdot \mathbf{H}^{a\cdot\mathbf{s}^{\circ -1}} \cdot U^{a\cdot b} = P \cdot \prod_{j=1}^{\log_2 m} L_j^{x_j^2} \cdot R_j^{x_j^{-2}}. \tag{3.32}$$

where $\mathbf{s} = \{s_i\}_{i=1}^{N}, s_i = \prod_{j=1}^{\log_2 m} x_j^{b(i,j)}$ such that $b(i,j)$ is 1 if the j-th bit of $(i-1)$ is 1, and -1 otherwise. Note that $\mathbf{s}$ depends only on the challenges $\{x_j\}_{j=1}^{\log_2 m}$. For the inner product argument associated with Π_{MPP}, substituting the expression of P from (3.31), we get

$$\mathbf{G}_w^{a\cdot\mathbf{s}} \cdot \mathbf{H}^{b\cdot(\boldsymbol{\theta}\circ\mathbf{s})^{\circ -1}} \cdot U^{a\cdot b} = \left(U^{\hat{t}}(F)^{-r} A S^x \mathbf{G}_w^{\boldsymbol{\alpha}} \mathbf{H}^{\boldsymbol{\beta}}\right) \cdot \prod_{j=1}^{\log_2 m} L_j^{x_j^2} \cdot R_j^{x_j^{-2}}. \tag{3.33}$$

Moving everything to the LHS, we get

$$\mathbf{G}_w^{a\cdot\mathbf{s}-\boldsymbol{\alpha}} \cdot \mathbf{H}^{b\cdot(\boldsymbol{\theta}\circ\mathbf{s})^{\circ-1}-\boldsymbol{\beta}} \cdot U^{a\cdot b-\hat{t}} \cdot (F)^r \cdot A^{-1} \cdot S^{-x} \cdot \prod_{j=1}^{\log_2 m} L_j^{-x_j^2} \cdot R_j^{-x_j^{-2}} = 1_g. \tag{3.34}$$

Furthermore, we merge the verification equation (V3) in (3.35) using a random scalar $c \leftarrow \mathbb{Z}_q$.

$$\mathbf{G}_w^{a\cdot\mathbf{s}-\boldsymbol{\alpha}} \cdot \mathbf{H}^{b\cdot(\boldsymbol{\theta}\circ\mathbf{s})^{\circ-1}-\boldsymbol{\beta}} \cdot U^{a\cdot b-\hat{t}} \cdot (F)^r \cdot A^{-1} \cdot S^{-x} \cdot \prod_{j=1}^{\log_2 m} L_j^{-x_j^2} \cdot R_j^{-x_j^{-2}} \cdot \left(H^{\hat{t}-\delta} G^{\tau} C_{\text{res}}^{-z^2} T_1^{-x} T_2^{-x^2}\right)^c = 1_g. \tag{3.35}$$

Effectively, the verification of an MProve+ boils down to a single multi-exponentiation check of size $\mathcal{O}(2m + 2\log_2 m + 9)$.

3.3 Security Properties

In this section, we describe the same security properties of the MProve+ protocol which were discussed for the MProve and MProvisions protocols in the previous chapter. As discussed before, the collusion resistance property prevents two or more exchanges from sharing a common one-time address as a source of funds while generating a MProve+ reserves proof. The inflation resistance property prevents an exchange from publishing a commitment C_{res} to an amount which exceeds the actual reserves amount. The pre-spend privacy property provides privacy to the exchange given that it has not spent from a source one-time address used in the reserves proofs. Additionally, MProve+ alleviates the drawback seen in MProve/MProvisions if the exchange carefully chooses the transaction rings while spending from source addresses.

3.3.1 Inflation Resistance

We say that the MProve+ scheme is inflation resistant if no probabilistic polynomial time (PPT) exchange can generate an accepting MProve+ transcript committing to an amount $a'_{\text{res}} \neq \sum_{j=1}^{s} a_j$ as the reserves amount. This is similar to proving that in an Omniring transaction, the sum of inputs is equal to the sum of outputs. Hence the inflation resistance property for MProve+ follows directly from the balance property given in Theorem 4.2 of the Omniring [22] paper.

3.3.2 Collusion Resistance

In the MProve+ protocol, for each owned address P, the exchange has to publish a key image I such that the following relation holds,

$$P = G^x \wedge I = (H_p(P))^x, \tag{3.36}$$

where x is the secret key corresponding to P. Note that for a given P, the key image I in equation (3.36) is unique. Hence if two PPT exchanges use a common one-time address as a source address to generate reserves proofs, the key image corresponding to that one-time address will appear in both the reserves proofs. Thus a verifier can easily detect collusion between exchanges. If we assume the exchanges are PPT, then they can generate different key images for the same source address only with a negligible probability. This follows from the unforgeability of the argument of knowledge of the MProve+ protocol.

3.3.3 Privacy

A fundamental requirement for a Monero proof of reserves protocol is to show that the source addresses that are used in the proof are not spent already. The simplest way to show this is to reveal the key images corresponding to the source addresses. Any verifier can then check that the source addresses are unspent by checking if the key images have appeared in the set of key images $\mathcal{I}$ on the Monero blockchain.

One might wonder if it is possible to construct a reserves proof for Monero which does not explicitly reveal the key images of source addresses. A scheme called *UnspentProof* for proving that a one-time address is not spent without revealing the corresponding key image was proposed by Koe *et al.* [25, Section 8.1.5]. In Appendix B.1, we discuss the difficulties in using UnspentProof in a privacy focused proof of reserves protocol. We also discuss the other challenges in hiding the key images corresponding to the source addresses in the same appendix.

From the construction, we see that the MProve+ protocol publishes the key images of the source addresses explicitly. Hence when an exchange-owned address is spent in a later transaction, the fact that the exchange is spending in the transaction is revealed. In this aspect, MProve+ has the same drawback as MProve and MProvisions. However, the identity of the particular source address being spent in the transaction will not be revealed if the exchange chooses the ring in the transaction carefully.

In the following subsection, we characterize the information revealed by the publication of multiple MProve+ proofs by a set of bipartite graphs with edges between the anonymity sets and the key image sets. These graphs illustrate how MProve+ affects the privacy of the exchange as well as the privacy of the entire Monero network.

Privacy Implications of Publishing a Polynomial Number of MProve+ Proofs

Let $f(\lambda)$ denote a polynomial of the security parameter λ. Suppose a Monero exchange has generated $f(\lambda)$ MProve+ proofs with the anonymity sets and the key image sets $\{\mathbf{P}^{(i)}\}_{i=1}^{f(\lambda)}$ and $\{\mathbf{I}^{(i)}\}_{i=1}^{f(\lambda)}$ respectively. Let the corresponding cardinalities of those sets be $\{n_i\}_{i=1}^{f(\lambda)}$ and $\{s_i\}_{i=1}^{f(\lambda)}$ respectively.

Now let us consider the information revealed to a PPT adversary who observes $\left\{\mathbf{P}^{(i)}, \mathbf{I}^{(i)}\right\}_{i=1}^{f(\lambda)}$ together. First, consider only the ith MProve+ proof. When $(\mathbf{P}^{(i)}, \mathbf{I}^{(i)})$ is revealed together, then it is revealed is that any key image in $\mathbf{I}^{(i)}$ could have originated[¶] from any one-time address in $\mathbf{P}^{(i)}$. This information can be represented by a complete bipartite graph[‖] with the disjoint sets of vertices $(\mathbf{P}^{(i)}, \mathbf{I}^{(i)})$ and the edge set $\mathbf{P}^{(i)} \times \mathbf{I}^{(i)}$. Here an edge between a one-time address $P \in \mathbf{P}^{(i)}$ and a key image $I \in \mathbf{I}^{(i)}$ denotes that I could have originated from P.

Now consider the case when $f(\lambda)$ MProve+ proofs are published and $\left\{\mathbf{P}^{(i)}, \mathbf{I}^{(i)}\right\}_{i=1}^{f(\lambda)}$ are revealed. The information in the individual bipartite graphs can be combined to identify the set of one-time addresses which could have originated a particular key image. For a key image $I \in \{\mathbf{I}^{(i)}\}_{i=1}^{f(\lambda)}$, let $\mathscr{P}_{\text{orig}}(I)$ denote the set of one-time addresses of minimal cardinality which could have originated I, from the perspective of a PPT adversary which has access to the Monero blockchain and the MProve+ proofs. We call $\mathscr{P}_{\text{orig}}(I)$ the *originating set* for I. Suppose I has appeared in j_1th, j_2th,..., j_rth proofs among the overall $f(\lambda)$ MProve+ proofs. Then it is obvious that,

$$\mathscr{P}_{\text{orig}}(I) \subset \bigcap_{k=1}^{r} \mathbf{P}^{(j_k)}. \tag{3.37}$$

Consider the following example.

Example 3.1. *Suppose the adversary observes three MProve+ proofs. The anonymity*

[¶]The statement that the key image I has originated from the one-time address P implies that there exists a scalar $x \in \mathbb{Z}_q$ such that $P = G^x \wedge I = (H_p(P))^x$ holds.

[‖]This formulation was introduced in [30].

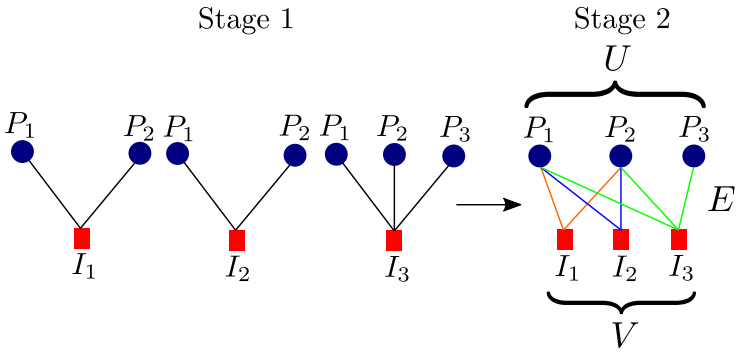

Figure 3.5: Illustration of Example 3.1.

sets and the key image sets are as follows.

$$\boldsymbol{P}^{(1)} = \{P_1, P_2\}, \qquad \boldsymbol{I}^{(1)} = \{I_1\},$$
$$\boldsymbol{P}^{(2)} = \{P_1, P_2\}, \qquad \boldsymbol{I}^{(2)} = \{I_2\},$$
$$\boldsymbol{P}^{(3)} = \{P_1, P_2, P_3\}, \qquad \boldsymbol{I}^{(3)} = \{I_3\}.$$

As there are 3 MProve+ proofs, there are 3 corresponding complete bipartite graphs as shown in stage 1 of Figure 3.5. If we use the intersection formula for $\mathscr{P}_{\text{orig}}(\cdot)$ as given in equation (3.37), then we get

$$\mathscr{P}_{\text{orig}}(I_3) \subset \mathbf{P}^{(3)} = \{P_1, P_2, P_3\}.$$

But one can see that P_3 is the only possible one-time address which could have possibly originated I_3. This is because $\{P_1, P_2\}$ together have to originate $\{I_1, I_2\}$[**]. This makes P_3 as the only member of $\mathbf{P}^{(3)}$ which could possibly originate I_3. To get a precise definition of the originating set, we construct the simple[††] bipartite graph (U, V, E), using the $f(\lambda)$ anonymity sets and key image sets. Here U, V are the disjoint vertex sets given by

$$U = \bigcup_{i=1}^{f(\lambda)} \mathbf{P}^{(i)}, \quad V = \bigcup_{i=1}^{f(\lambda)} \mathbf{I}^{(i)},$$

and E is the edge set given by

$$E = \bigcup_{i=1}^{f(\lambda)} \left(\mathbf{P}^{(i)} \times \mathbf{I}^{(i)}\right).$$

[**]The set $\{P_1, P_2\}$ is termed as *closed set* in [33]. This kind of structure makes some cover addresses useless in the anonymity sets/rings of transactions.

[††]By a simple graph, we mean undirected graph with no loops or multiple edges.

Since we are requiring the graph to be simple, the edge set E will not have multiple edges. If an edge appears in both $\mathbf{P}^{(i)} \times \mathbf{I}^{(i)}$ and $\mathbf{P}^{(j)} \times \mathbf{I}^{(j)}$ for $i \neq j$, then we include it only once. The bipartite graph (U, V, E) corresponding to Example 3.1 is shown in stage 2 of Figure 3.5. Here the orange edges, blue edges, and green edges of E come from the first, second, and the third proof respectively.

A matching on a graph is a subset of the edge set such that the subset elements have no common vertices [42]. We give the following definition for $\mathscr{P}_{\text{orig}}(I)$.

Definition 3.1. *Let $\mathcal{M}$ be the set of all maximum cardinality matchings on the bipartite graph (U, V, E) induced by the $f(\lambda)$ MProve+ proofs such that for each $M \in \mathcal{M}$ the set of edges $M \bigcap \left(\boldsymbol{P}^{(i)} \times \boldsymbol{I}^{(i)}\right)$ is a maximum cardinality matching in the bipartite graph $\left(\boldsymbol{P}^{(i)}, \boldsymbol{I}^{(i)}, \boldsymbol{P}^{(i)} \times \boldsymbol{I}^{(i)}\right)$ for all $i = 1, 2, \ldots, f(\lambda)$.*

We define $\mathscr{P}_{orig}(I)$ for a key image I in $\bigcup_{i=1}^{f(\lambda)} \boldsymbol{I}^{(i)}$ as

$$\mathscr{P}_{orig}(I) = \left\{ P \in \bigcup_{i=1}^{f(\lambda)} \boldsymbol{P}^{(i)} \middle| (P, I) \text{ belongs to a matching in } \mathcal{M} \right\}.$$

The above definition gives $\mathscr{P}_{\text{orig}}(I_1) = \mathscr{P}_{\text{orig}}(I_2) = \{P_1, P_2\}$ and $\mathscr{P}_{\text{orig}}(I_3) = \{P_3\}$ as desired. Now we give the following theorem which precisely characterizes the information revealed by the MProve+ protocol. The proof is given in Appendix B.2.

Theorem 3.3. *The only information that a PPT adversary can obtain from the $f(\lambda)$ MProve+ proofs is the $f(\lambda)$ bipartite graphs $\left(\boldsymbol{P}^{(i)}, \boldsymbol{I}^{(i)}, \boldsymbol{P}^{(i)} \times \boldsymbol{I}^{(i)}\right)_{i=1}^{f(\lambda)}$.*

Even if we were to disregard the edges in the graph, the key image sets $\{\mathbf{I}^{(i)}\}_{i=1}^{f(\lambda)}$ can affect the privacy of the exchange. For example, when a PPT adversary observes only $\{\mathbf{I}^{(i)}\}_{i=1}^{f(\lambda)}$, the following information is revealed to her.

1. The number of source addresses used in the proofs (cardinalities of $\mathbf{I}^{(i)}$s, i.e. s_is).
2. The number of new source addresses used in the $(i+k)$th proof ($k \geq 1$) which were not there in the ith proof (the number of new key images in $\mathbf{I}^{(i+k)}$ which were not there in $\mathbf{I}^{(i)}$).
3. The number of source addresses in the ith proof which were removed from the $(i+k)$th proof (the number of key image in $\mathbf{I}^{(i)}$ which are not there in $\mathbf{I}^{(i+k)}$).

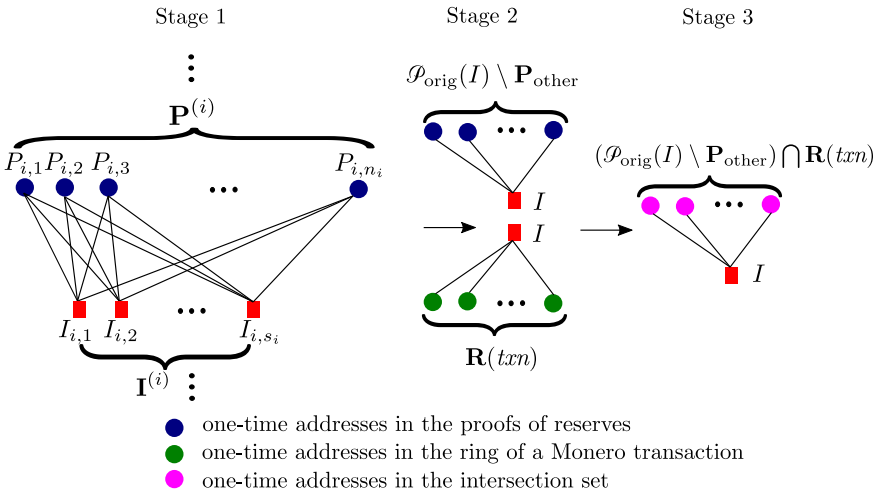

Figure 3.6: Linking key image for MProve+ when a source address is spent.

4. The number of source addresses which are being used repeatedly. For example, consider a key image I which has appeared in $\mathbf{I}^{(i)}$, removed in $\mathbf{I}^{(i+1)}$ onwards, and appears again in $\mathbf{I}^{(i+k)}$. Then the appearance of I reveals that a source address was used in the ith proof, not in use from the $(i+1)$th proof to the $(i+k)$th proof, and was used again in the $(i+k)$th proof.

Choice of the anonymity sets. In our analysis, we have shown that multiple MProve+ proofs can reveal the originating set $\mathscr{P}_{\text{orig}}(I)$ for a particular published key image I. While choosing the anonymity sets across multiple proofs, the exchange needs to have a proper strategy. The goal of such a strategy should be to make the cardinalities of the originating sets as large as possible. Proposing such a strategy is an interesting direction for future research.

Next, we discuss what happens when an exchange spends from a source address which was used in the MProve+ protocol. In particular, we describe the consequence of using MProve+ as the proof of reserves protocol in Example 2.1.

Effect of MProve+ on Monero transactions

Suppose Ex has used P as a source address in some of the $f(\lambda)$ published MProve+ proofs and I has appeared in some sets in $\{\mathbf{I}^{(i)}\}_{i=1}^{f(\lambda)}$. As discussed above, each MProve+ proof induces a complete bipartite graph. This is shown in stage 1 of Figure 3.6. From these $f(\lambda)$ MProve+ proofs, the originating set for I i.e. $\mathscr{P}_{\text{orig}}(I)$ is revealed. Let $\mathscr{A}$ be a PPT adversary which wants to obtain the originating address of I (here P) from $\mathscr{P}_{\text{orig}}(I)$. If $\mathscr{A}$

is a participant in the Monero network, then it might have the side information that some addresses in $\bigcup_{i=1}^{f(\lambda)} \mathbf{P}^{(i)}$ do not belong to Ex and are definitely cover addresses. We model this side information by the set $\mathbf{P}_{\text{other}} \subset \bigcup_{i=1}^{f(\lambda)} \mathbf{P}^{(i)}$. $\mathcal{A}$ is given access to the set $\mathbf{P}_{\text{other}}$. Now consider the scenario just before a source address spending transaction txn appears on the Monero blockchain. $\mathcal{A}$ knows that any address in $\mathbf{P}_{\text{other}}$ cannot be the originating address for I. If we ignore negligible probabilities, the probability that $\mathcal{A}$ successfully outputs P as the originating address for I is given by,

$$\Pr[\mathcal{A}(I, \mathscr{P}_{\text{orig}}(I), \mathbf{P}_{\text{other}}) = P] = \frac{1}{|\mathscr{P}_{\text{orig}}(I) \setminus \mathbf{P}_{\text{other}}|}. \tag{3.38}$$

Next, txn appears in the Monero blockchain with key image I and ring $\mathbf{R}(txn)$. The view of $\mathcal{A}$ in this situation is shown in stage 2 of Figure 3.6. With this additional information, $\mathcal{A}$ knows that any address in the set $(\mathscr{P}_{\text{orig}}(I) \setminus \mathbf{P}_{\text{other}}) \bigcap \mathbf{R}(txn)$ could be the originating address corresponding to I. The intersection of the corresponding graphs is shown in stage 3 of Figure 3.6. The equation (3.38) is modified as follows to give the probability that $\mathcal{A}$ successfully links P with I.

$$\Pr[\mathcal{A}(I, \mathscr{P}_{\text{orig}}(I), \mathbf{P}_{\text{other}}, \mathbf{R}(txn)) = P] = \frac{1}{|(\mathscr{P}_{\text{orig}}(I) \setminus \mathbf{P}_{\text{other}}) \bigcap \mathbf{R}(txn)|}. \tag{3.39}$$

It is desirable for Ex that the probability given in equation (3.39) is as low as possible. Assuming that Ex does not have the knowledge of $\mathbf{P}_{\text{other}}$, a strategy for Ex to reduce the probability in equation (3.39) is as follows.

For a given source address P with the associated key image I, Ex should choose anonymity sets in such a way that $\mathscr{P}_{orig}(I)$ becomes as large as possible. Later when Ex spends from P in txn, Ex should choose $\boldsymbol{R}(txn)$ as a proper subset of $\mathscr{P}_{orig}(I)$.

Now observe equation (2.35) and (3.39). For MProve, no matter how the anonymity set and the ring of the transaction are chosen, the linking probability is always 1. However for MProve+, the exchange can choose the anonymity sets and the transaction ring carefully and keep the linking probability away from 1. Hence we conclude that *MProve+ is better than MProve when the privacy of the entire Monero network including the exchange is of concern.*

3.4 Effect of MProve+ on Monero Privacy

As discussed in Section 2.2, the major privacy properties for the Monero scheme are untraceability, amount confidentiality, and unlinkability. We have discussed how the MProve and MProvisions protocols affect these privacy features in Section 2.7. In this section, we give a similar analysis for the MProve+ protocol.

3.4.1 Effect on the Untraceability Property of Monero

As discussed in Section 2.2.2, a Monero transaction is called untraceable when a PPT adversary cannot determine which one-time address in the ring is actually being spent i.e. originates the associated key image. Consider a key image I which has appeared across multiple MProve+ proofs. As discussed in Section 3.3.3, when I appears in the source spending transaction txn, it is linked with the source one-time address P if the set $(\mathscr{P}_{\text{orig}}(I) \setminus \mathbf{P}_{\text{other}}) \bigcap \mathbf{R}(txn)$ becomes a singleton set containing only P. The exchange can plausibly avoid this by choosing the ring $\mathbf{R}(txn)$ properly. Hence we conclude that the MProve+ protocol does not preserve the untraceability property of Monero in general. But unlike MProve/MProvisions, it does not always reveal the true source address in a source spending transaction.

3.4.2 Effect on the Amount Confidentiality Property of Monero

From Theorem 3.3, it is verified that the MProve+ protocol does not reveal the amounts corresponding to the addresses in the anonymity sets or the total reserves amount. However, the amount confidentiality is affected when a source spending transaction txn appears in the blockchain. As mentioned in the above section, in the extreme scenario when the set $(\mathscr{P}_{\text{orig}}(I) \setminus \mathbf{P}_{\text{other}}) \bigcap \mathbf{R}(txn)$ becomes a singleton set, then the transaction txn becomes traceable. Then the amount confidentiality of the other transactions containing P (the source of txn) in their rings are affected in the same way as described in Section 2.7.2.

Even if the set $(\mathscr{P}_{\text{orig}}(I) \setminus \mathbf{P}_{\text{other}}) \bigcap \mathbf{R}(txn)$ is not a singleton set, its size might be less than that of the ring $\mathbf{R}(txn)$. Then it is known that any of the one-time addresses in the set $\mathbf{R}(txn) \setminus ((\mathscr{P}_{\text{orig}}(I) \setminus \mathbf{P}_{\text{other}}) \bigcap \mathbf{R}(txn))$ cannot be the source of the transaction txn. Then the amount confidentiality of the transaction txn is affected in the way as described in Section 2.7.2. Hence we conclude that the MProve+ protocol does not preserve the

amount confidentiality property of Monero in general.

3.4.3 Effect on the Unlinkability Property of Monero

The unlinkability property of Monero implies that a PPT adversary can link a one-time address with its corresponding public key pair only with a negligible probability (Section 2.2.1). To show that the MProve+ protocol preserves the unlinkability property of Monero, we give the following analysis which is similar to that given in Section 2.7.3.

Let $\{\mathcal{M}_\lambda\}$ denote a sequence of Monero-like systems indexed by the security parameter λ. We consider one such particular system $\mathcal{M}_\lambda$ from the sequence and define the following `MoneroLink` experiment (in the multiplicative notation) to precisely characterize the unlinkability property of Monero.

1. An experimenter chooses some scalars $x_0, y_0, x_1, y_1, r \xleftarrow{\$} \mathbb{Z}_q$. She sets two public key pairs $(X_0 = G^{x_0}, Y_0 = G^{y_0}), (X_1 = G^{x_1}, Y_1 = G^{y_1})$, and a random point $R = G^r$.

2. The experimenter selects a bit $b \xleftarrow{\$} \{0, 1\}$. Then she generates a one-time address $P = G^{H(X_b^r)} \cdot Y_b$.

3. The experimenter sends $(X_0, Y_0, X_1, Y_1, R, P)$ to a PPT adversary $\mathcal{A}$. The adversary $\mathcal{A}$ outputs $\hat{b}$ as a prediction of b. $\mathcal{A}$ wins if $\hat{b} = b$.

Owing to the unlinkability property of Monero, we have the following lemma.

Lemma 3.1. *For every PPT adversary $\mathcal{A}$ in the `MoneroLink` experiment, there exists a negligible function $\mathsf{negl}(\lambda)$ of the security parameter λ such that the following inequality holds.*

$$\left| \Pr[\mathcal{A}(X_0, Y_0, X_1, Y_1, R, P) = b] - \frac{1}{2} \right| \leq \mathsf{negl}(\lambda). \tag{3.40}$$

Next, we propose the following `MPPLink` experiment for the MProve+ protocol.

1. An experimenter chooses some scalars $x_0, y_0, x_1, y_1, r \xleftarrow{\$} \mathbb{Z}_q$. She sets two public key pairs $(X_0 = G^{x_0}, Y_0 = G^{y_0}), (X_1 = G^{x_1}, Y_1 = G^{y_1})$ and a random point $R = G^r$.

2. The experimenter selects a bit $\mathfrak{b} \xleftarrow{\$} \{0, 1\}$. Then she generates a one-time address $P = G^{H_s(X_\mathfrak{b}^r)} \cdot Y_\mathfrak{b}$. The secret key is $x = H_s(X_\mathfrak{b}^r) + y_\mathfrak{b}$.

3. The experimenter produces $f(\lambda)$ MProve+ proofs $\mathsf{MPP}_{\mathsf{act}}$ using the singleton set $\{P\}$ as the anonymity set in all of them. The proofs contain the key image $I = H_p(P)^x$.

4. The experimenter sends $(X_0, Y_0, X_1, Y_1, R, P, \mathsf{MPP}_{\mathsf{act}})$ to a PPT adversary $\mathcal{B}$. $\mathcal{B}$ outputs $\hat{\mathfrak{b}}$ as a prediction of $\mathfrak{b}$. $\mathcal{B}$ wins if $\hat{\mathfrak{b}} = \mathfrak{b}$.

Now we give the following definition.

Definition 3.2. *The MProve+ protocol is said to preserve the unlinkability property of Monero, if for every PPT adversary $\mathcal{B}$ in the* `MPPLink` *experiment, there exists a negligible function $\mathsf{negl}_1(\lambda)$ of the security parameter λ such that the following inequality holds.*

$$\left| \Pr[\mathcal{B}(X_0, Y_0, X_1, Y_1, R, P, \mathit{MPP}_{\mathit{act}}) = \mathfrak{b}] - \frac{1}{2} \right| \leq \mathsf{negl}_1(\lambda). \tag{3.41}$$

We give the following theorem and its proof.

Theorem 3.4. *The MProve+ protocol preserves the unlinkability property of Monero in the random oracle model under the DDH assumption and given that the Lemma 3.1 holds.*

Proof. We prove the theorem by contradiction. Suppose, there exists a PPT adversary $\mathcal{B}$ in the `MPPLink` experiment for which the following inequality holds,

$$\left| \Pr[\mathcal{B}(X_0, Y_0, X_1, Y_1, R, P, \mathsf{MPP}_{\mathsf{act}}) = \mathfrak{b}] - \frac{1}{2} \right| \geq \frac{1}{p(\lambda)}, \tag{3.42}$$

where $p(\lambda)$ is a polynomial of the security parameter λ. Let $\mathsf{MPP}_{\mathsf{sim}}$ denote $f(\lambda)$ simulated proofs which is generated by the method given in Appendix B.2 and using the singleton set $\{P\}$ as the $f(\lambda)$ anonymity sets. From Claim 4 in Appendix B.2, no PPT adversary can distinguish between $\mathsf{MPP}_{\mathsf{act}}$ and $\mathsf{MPP}_{\mathsf{sim}}$ with a probability non-negligibly better than $\frac{1}{2}$. So for the adversary $\mathcal{B}$, there exists another PPT adversary $\mathfrak{B}$ such that the following inequality holds,

$$\left| \Pr[\mathfrak{B}(X_0, Y_0, X_1, Y_1, R, P, \mathsf{MPP}_{\mathsf{sim}}) = \mathfrak{b}] - \frac{1}{2} \right| \geq \frac{1}{p(\lambda)}, \tag{3.43}$$

given that inequality (3.42) is true for $\mathcal{B}$. Next, we construct a PPT adversary $\mathcal{A}$ for the `MoneroLink` experiment using $\mathfrak{B}$ as a subroutine. The construction of $\mathcal{A}(X_0, Y_0, X_1, Y_1, R, P)$ is given below.

1. $\mathcal{A}$ generates $f(\lambda)$ simulated MProve+ proofs $\mathsf{MPP}_{\mathsf{sim}}$ using the singleton set $\{P\}$ as the anonymity set in all of them, following the same steps of the simulator $\mathcal{S}_{\mathsf{MPP}}$ given in Appendix B.2.

2. $\mathcal{A}$ sends $(X_0, Y_0, X_1, Y_1, R, P, \mathsf{MPP}_{\mathsf{sim}})$ to $\mathfrak{B}$ and receives $\hat{\mathfrak{b}}$.

3. It outputs $\hat{\mathfrak{b}}$ as the estimation of b.

Now we have,

$$\begin{aligned} &\left| \Pr[\mathcal{A}(X_0, Y_0, X_1, Y_1, R, P) = b] - \frac{1}{2} \right| \\ &= \left| \Pr[\mathfrak{B}(X_0, Y_0, X_1, Y_1, R, P, \mathsf{MPP}_{\mathsf{sim}}) = b] - \frac{1}{2} \right| \\ &\geq \frac{1}{p(\lambda)}. \end{aligned} \tag{3.44}$$

This contradicts Lemma 3.1. Hence there cannot be a PPT adversary $\mathcal{B}$ for which the inequality (3.42) holds. □

3.5 Performance

We compare our proof of reserves protocol MProve+ with MProve because the latter performs better than MProvisions. In both MProve and MProve+, the anonymity set $\mathbf{P}$ is to be revealed as a part of the proof. Suppose the anonymity set size is n and the number of owned addresses is s. An MProve+ prover needs to compute $12m+2n+2s+2\log_2(m)-2$ group exponentiations and $\mathcal{O}(m)$ point additions and field operations. An MProve+ verifier needs to compute $2m+2n+s+2\log_2(m)+9$ group exponentiations and $\mathcal{O}(m)$ point additions and field operations. Whereas an MProve prover needs to compute $11n$ group exponentiations and $\mathcal{O}(n)$ point additions and field operations. An MProve verifier needs to compute $12n$ group exponentiations and $\mathcal{O}(n)$ point additions and field operations.

The proof sizes of MProve+ and MProve are respectively $(n+s+2\lceil\log_2 m\rceil+4)$ group elements, 5 scalars and $3n+2$ group elements, $6n$ scalars. Here m denotes the length of the witness vectors defined in Figure 3.1. Figure 3.7(a) shows the growth of proof sizes with anonymity set size for $s = 100$. Although the proof sizes of both MProve+ and MProve grow linearly, proof size of MProve+ is typically an order of magnitude smaller. For anonymity set size $n = 10^5$ and the number of owned addresses $s = 10^3$, an MProve+ proof size is 3MB as against 29MB for MProve. The difference in proof sizes increases as n grows. If exchanges are required to publish frequent proofs of reserves and store them for later audits, protocols with smaller proof sizes will be preferred.

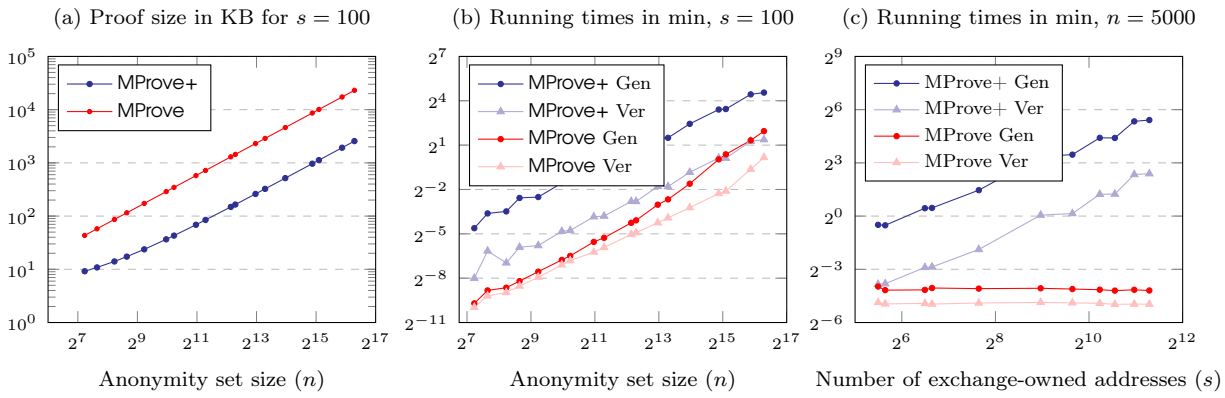

Figure 3.7: Performance comparison of MProve+ and MProve for $\mathbb{G}$ = Ristretto elliptic curve.

We have implemented MProve+ in Rust over the Ristretto elliptic curve. We demonstrate how Ristretto encoding of existing addresses in Monero could be computed, ensuring adaptability to the existing Monero framework [43]. For fair comparison, we have also implemented MProve over Ristretto. All experiments were run on a 2.6 GHz Intel Core i7 desktop with 8GB RAM. Our code is open-sourced on GitHub [44, 45].

Figure 3.7(b) shows the proof generation and verification times of MProve+ and MProve. For a constant s, we see a linear growth of proof generation and verification times of MProve as well as MProve+ with the anonymity set size n. Since the inner product protocol requires witness sizes to be a power of 2, the witness vectors of MProve+ in Figure 3.1 are appended with 0's to convert their size to the next power of 2. For witness sizes $m_1 \neq m_2$ such that $\lceil \log_2 m_1 \rceil = \lceil \log_2 m_2 \rceil$, the timings in the two cases will not differ much. Therefore, we observe a step-wise increment in the generation and verification timings of MProve+. An exchange owning 1000 addresses and wishing to have 49000 cover addresses would spend about 150 minutes in a MProve+ proof generation and the proof verification would take 20 minutes. An MProve proof of same configuration would take a minute for generation and verification each. Although the proof generation time for MProve+ is significantly higher than that of MProve owing to the greater number of group operations, the timings are not unreasonable for practical deployment. The verification of an MProve+ proof is around 8 times faster than its generation because the inner product protocol can be verified using a single multi-exponentiation of size $\mathcal{O}(2s{\cdot}n + 2\log(s{\cdot}n))$ as explained in Section 3.2.6. Faster verification enables customers of an exchange to

verify the proofs without much computational cost and specialized hardware. From the perspective of an exchange, the privacy benefits combined with the smaller proof sizes of MProve+ overshadow the higher computational cost in using it.

A notable difference between the MProve+ and MProve protocol is that in MProve+, we reveal the number of addresses an exchange owns. While this may seem like a privacy concern, an exchange can create some addresses which have zero amount in them for the purpose of *padding* the number of owned addresses. An implication of revealing the number of owned addresses s is that the proof size as well as generation and verification times depend on the number of exchange-owned addresses. Figure 3.7(c) shows the dependence of generation and verification timings of MProve+ and MProve with respect to the number of owned addresses and for a constant anonymity set size. While timings for MProve remain constant, MProve+ timings grow linearly with s.

3.6 Conclusion

We presented the MProve+ protocol which can be used to provide better privacy than the MProve/MProvisions protocols if the exchange is careful while choosing the transaction rings spending from source addresses. The MProve+ protocol provides a significant improvement in terms of proof size over the MProve protocol. The proof generation and verification times of the MProve+ protocol are also practical. The revelation of the key images in the MProve+ protocol leaks information about the exchange-owned addresses. Designing a proof of reserves protocol for Monero which does not explicitly reveal key images of source addresses remains an open problem.

Chapter 4

Revelio: A MimbleWimble Proof of Reserves Protocol

In this chapter, we propose a proof of reserves protocol for the MimbleWimble [8,9] cryptocurrency scheme. Grin [20] and Beam [21] are cryptocurrencies which are built on the MimbleWimble proposal. To clarify the design of MimbleWimble, we give a brief overview of Grin covering the structure of outputs and transactions, the transaction validation procedure, and the interactive transaction construction procedure. Then we describe non-interactive zero-knowledge proof of knowledge signatures formalized by Camenisch and Stadler [46, 47]. These signatures serve as the building blocks for the proof of reserves protocols in this chapter as well as in the next chapter. Then we present Revelio, the first proof of reserves protocol for MimbleWimble providing privacy to the exchange.

4.1 Overview of Grin

Grin is a cryptocurrency which became operational on January 15$^{\text{th}}$, 2019 [20]. It is a permissionless blockchain which uses proof-of-work (PoW) consensus to resolve forks. Being an implementation of the MimbleWimble protocol, Grin does not have addresses. To transfer coins, the sender and receiver have to interactively build a transaction. This is unlike other cryptocurrencies (like Bitcoin) where a sender needs only the receiver's address to transfer coins. However, the sender and receiver are not required to be online at the same time to build a transaction. The required interaction can be performed asynchronously via media like email.

4.1.1 Outputs

Coins are stored in *outputs* each of which consists of three fields:

1. A flag marking the output as a *coinbase output* or a *plain output*.
2. A Pedersen commitment $C = kG + vH$ where
   - k and v are scalars in the prime field $\mathbb{Z}_n$ with n equal to the order of the group in the secp256k1 curve [48]. The scalar v represents the amount stored in the output. The scalar k is chosen randomly to conceal the amount and is called the *blinding factor*.
   - G is the base point of the secp256k1 curve.
   - H is another point on the secp256k1 curve whose discrete logarithm with respect to G is not known. It is obtained as the SHA256 hash of an encoding of the point G [49].
3. A range proof which proves that the value v stored in the commitment C is in the range $\{0, 1, \ldots, 2^{64} - 1\}$*.

Coinbase outputs are included by miners to store the mining reward and fees they earn during block creation. All the non-coinbase outputs are marked as plain outputs.

Grin does not permit the existence of multiple outputs containing the same commitment C. Hence it is convenient to identify an output by the commitment it contains.

4.1.2 Transaction Fields

A Grin transaction consists of a scalar $k_{\text{off}} \in \mathbb{Z}_n$ called the *kernel offset* and a *transaction body*. The transaction body itself consists of three fields:

1. A vector of *inputs* where each input contains the Pedersen commitment of an unspent output. These inputs will be the sources of coins in the transaction.
2. A vector of *outputs* representing the destinations of coins in the transaction.

*As in Monero, the maximum number of indivisible units of the Grin currency which can come into existence is $2^{64} - 1$.

3. A vector of *transaction kernels*[†].

Transaction kernels are used to verify the validity of the transaction. Each kernel consists of five fields:

1. A *kernel features* flag to indicate whether the kernel is a plain kernel, a coinbase kernel, or a height-locked kernel.
    - Coinbase kernels are used to validate the coinbase outputs included by miners in blocks.
    - Plain kernels are used to validate the inputs and plain outputs in a transaction or a block.
    - Height-locked kernels are similar to plain kernels but have a non-zero lock height (defined below).
2. A *fee* represented by a 64-bit unsigned integer.
3. A *lock height* represented by a 64-bit unsigned integer.
    - A kernel with a non-zero lock height is included in a block only if the block height is at least as large as the lock height. This effectively prevents a transaction containing a height-locked kernel from appearing in blocks on the blockchain whose height is less than the lock height.
4. A *kernel excess* represented by a point on the secp256k1 curve. It is of the form xG where G is the base point of the secp256k1 curve and $x \in \mathbb{Z}_n$.
5. A Schnorr *signature* which can be verified by using the kernel excess as a public key.
    - The message used to generate this signature depends on the kernel type.
        - For coinbase kernels, the message is the hash of the kernel features flag.
        - For plain kernels, it is the hash of the concatenation of the kernel features flag and the fee.
        - For height-locked kernels, it is the hash of the concatentation of the kernel features flag, the fee, and the lock height.

[†]Note that the structure of the transaction kernel is not used in the design of Revelio. The details are presented here only for the sake of completeness.

4.1.3 Transaction Validation

Transaction validation proceeds as follows:

1. The transaction body is validated using the following rules.

   (a) Each output flag is checked to ensure that the output is a plain output. Coinbase outputs can appear only as part of a block and not as part of a standalone transaction.

   (b) The range proof associated with each output commitment is verified.

   (c) Each kernel features flag is checked to ensure that the kernel is not a coinbase kernel.

   (d) If the kernel is a plain kernel, it is checked to ensure that it has a zero lock height.

   (e) The signature in each kernel is verified using the kernel excess as the public key and the message generated according to the kernel type.

   This step ensures that the kernel excess is of the form xG and not of the form $xG + vH$ for some non-zero scalar v. If the kernel were of the latter form, the signature creation would require knowledge of the scalar y such that $yG = xG + vH$. This in turn would imply knowledge of the discrete logarithm of H with respect to G.

2. To ensure that the sum of the input amounts equals the sum of the output amounts and fees, the following check is performed.

   (a) Suppose the transaction has L inputs, M outputs, and N kernels.

   (b) Let the input commitments be $C_1^{\text{in}}, C_2^{\text{in}}, \ldots, C_L^{\text{in}}$, the output commitments be $C_1^{\text{out}}, C_2^{\text{out}}, \ldots, C_M^{\text{out}}$, and the kernel excesses be $X_1, X_2, \ldots, X_N$.

   (c) Let the kernel offset of the transaction be $k_{\text{off}} \in \mathbb{Z}_n$ and the fee be $f \in \mathbb{Z}_n$.

   (d) The following equality is checked.

$$\sum_{i=1}^{M} C_i^{\text{out}} + fH - \sum_{i=1}^{L} C_i^{\text{in}} = \sum_{i=1}^{N} X_i + k_{\text{off}}G. \tag{4.1}$$

To see why equality in equation (4.1) is sufficient to ensure the input and output amounts in the transaction balance out, let $C_i^{\text{in}} = k_i^{\text{in}}G + v_i^{\text{in}}H$, $C_i^{\text{out}} = k_i^{\text{out}}G + v_i^{\text{out}}H$, and $X_i = x_iG$. The equality in equation (4.1) will hold if the following equalities hold:

$$\sum_{i=1}^{L} v_i^{\text{in}} = \sum_{i=1}^{M} v_i^{\text{out}} + f, \tag{4.2}$$

$$\sum_{i=1}^{M} k_i^{\text{out}} - \sum_{i=1}^{L} k_i^{\text{in}} = \sum_{i=1}^{N} x_i + k_{\text{off}}. \tag{4.3}$$

The equality in equation (4.2) implies that the sum of the input amounts is equal to the sum of the output amounts and the fee. These terms appear as the multipliers of the H point in equation (4.1). The equality in equation (4.3) implies that the multipliers of the G point in equation (4.1) balance out. If the equality in equation (4.1) were satisfied *without* the equality in equation (4.2) being satisfied, then we would have the relation

$$\left(\sum_{i=1}^{M} v_i^{\text{out}} + f - \sum_{i=1}^{L} v_i^{\text{in}}\right) H = \left(-\sum_{i=1}^{M} k_i^{\text{out}} + \sum_{i=1}^{L} k_i^{\text{in}} + \sum_{i=1}^{N} x_i + k_{\text{off}}\right) G. \tag{4.4}$$

As the multiplier of H in the above equation is non-zero, multiplying on both sides by its multiplicative inverse in the field $\mathbb{Z}_n$ will give the discrete logarithm of H with respect to G.

The reason for including the kernel offset k_{off} in the transaction is to hide the relationship between the transaction's inputs and outputs after it is included in a block along with other transactions. When transactions are aggregated into a block, their respective kernel offsets are added and included in the block header as a *total kernel offset*. All the inputs, outputs, and kernels from all the transactions are stored in the block as sorted lists. In fact, a Grin block looks like one big transaction with a block header attached. There is no indication of which inputs and outputs originally appeared as part of a single transaction. If there were no kernel offset in a transaction, an adversary could attempt to deconstruct the individual transactions by trying combinations of inputs, outputs, and kernels which satisfy the transaction validity equation. With a total kernel offset resulting from at least two transaction kernel offsets, this strategy cannot be used.

Note that the check which ensures that the transaction inputs refer to unspent outputs did not appear as part of the transaction validation. This check is performed during the block validation.

4.1.4 Interactive Transaction Construction

To spend from an output, the spender needs to know the value v and the blinding factor k used to generate the output commitment. Suppose Alice knows the value and blinding factor in an unspent output commitment $C_{\text{in}} = k_A G + v_A H$. She wants to send v_B coins to Bob where $v_B < v_A$. For a transaction fee f, she wants the remaining $v_A - v_B - f$ coins to be stored in a change output with commitment $C_{\text{chg}} = k_C G + (v_A - v_B - f)H$ where the blinding factor k_C is known only to her. Alice and Bob will collaborate to construct a transaction which will have C_{in} as input and two outputs C_{chg} and $C_{\text{out}} = k_B G + v_B H$ where the blinding factor k_B is known only to Bob.

Alice and Bob will exchange a data structure called a *slate* which contains a transaction that will be progressively populated during the interaction to yield the final transaction. The transaction construction consists primarily of the three steps detailed below.

1. In the first step, Alice will initialize the slate and send it to Bob. She performs the following steps:

   (a) She adds the input being spent C_{in} to the transaction in the slate.

   (b) She adds the amount being transferred v_B to the slate.

   (c) She sets the lock height of the transaction in the slate to be equal to the current height h of the blockchain.

   (d) She calculates the transaction fee f and adds it to the slate.

   - The transaction fee in Grin is currently required to be at least *transaction weight* milligrins. For a transaction with L inputs, M outputs, and N kernels, the transaction weight is

   $$\max\left(4M + N - L, 1\right). \tag{4.5}$$

   In our example, the transaction weight is 8 as $L = 1, M = 2$, and $N = 1$.

   (e) She chooses the change output blinding factor k_C uniformly from $\mathbb{Z}_n$ and calculates the change output commitment $C_{\text{chg}} = k_C G + (v_A - v_B - f)H$. This output is added to the transaction in the slate along with a range proof.

   (f) She chooses the kernel offset k_{off} uniformly from $\mathbb{Z}_n$ and calculates the *sender kernel excess secret key* as $k'_A = k_C - k_A - k_{\text{off}}$. The kernel offset k_{off} and the *sender kernel excess* $X_A = k'_A G$ are added to the transaction in the slate.

(g) She chooses a random nonce r_A uniformly from $\mathbb{Z}_n$ and adds the nonce public key $R_A = r_A G$ to the slate.

To summarize, the initial slate sent by Alice to Bob contains $C_{\text{in}}, C_{\text{out}}, v_B, h, f, X_A, R_A$, and k_{off}.

2. In the second step, Bob will add a receiver kernel excess, his own nonce public key, and a Schnorr signature to the slate before returning it to Alice. He performs the following steps:

   (a) He chooses his output blinding factor k_B uniformly from $\mathbb{Z}_n$ and calculates the output commitment $C_{\text{out}} = k_B G + v_B H$ using the amount v_B from the slate. This output is added to the transaction in the slate along with a range proof.

   (b) He calculates the *receiver kernel excess* $X_B = k_B G$ and adds it to the slate. So the output commitment blinding factor and the receiver kernel excess secret key are the same.

   (c) He chooses a random nonce r_B uniformly from $\mathbb{Z}_n$ and adds the nonce public key $R_B = r_B G$ to the slate.

   (d) Let $\|$ denote the concatenation operator. Bob calculates the receiver Schnorr signature on the message $m = f\|h$ as (s_B, R_B) where $s_B = r_B + ek_B$. The scalar $e \in \mathbb{Z}_n$ is obtained as

   $$e = \text{SHA256}(R_A + R_B \| X_A + X_B \| m). \tag{4.6}$$

   He adds the signature to the slate. It can be verified using the public key X_B.

3. Upon receiving the slate from Bob, Alice completes the transaction construction as follows:

   (a) She verifies Bob's signature (s_B, R_B) by checking the equality

   $$s_B G = R_B + eX_B, \tag{4.7}$$

   where e is calculated as shown in equation (4.6).

   (b) She calculates the sender Schnorr signature (s_A, R_A) on the same message m as $s_A = r_A + ek'_A$ where r_A is her random nonce and k'_A is the sender kernel excess secret key.

(c) She sets the transaction kernel excess to be equal to $X_A + X_B$.

(d) She sets the signature in the transaction kernel to be equal to $(s_A + s_B, R_A + R_B)$.

After the third step, Alice can broadcast the transaction containing the kernel offset k_{off}, the input commitment C_{in}, the output commitments C_{out} and C_{chg}, and the transaction kernel. The kernel itself consists of the fee f, the lock height h, the kernel excess $X_A + X_B$, and the signature $(s_A + s_B, R_A + R_B)$. This transaction satisfies the condition given in equation (4.1) as

$$\begin{aligned} & C_{\text{out}} + C_{\text{chg}} + fH - C_{\text{in}} \\ &= k_B G + v_B H + k_C G + (v_A - v_B - f)H + fH \\ &\quad - k_A G - v_A H \\ &= k_B G + (k_C - k_A)G = k_B G + (k_C - k_A - k_{\text{off}})G + k_{\text{off}} G \\ &= k_B G + k'_A G + k_{\text{off}} G = X_B + X_A + k_{\text{off}} G. \end{aligned} \tag{4.8}$$

During the interaction, Alice does not learn Bob's output blinding factor k_B as he only reveals the point $X_B = k_B G$. But she does know the amount v_B in the output commitment C_{out}. On the other hand, Bob learns neither Alice's output blinding factor k'_A nor the change amount $v_A - v_B - f$ in the commitment C_{chg}.

4.1.5 Blocks

Blocks in the Grin protocol consist of a *block header* and a transaction body which has the same structure as the body of a regular transaction. The block header has fields specifying the block height, timestamp, and PoW details. Additionally, the header has the *total kernel offset* which is obtained as the sum of the kernel offsets of all transactions which are part of the block. The block's transaction body is obtained by merging the lists of inputs, outputs, and kernels from the constituent transactions. If any commitments in the output list also appear in the input list, they are removed as they are not required for the block validation (see the left hand side of equation (4.1)). This process is called *cut through*. The same cut through process can be applied to outputs appearing in old blocks after they are spent by transactions in later blocks. This increases scalability of the Grin

blockchain requiring network nodes to only keep track of commitments corresponding to unspent outputs.

The following section gives some signature primitives which are used to the construct the Revelio protocol.

4.2 Signatures Proving Statements about Discrete Logarithms

Let $\mathbb{G}$ be a cyclic group of prime order n where solving the discrete logarithm problem is assumed to be hard. Schnorr [50] proposed a signature scheme proving knowledge of a secret $x \in \mathbb{Z}_n$ such that $X = xG$. This is a proof of knowledge of the representation of X with respect to the generator G. It is a special case of non-interactive zero-knowledge proof of knowledge (NIZKPoK) signatures which were formalized by Camenisch and Stadler [46, 47]. Below we give two kinds of NIZKPoK signatures which are used as building blocks in both Revelio and Nummatus (Chapter 5).

Let G, H, G' be generators of the group G such that the discrete logarithm relations between them are unknown. Suppose a prover Prov wants to prove knowledge of scalars (α, β) to a verifier Verf such that the following statement holds.

$$X = \alpha G + \beta H \ \wedge \ Y = \alpha G' + \beta H. \tag{4.9}$$

The NIZKPoK signature for proving conjunctive statements like in equation (4.9) is defined below.

Definition 4.1. *A triple of scalars $(c, s_1, s_2) \in \mathbb{Z}_n^3$ is a NIZKPoK signature and gives representations of elements $X, Y \in \mathbb{G}$ with respect to the generator pairs G, H and G', H respectively, if they satisfy*

$$c = \mathcal{H}\left(S, s_1 G + s_2 H + cX, s_1 G' + s_2 H + cY\right), \tag{4.10}$$

where $S = G \| G' \| H \| X \| Y$ and $\mathcal{H}$ is a strong collision resistant hash function [47, Section 3.2.1]. Such a triple will be denoted by

$$PoK\{(\alpha, \beta) \mid X = \alpha G + \beta H \ \wedge \ Y = \alpha G' + \beta H\}.$$

Prov with the knowledge of (α, β) can generate such a proof as follows:

- She chooses $r_1, r_2 \xleftarrow{\$} \mathbb{Z}_n$ randomly from $\mathbb{Z}_n$ and calculates

$$c = \mathscr{H}\left(S, r_1 G + r_2 H, r_1 G' + r_2 H\right).$$

- She sets s_1 and s_2 as

$$s_1 = r_1 - c\alpha,$$
$$s_2 = r_2 - c\beta.$$

Here S might be considered as the message m which contains all the group elements involved in the signature. To verify the signature, Verf needs to check the equality given in equation (4.10) using the signature (c, s_1, s_2). Equation (4.10) holds because of the following relations

$$\begin{aligned} s_1 G + s_2 H + cX &= r_1 G + r_2 H - c(\alpha G + \beta H) + cX \\ &= r_1 G + r_2 H, \\ s_1 G' + s_2 H + cY &= r_1 G' + r_2 H - c(\alpha G' + \beta H) + cY \\ &= r_1 G' + r_2 H. \end{aligned}$$

Now we present a signature scheme which proves disjunctions of statements about discrete logarithms. The main idea behind the following construction was first proposed by Cramer et al. [51]. In particular, for given group elements $X, Y \in \mathbb{G}$, Prov wants to prove knowledge of (α, β, γ) such that either $X = \alpha G + \beta H \ \wedge \ Y = \alpha G' + \beta H$ or $Y = \gamma G'$ i.e. the following statement holds,

$$\left(X = \alpha G + \beta H \ \wedge \ Y = \alpha G' + \beta H\right) \vee \left(Y = \gamma G'\right). \tag{4.11}$$

The NIZKPoK signature for proving disjunctive statements like in equation (4.11) is defined below.

Definition 4.2. *A 5-tuple of scalars $(c_1, c_2, s_1, s_2, s_3) \in \mathbb{Z}_n^5$ is a NIZKPoK signature of* ***either***

1. *the knowledge and equality of the representations of elements $X, Y \in \mathbb{G}$ with respect to the generator pairs G, H and G', H respectively,* ***or***

2. *the knowledge of the discrete logarithm of the element* $Y \in \mathbb{G}$ *with respect to the generator* G',

if they satisfy

$$c_1 + c_2 = \mathcal{H}(S, V_1, V_2, V_3) \tag{4.12}$$

where $S = G\|G'\|H\|X\|Y$ *and*

$$\begin{aligned} V_1 &= s_1 G + s_2 H + c_1 X, \\ V_2 &= s_1 G' + s_2 H + c_1 Y, \\ V_3 &= s_3 G' + c_2 Y. \end{aligned} \tag{4.13}$$

Such a 5-tuple will be denoted by

$$PoK\{(\alpha, \beta, \gamma) \mid (X = \alpha G + \beta H \ \wedge \ Y = \alpha G' + \beta H) \vee (Y = \gamma G')\}.$$

Suppose Prov knows only γ such that $Y = \gamma G'$. Then she can generate a proof as follows:

- She chooses $r_3, c_1, s_1, s_2 \xleftarrow{\$} \mathbb{Z}_n$ randomly from $\mathbb{Z}_n$ and calculates c_2 as

$$c_2 = \mathcal{H}(S, V_1, V_2, r_3 G') - c_1, \tag{4.14}$$

  where $S = G\|G'\|H\|X\|Y$, $V_1 = s_1 G + s_2 H + c_1 X$, and $V_2 = s_1 G' + s_2 H + c_1 Y$.

- She sets s_3 as

$$s_3 = r_3 - c_2 \gamma. \tag{4.15}$$

This is similar to Schnorr's signature scheme. Now consider the case when Prov knows α, β (not γ) such that $X = \alpha G + \beta H$ and $Y = \alpha G' + \beta H$. She can generate a proof as follows:

- She chooses r_1, r_2, c_2, s_3 randomly from $\mathbb{Z}_n$ and calculates c_1 as

$$c_1 = \mathcal{H}(S, r_1 G + r_2 H, r_1 G' + r_2 H, V_3) - c_2, \tag{4.16}$$

  where $S = G\|G'\|H\|X\|Y$ and $V_3 = s_3 G' + c_2 Y$.

- She sets s_1 and s_2 as

$$s_1 = r_1 - c_1\alpha,$$
$$s_2 = r_2 - c_1\beta. \tag{4.17}$$

This is similar to the signature generation procedure followed in Definition 4.1.

To verify the signature, Verf needs to verify the equality in equation (4.12) with the signature $(c_1, c_2, s_1, s_2, s_3)$. The message S is imbibed in the signature as discussed before. In this way Prov can prove knowledge of either (α, β) or γ such that equation (4.11) satisfies. Also Verf does not get to know whether Prov knows (α, β) or γ.

4.3 Revelio Proof of Reserves Protocol

In a MimbleWimble-based cryptocurrency like Grin, coins are stored in outputs which contain a Pedersen commitment $C = kG+vH$ where k and v are scalars in $\mathbb{Z}_n$ representing the blinding factor and amount respectively. Knowledge of both these scalars implies ownership of the output. We will describe the Revelio protocol in the context of Grin. So G is the base point of the secp256k1 curve and H is another point on the secp256k1 curve having unknown discrete logarithm relationship with respect to G. The output also has a range proof which proves that the amount v stored in the commitment C is in the range $\{0, 1, \ldots, 2^{64} - 1\}$.

Like the proof of reserves protocols in the previous chapters, the Revelio protocol will output a Pedersen commitment C_{res} to an amount which is equal to the number of coins owned by the exchange. Given a Pedersen commitment C_{liab} to the total liabilities of the exchange, it can prove solvency via a range proof which shows that the amount committed to in $C_{\text{res}} - C_{\text{liab}}$ is non-negative.

Let $\mathcal{C}_{\text{unspent}}$ be the set of all unspent outputs on the Grin blockchain. Let $\mathcal{C}_{\text{own}}$ be the set of unspent outputs owned by the exchange. The exchange will choose a subset $\mathcal{C}_{\text{anon}}$ of $\mathcal{C}_{\text{unspent}}$ such that $\mathcal{C}_{\text{own}} \subseteq \mathcal{C}_{\text{anon}}$. The set $\mathcal{C}_{\text{anon}}$ represents the *anonymity set* of unspent outputs which contains the unspent outputs actually owned by the exchange. This anonymity set will be revealed in the Revelio protocol. But an observer will not be able to distinguish between members of $\mathcal{C}_{\text{anon}}$ which belong to $\mathcal{C}_{\text{own}}$ from those which do not.

If an exchange does not care about revealing the identity of the unspent outputs it owns, then there is no need for the Revelio protocol. The exchange can simply reveal $\mathcal{C}_{\text{own}}$ and generate C_{res} as

$$C_{\text{res}} = \sum_{C \in \mathcal{C}_{\text{own}}} C. \tag{4.18}$$

It can then prove knowledge of scalars k and v such that $C_{\text{res}} = kG + vH$ by giving a proof of knowledge of the representation of C_{res} with respect to generators G and H. We compare the performance of Revelio to this simple non-private protocol in Section 4.5.

4.3.1 Proof Generation

Let G' be another generator of the secp256k1 curve whose discrete logarithms with respect to G and H are not known. Let $\mathcal{H}$ be equal to the SHA256 hash function.

The Revelio proof of reserves protocol proceeds as follows:

1. The exchange chooses a long-term secret key[‡] k_{exch} uniformly from $\mathbb{Z}_n$. This key must remain the same in all the Revelio proofs generated by the exchange, i.e. this key is chosen when the exchange generates a Revelio proof for the first time and subsequently remains unchanged.

2. The exchange chooses a list of unspent outputs $\mathcal{C}_{\text{anon}} = (C_1, C_2, \ldots, C_N)$ from the Grin blockchain such that it owns the outputs in a subset $\mathcal{C}_{\text{own}}$ of $\mathcal{C}_{\text{anon}}$. The list $\mathcal{C}_{\text{anon}}$ is made public by the exchange. Ownership of an output $C_i \in \mathcal{C}_{\text{own}}$ is equivalent to knowledge of scalars k_i and v_i such that $C_i = k_i G + v_i H$. For $C_i \in \mathcal{C}_{\text{anon}} \setminus \mathcal{C}_{\text{own}}$, the exchange may know the value v_i if it was the sender in the transaction which created C_i. But it does not know the blinding factor k_i for such an output.

3. For each $C_i \in \mathcal{C}_{\text{anon}}$ such that $C_i = k_i G + v_i H$, the exchange generates a curve point I_i as

$$I_i = \begin{cases} k_i G' + v_i H & \text{if } C_i \in \mathcal{C}_{\text{own}}, \\ y_i G' & \text{if } C_i \notin \mathcal{C}_{\text{own}}, \end{cases} \tag{4.19}$$

where $y_i = \mathcal{H}(k_{\text{exch}}, C_i)$. The points $(I_1, I_2, \ldots, I_N)$ are published by the exchange. Note that the I_is are a deterministic function of the respective C_is (for a fixed long-term secret key k_{exch}).

[‡]The long term secret key is needed for reasons similar to those described in Chapter 2.

4. For each $i = 1, 2, \ldots, N$, the exchange uses the method described in Section 4.2 to generate a NIZKPoK $\sigma_i = (c_1^i, c_2^i, s_1^i, s_2^i, s_3^i)$ of the form

$$\text{PoK}\left\{(\alpha, \beta, \gamma) \mid (C_i = \alpha G + \beta H \ \wedge \ I_i = \alpha G' + \beta H) \vee (I_i = \gamma G')\right\}.$$

   The proofs $(\sigma_1, \sigma_2, \ldots, \sigma_N)$ are published by the exchange.

5. The exchange claims that the commitment C_{res} given by

$$C_{\text{res}} = \sum_{i=1}^{N} I_i \tag{4.20}$$

   is a Pedersen commitment of the form $x_{\text{tot}} G' + v_{\text{tot}} H$ where v_{tot} is the total amount of Grin it owns.

Note that the blinding factor x_{tot} is multiplying G' and not G in the commitment C_{res}. The proof of solvency needs to take this into account by generating the C_{liab} commitment to have the form $yG' + v_{\text{liab}} H$.

The intuition behind the protocol construction is as follows. The NIZKPoK σ_i proves that if the exchange does not own the output C_i then I_i is a commitment to the zero amount. Furthermore, it proves that in case the exchange does own the output C_i it can generate I_i as a commitment *only* to the amount v_i which is committed to by C_i. These two properties of I_i force the C_{res} calculated in equation (4.20) to be a commitment to an amount which is *at most* equal to the total amount of Grin owned by the exchange. C_{res} is not necessarily a commitment to an amount equal to the total amount owned by the exchange because the exchange can choose I_i to be a commitment to the zero amount in spite of owning C_i.

The above argument does not clarify the need for introducing the generator G' or the need for setting y_i equal to $\mathcal{H}(k_{\text{exch}}, C_i)$. The former is needed to detect collusion between exchanges while the latter is needed to prevent identification of outputs belonging to the exchange.

Let us consider the need for G' first. One can get the same guarantees regarding C_{res} by defining I_i as follows for $C_i = k_i G + v_i H$.

$$I_i = \begin{cases} x_i G + v_i H & \text{if } C_i \in \mathcal{C}_{\text{own}}, \\ y_i G & \text{if } C_i \notin \mathcal{C}_{\text{own}}, \end{cases} \tag{4.21}$$

where the x_is are chosen uniformly from $\mathbb{Z}_n$ and $y_i \in \mathbb{Z}_n$. Note that for $C_i \in \mathcal{C}_{\text{own}}$ the blinding factor in I_i is different from the blinding factor in C_i but the amounts in both commitments are the same. One can prove that I_i has this structure by giving a NIZKPoK of the form

$$\text{PoK}\left\{(\alpha, \beta, \gamma, \delta) \mid (C_i = \alpha G + \beta H \ \wedge \ I_i = \delta G + \beta H) \vee (I_i = \gamma G)\right\}.$$

While this definition of I_i guarantees that the C_{res} calculated in equation (4.20) has the right properties, it does not prevent collusion between exchanges. Two exchanges could share an output C_i to generate their respective asset proofs without being detected as the blinding factors x_i in equation (4.21) can be chosen freely. By forcing I_i to have the structure given in equation (4.19), we are ensuring that I_i is a deterministic function of C_i for $C_i \in \mathcal{C}_{\text{own}}$. Consequently, it plays the role of a *key image* of C_i which enables detection of collusion between exchanges. In case the same I_i appears in the proofs of reserves of two different exchanges, collusion between them is revealed. This structure of I_i is the main innovation in the Revelio protocol.

Finally, the reason for setting $y_i = \mathcal{H}(k_{\text{exch}}, C_i)$ is to make I_i a deterministic function of C_i even when $C_i \notin \mathcal{C}_{\text{own}}$, without revealing the non-membership of C_i in $\mathcal{C}_{\text{own}}$. Suppose I_i was not a deterministic function of C_i for $C_i \notin \mathcal{C}_{\text{own}}$. For example, the y_is could simply be chosen uniformly from $\mathbb{Z}_n$ every time an exchange generated a Revelio proof (the y_i values in the current Revelio proof will be independent of the y_i values in the previous Revelio proofs). It is realistic to assume that a proof of reserves protocol will need to be executed multiple times by an exchange. In this scenario, the I_i points corresponding to $C_i \in \mathcal{C}_{\text{anon}} \setminus \mathcal{C}_{\text{own}}$ will keep changing in each Revelio proof while the I_i points corresponding to $C_i \in \mathcal{C}_{\text{own}}$ will remain the same. Outputs C_i which appear in multiple Revelio proofs of an exchange with the same I_i will be identified as outputs belonging to $\mathcal{C}_{\text{own}}$. Outputs C_i which appear in multiple Revelio proofs with different I_i will be identified as outputs not belonging to $\mathcal{C}_{\text{own}}$.[§] Of course, the method of making the y_is a deterministic function of C_i using $\mathcal{H}$ and k_{exch} is largely a matter of convenience. An exchange could also implement such a function using a lookup table where the y_is are chosen uniformly and independently from $\mathbb{Z}_n$ once and reused in all subsequent Revelio proofs.

[§]Our earlier proposal had this flaw. One of the anonymous reviewers of our CVCBT 2019 paper submission identified this flaw and suggested the fix.

4.3.2 Proof Verification

The output of an exchange in the Revelio protocol consists of the following:

- The list $\mathcal{C}_{\text{anon}} = (C_1, C_2, \ldots, C_N)$.
- The key image commitment list $(I_1, I_2, \ldots, I_N)$.
- The proofs $\sigma_i = (c_1^i, c_2^i, s_1^i, s_2^i, s_3^i)$ for $i = 1, 2, \ldots, N$.

Verification involves the following operations:

1. The verifier checks that the list $\mathcal{C}_{\text{anon}}$ consists of only unspent outputs.
2. For each $i = 1, 2, \ldots, N$, the verifier verifies the proof σ_i using the pair (C_i, I_i).
3. The verifier also checks that none of the key image commitments I_i published by the exchange appear in the proofs of reserves published by other exchanges. If a key image commitment is common to the proofs published by two different exchanges, collusion is declared.

4.4 Security Properties

The Revelio protocol provides three main security properties, namely, *inflation resistance*, *collusion resistance*, and *privacy*. The inflation resistance property prevents a probabilistic polynomial time (PPT) exchange from generating a commitment C_{res} to an amount which exceeds its total reserves. The collusion resistance property prevents two or more exchanges from sharing a common output as source while generating the reserves proofs. The privacy property ensures that a PPT adversary can extract some secret information of the exchange from a polynomial number of Revelio proofs only with a negligible probability.

4.4.1 Inflation Resistance

In order to generate a C_{res} commitment to an amount which is greater than the total amount it owns, an exchange would have to create an I_i which is either a commitment to the amount in C_i when it does not own C_i or a commitment to an amount larger than the

amount in C_i. But this would be a forgery of the NIZKPoK proof σ_i. So a PPT exchange can possibly achieve asset inflation with only a negligible probability of success.

4.4.2 Collusion Resistance

When I_i is a commitment to the amount in C_i, the NIZKPoK proof σ_i guarantees that I_i is a deterministic function of C_i except with negligible probability. This ensures that collusion between exchanges (via output sharing) is detected.

4.4.3 Privacy

We consider the scenario when a MimbleWimble exchange publishes $f(\lambda)$ Revelio proofs where $f(\lambda)$ denotes a polynomial of the security parameter λ. Let the size of the ith anonymity set $\mathbf{C}^{(i)}_{\text{anon}}$ be n_i. We denote the ith Revelio proof by $\left(\mathbf{C}^{(i)}_{\text{anon}}, \mathbf{I}^{(i)}, \mathbf{\Sigma}^{(i)}\right)$ where,

$$\mathbf{C}^{(i)}_{\text{anon}} = (C_{i,1}, C_{i,2}, \ldots, C_{i,n_i}), \tag{4.22}$$

$$\mathbf{I}^{(i)} = (I_{i,1}, I_{i,2}, \ldots, I_{i,n_i}), \tag{4.23}$$

$$\mathbf{\Sigma}^{(i)} = (\sigma_{i,1}, \sigma_{i,2}, \ldots, \sigma_{i,n_i}), \tag{4.24}$$

are the vectors of outputs, key images, and NIZKPoK signatures respectively. Let the $f(\lambda)$ Revelio proofs be denoted by $\mathsf{Rev}_{\mathsf{act}} = \left(\mathbf{C}^{(i)}_{\text{anon}}, \mathbf{I}^{(i)}, \mathbf{\Sigma}^{(i)}\right)_{i=1}^{f(\lambda)}$. We want to check if $\mathsf{Rev}_{\mathsf{act}}$ leaks some information about the exchange like which outputs in $\left(\mathbf{C}^{(i)}_{\text{anon}}\right)_{i=1}^{f(\lambda)}$ are owned by the exchange, their amounts, or the total reserves amount. To show that $\mathsf{Rev}_{\mathsf{act}}$ reveals no such information, we take an approach similar to that has been taken in the previous chapters. In particular, we construct a simulator $\mathcal{S}_{\mathsf{Rev}}$ that has access to $\mathsf{Rev}_{\mathsf{act}}$ but does not know any secret information of the exchange. The simulator proceeds as follows.

She chooses a long term key $k_{\text{sim}} \xleftarrow{\$} \mathbb{Z}_n$. To construct the ith simulated Revelio proof, she reads the ith Revelio proof $\left(\mathbf{C}^{(i)}_{\text{anon}}, \mathbf{I}^{(i)}, \mathbf{\Sigma}^{(i)}\right)$. In the ith simulated proof, she keeps $\mathbf{C}^{(i)}_{\text{anon}}$ as it is and changes the other elements in the following manner. Let $[n_i]$ denote the set $\{1, 2, \ldots, n_i\}$. She sets $\hat{I}_{i,k} = q_{i,k}G'$, where $q_{i,k} = \mathcal{H}(k_{\text{sim}}, C_{i,k})$ for all $k \in [n_i]$. She computes the simulated NIZKPoK proof $\hat{\sigma}_{i,k}$ using the knowledge of $q_{i,k}$ for all $k \in [n_i]$. She constructs the vectors $\hat{\mathbf{I}}^{(i)}, \hat{\mathbf{\Sigma}}^{(i)}$ with the computed quantities $\hat{I}_{i,k}$ and $\hat{\sigma}_{i,k}$ respectively for all $k \in [n_i]$. She sets $\mathsf{Rev}_{\mathsf{sim}} = \left(\mathbf{C}^{(i)}_{\text{anon}}, \hat{\mathbf{I}}^{(i)}, \hat{\mathbf{\Sigma}}^{(i)}\right)_{i=1}^{f(\lambda)}$.

If no PPT distinguisher $\mathcal{D}_{\mathsf{Rev}}$ can distinguish between $\mathsf{Rev}_{\mathsf{act}}$ and $\mathsf{Rev}_{\mathsf{sim}}$ except with a probability negligibly better than that of random guessing, then we can say $\mathsf{Rev}_{\mathsf{act}}$ does not reveal any information about the exchange. In particular, we define the privacy experiment `RevPriv` for the Revelio scheme as follows.

1. $\mathcal{S}_{\mathsf{Rev}}$ sets $\mathsf{Rev}_0 = \mathsf{Rev}_{\mathsf{sim}}$ and $\mathsf{Rev}_1 = \mathsf{Rev}_{\mathsf{act}}$.
2. $\mathcal{S}_{\mathsf{Rev}}$ chooses a bit $b \xleftarrow{\$} \{0, 1\}$ randomly.
3. $\mathcal{S}_{\mathsf{Rev}}$ sends Rev_b to $\mathcal{D}_{\mathsf{Rev}}$.
4. $\mathcal{D}_{\mathsf{Rev}}$ outputs a bit $\mathcal{D}_{\mathsf{Rev}}(\mathsf{Rev}_b)$ as a prediction of b.

Now we give the following definition,

Definition 4.3. *The Revelio protocol is said to provide privacy if for every PPT $\mathcal{D}_{\mathsf{Rev}}$ in the `RevPriv` experiment, there exists a negligible function $\mathsf{negl}(\lambda)$ of the security parameter λ such that,*

$$\left| \Pr[\mathcal{D}_{\mathsf{Rev}}(\mathsf{Rev}_b) = b] - \frac{1}{2} \right| \leq \mathsf{negl}(\lambda). \tag{4.25}$$

Now we give the following theorem.

Theorem 4.1. *The Revelio protocol provides privacy in the random oracle model under the DDH assumption.*

The proof of Theorem 4.1 is given in Appendix C.

4.5 Performance

Revelio is the first proof of reserves protocol for MimbleWimble exchanges which provides privacy. Let n be the size of the anonymity set and s be the number of exchange-owned addresses. A Revelio prover needs to compute $(8n - s)$ group exponentiations and $\mathcal{O}(n)$ point additions and field operations. A Revelio verifier needs to compute $8n$ group exponentiations and $\mathcal{O}(n)$ point additions and field operations. The proof size is $2n$ group elements and $5n$ scalars.

Table 4.1: Proof Generation and Verification Performance for Revelio and Simple

$\mathcal{C}_{\text{anon}}$ Size	$\mathcal{C}_{\text{own}}$ Size	Revelio Proof Size	Revelio Gen. Time	Revelio Ver. Time	Simple Proof Size	Simple Gen. Time	Simple Ver. Time
100	25	0.02 MB	1.13 s	1.14 s	0.92 KB	21.96 ms	22.00 ms
100	50	0.02 MB	1.14 s	1.15 s	1.74 KB	22.12 ms	22.17 ms
100	75	0.02 MB	1.15 s	1.16 s	2.57 KB	22.38 ms	22.40 ms
1000	250	0.22 MB	11.98 s	11.98 s	8.34 KB	23.52 ms	23.56 ms
1000	500	0.22 MB	11.80 s	11.86 s	0.01 MB	25.42 ms	25.42 ms
1000	750	0.22 MB	11.73 s	11.84 s	0.02 MB	27.46 ms	27.46 ms
10000	2500	2.26 MB	109.57 s	110.23 s	0.08 MB	41.36 ms	41.28 ms
10000	5000	2.26 MB	109.87 s	110.78 s	0.16 MB	60.36 ms	60.18 ms
10000	7500	2.26 MB	109.57 s	110.46 s	0.24 MB	79.10 ms	78.73 ms

There is no existing benchmark which can be used to evaluate the relative performance of Revelio. Hence, we compare Revelio to the simple non-private protocol described in equation (4.18). The simulation code was implemented in Rust using the rust-secp256k1-zkp library [52]. It is available at [53].

The performance of the Revelio proof generation and verification algorithms is given in Table 4.1 for anonymity list $\mathcal{C}_{\text{anon}}$ having sizes 100, 1000, and 10000. For each case, the percentage of known addresses is either 25%, 50%, or 75%. The table also shows the performance of the simple non-private protocol as a function of $\mathcal{C}_{\text{own}}$ size (the $\mathcal{C}_{\text{anon}}$ parameter is irrelevant for this protocol). The execution times were measured on a single core of an Intel i7-7700 3.6 GHz CPU. The Revelio protocol is orders of magnitude slower and its proof size an order of magnitude larger compared to the simple non-private protocol. Nevertheless the proof sizes and running times of Revelio are practical and the higher values are justified by the privacy it provides. As the NIZKPoK proof generation and verification for different outputs can proceed in parallel, running times can be reduced by parallel execution.

4.6 Conclusion

We have presented the design of the first privacy-preserving collusion-resistant proof of reserves protocol for MimbleWimble exchanges. Our simulations show that the proof generation and verification times are practical. Here are two potential areas for improvement.

- The collusion-resistance property of Revelio works only if all the exchanges generate their proofs using the same blockchain state. If an exchange generates a Revelio proof and then transfers some coins to another exchange before the latter generates its Revelio proof, then these two exchanges end up effectively sharing some coins. Enforcing simultaneous proof generation is an interesting direction for future research.

- Revelio provides privacy in the sense that a PPT adversary cannot tell where an output in $\mathcal{C}_{\text{anon}}$ belongs to $\mathcal{C}_{\text{own}}$ or $\mathcal{C}_{\text{anon}} \setminus \mathcal{C}_{\text{own}}$. But this level of privacy is far from perfect as the adversary knows that the exchange owns some outputs in $\mathcal{C}_{\text{anon}}$. Since the proof sizes and generation/verification times in Revelio increase linearly with the size of $\mathcal{C}_{\text{anon}}$, simply setting $\mathcal{C}_{\text{anon}}$ to be equal to the whole unspent output set $\mathcal{C}_{\text{unspent}}$ is not a scalable strategy. Development of more efficient proof of reserves protocols which will enable perfect privacy is another interesting direction for future research.

Bagad et al. [54] address the last issue mentioned above. They propose RevelioBP, a Bulletproofs [17] based technique for MimbleWimble proof of reserves. RevelioBP gives proof size logarithmic in terms of the anonymity set. However, the proof generation and verification times are worse compared to Revelio.

Chapter 5

Nummatus: A Proof of Reserves Protocol for Quisquis

Quisquis [10] is a recently proposed design for an account-based privacy focused cryptocurrency. It solves the problem of the always growing UTXO set which plagues other privacy focused cryptocurrencies like Monero [6] and ZCash [7]. In this chapter*, we propose Nummatus, a proof of reserves protocol for Quisquis exchanges†. Our protocol is privacy preserving in the sense that it only reveals that the exchange-owned accounts belong to a larger anonymity set of accounts, without identifying which ones are exchange-owned. The protocol gives a technique to detect collusion between exchanges who use the same account to generate their respective proofs of reserves. We also describe a non-private proof of reserves protocol for Quisquis exchanges called Simplus, with the intention of quantifying the cost of deploying a privacy preserving protocol. We give simulation results to compare the performance of the Nummatus and Simplus protocols. These simulations show that, while privacy has a cost, deploying Nummatus is a practical proposition.

While no reference implementation of Quisquis exists, the design is novel enough to warrant designing proof of reserves protocol for it. So when an implementation does emerge and the Quisquis cryptocurrency becomes available on exchanges, the Nummatus proof of reserves protocol can be employed in proofs of solvency. Below we describe the

*This chapter is based on joint work with Arnab Jana.

†*Quisquis* is Latin for "whoever, whatever" [55]. *Nummatus* is Latin for "moneyed, rich" [56]. We chose this name for our protocol as it enables an exchange to show that it is *rich enough* to meet its liabilities.

Nummatus scheme after giving a brief description of the Quisquis scheme.

5.1 Overview of Quisquis

Privacy focused cryptocurrencies like Monero and Zcash allow users to conceal the source of coins in a transaction using ring signatures [27] or zk-SNARKs [57]. As the true source of coins is not revealed, a one-time address in Monero and a commitment in Zcash cannot be considered spent.[‡] Consequently, these cryptocurrencies have poor scalability in the long term without the opportunity to prune spent outputs from the blockchain.

Quisquis is a cryptocurrency proposal which offers both privacy and scalability [10]. It is an account-based design (as opposed to a UTXO-based design), where each account consists of a public key and a commitment to the balance in the account. The public key is generated from the secret key and a randomizing scalar. Hence there are many possible public keys corresponding to a secret key (unlike Bitcoin where the public key is a deterministic function of the secret key).

Each Quisquis transaction involves some *input accounts* and an *equal* number of *output accounts*. Each output account is an updated version of exactly one of the input accounts, where the update consists of an update of the input account's public key (the account secret key remains unchanged) *and/or* an update of the input account's balance. Unlike Bitcoin where the input UTXOs in a transaction represent the source of funds and output UTXOs represent destinations, the input accounts in a Quisquis transaction consist of both source accounts and destination accounts. Additionally, some *passive accounts* are added to the list of input accounts in the transaction to obfuscate the sources and destinations of funds. Only the public keys of the passive accounts are updated in a transaction and their balances are unchanged. On the other hand, the balances of source accounts are reduced and the balances of destination accounts are increased. For both source and destination accounts, the public keys are updated. The output accounts are presented in a lexicographical order to prevent linking of specific output accounts with input accounts. Once a Quisquis transaction appears on the blockchain, the input

[‡]Some addresses in Monero can be identified as spent by using techniques from Moser *et al.* [31], Kumer *et al.* [32], Yu *et al.* [33], and Hinteregger *et al.* [34]. But these techniques were able to identify only a small fraction of the RingCT outputs as spent.

accounts can be pruned. Quisquis has special transactions for creation and deletion of accounts. Regular transactions do not create new accounts and this is the main reason for the scalability of the design. In the following subsections, we present a more precise description of those aspects of Quisquis that are necessary to present Nummatus.

5.1.1 Quisquis Accounts

Let $\mathbb{G}$ be a group[§] with prime order p and generator g. The Decisional Diffie-Hellman (DDH) problem is assumed to be hard in $\mathbb{G}$. A Quisquis account based on $\mathbb{G}$ is specified by four group elements (a, b, c, d). The first two group elements specify a *public key* $\mathtt{pk} = (a, b) = (g^t, g^{k \cdot t})$ where $t \in \mathbb{F}_p$ is an arbitrary scalar and $k \in \mathbb{F}_p$ is the *secret key*. The last two group elements specify a *commitment* which depends on $\mathtt{pk}$ and is given by $\mathtt{com} = (c, d) = (a^r, g^v b^r)$ where $r \in \mathbb{F}_p$ is an arbitrary scalar. Here $v \in \mathbb{F}_p$ is the value being committed to by $\mathtt{com}$. In summary, a Quisquis account is of the form

$$\mathtt{acct} = (a, b, c, d) = (a, b, a^r, g^v b^r) = \left(g^t, g^{k \cdot t}, g^{t \cdot r}, g^{v + k \cdot t \cdot r}\right) \tag{5.1}$$

where k is the secret key, v is the value in the account, and r, t are arbitrary scalars.

In the Quisquis design, knowledge of the secret key k is sufficient to prove ownership of an account and to create transactions which transfer value out of it, i.e. knowledge of the scalars r and t is not required. This feature allows an entity to perform a *secret key preserving update* of an account, even when the entity has no knowledge of the secret key, value, or scalars used to create the account elements. An update of an account $\mathtt{acct} = (a, b, c, d)$ to an account $\mathtt{acct}' = (a_1, b_1, c_1, d_1)$ preserves the secret key k and changes the amount from v to $v + \delta$ if the following equations hold.

$$\begin{aligned} b &= a^k, \quad d = g^v c^k, \\ b_1 &= a_1^k, \quad d_1 = g^{v+\delta} c_1^k. \end{aligned} \tag{5.2}$$

A Quisquis transaction involves account updates of this kind in addition to range proofs on the values $v + \delta$ to ensure that the amount changes are valid.

The account update procedure is as follows:

[§]In this chapter, we shall follow multiplicative notation for group operations to be consistent with the Quisquis paper [10]. We shall also denote group elements by small letters unlike the previous chapters for the same reason.

1. Suppose `acct` in equation (5.1) is the account to be updated.

2. The updater chooses scalars $t_1, r_1, \delta \in \mathbb{F}_p$. While t_1, r_1 are chosen arbitrarily, δ represents the change in the value stored in the account.

3. The updater computes the updated public key as

$$\mathtt{pk}' = (a_1, b_1) = (a^{t_1}, b^{t_1}). \tag{5.3}$$

4. The updater computes the updated commitment as

$$\mathtt{com}' = (c_1, d_1) = (ca^{r_1}, dg^{\delta} b^{r_1}). \tag{5.4}$$

   Note that this update can be interpreted as the coordinate-wise product of the commitment (c, d) with the commitment $(a^{r_1}, g^{\delta} b^{r_1})$.

5. The updated account is $\mathtt{acct}' = (\mathtt{pk}', \mathtt{com}') = (a_1, b_1, c_1, d_1)$.

It is easy to check that the equations in (5.2) hold. Since $b = a^k$, we have

$$b_1 = b^{t_1} = \left(a^k\right)^{t_1} = \left(a_1^t\right)^k = a_1^k. \tag{5.5}$$

Since $d = g^v c^k$ and $c_1 = ca^{r_1}$, we have

$$d_1 = dg^{\delta} b^{r_1} = g^v c^k g^{\delta} \left(a^k\right)^{r_1} = g^{v+\delta} \left(ca^{r_1}\right)^k = g^{v+\delta} c_1^k. \tag{5.6}$$

To see that the updated account has the structure specified in equation (5.1), consider the following version of $\mathtt{acct}'$.

$$\mathtt{acct}' = (a_1, b_1, c_1, d_1) = \left(a^{t_1}, b^{t_1}, ca^{r_1}, dg^{\delta} b^{r_1}\right) \tag{5.7}$$

$$= \left(g^{t \cdot t_1}, g^{k \cdot t \cdot t_1}, g^{t(r+r_1)}, g^{v+\delta+k \cdot t \cdot (r+r_1)}\right) \tag{5.8}$$

$$= \left(g^{t'}, g^{k \cdot t'}, g^{t' \cdot r'}, g^{v+\delta+k \cdot t' \cdot r'}\right), \tag{5.9}$$

where $t' = t \cdot t_1$ and $r' = t_1^{-1} \cdot (r + r_1)$.

In the subsequent discussion, we will find it convenient to denote the above account update procedure by the notation $\mathtt{UpdateAcct}(\mathtt{acct}, t_1, r_1, \delta)$.

5.1.2 Quisquis Transactions

Suppose Alice owns account $\texttt{acct}_1$ and wants to transfer δ amount to account $\texttt{acct}_2$. Alice will choose $n-2$ additional accounts from the blockchain which will play the role of passive accounts. Let these passive accounts be denoted by $\texttt{acct}_3$, $\texttt{acct}_4$, ..., $\texttt{acct}_n$. Alice will construct a transaction with input accounts given by $\texttt{inputs} = \{\texttt{acct}_1, \texttt{acct}_2, \ldots, \texttt{acct}_n\}$. The input accounts will be listed in a canonical order like lexicographical ordering to conceal the identity of the non-passive accounts. Alice will update each of the input accounts to generate output accounts given by $\texttt{outputs} = \{\texttt{acct}'_1, \texttt{acct}'_2, \ldots, \texttt{acct}'_n\}$ where

$$\begin{aligned}
\texttt{acct}'_1 &= \texttt{UpdateAcct}(\texttt{acct}_1, t_1, r_1, -\delta),\\
\texttt{acct}'_2 &= \texttt{UpdateAcct}(\texttt{acct}_2, t_2, r_2, \delta),\\
\texttt{acct}'_3 &= \texttt{UpdateAcct}(\texttt{acct}_3, t_3, r_3, 0),\\
\vdots \quad &= \qquad \vdots\\
\texttt{acct}'_n &= \texttt{UpdateAcct}(\texttt{acct}_n, t_n, r_n, 0),
\end{aligned}$$

where the t_is and r_is are arbitrarily chosen scalars. Note that the balance in the source account $\texttt{acct}_1$ is reduced by δ and the balance in the destination account $\texttt{acct}_2$ is increased by δ. The balances of the passive accounts remain the same. The output accounts will also be presented in a canonical ordering to conceal the mapping from the inputs to the outputs. Alice then constructs a zero knowledge proof π that convinces the verifier of the following statements.

1. Each account in `outputs` is an update of exactly one of the accounts in `inputs`.
2. The account updates cumulatively satisfy preservation of balances i.e. $\sum \delta_i = 0$, where δ_i is the update of balance of $\texttt{acct}_i$.
3. The balance of the source account does not become negative after the update.
4. The balance of the destination account after the update is in the correct range of values (range proof).
5. The balances of the passive accounts remain unchanged.

The transaction `txn` itself consists of the sets `inputs`, `outputs`, and the zero knowledge proof π, i.e. $\texttt{txn} = (\texttt{inputs}, \texttt{outputs}, \pi)$. While our illustrative example had only

one source account and one destination account, transactions with multiple sources and destinations are allowed in Quisquis.

The implication of this transaction model to our context is that exchange-owned accounts may be updated several times before they are used in the proof of reserves protocol. If the exchange is not involved in all the updates of an account, it will not know the discrete logarithm of the group elements forming the public key and commitment with respect to the generator g. This fact has to be taken into consideration in the proof of reserves protocol design.

5.2 Nummatus Proof of Reserves Protocol

The overall design of the Nummatus protocol is similar to the proof of reserves protocols which have been discussed in previous chapters. However, unlike these previously proposed protocols, the Nummatus protocol requires a sequence of elements $h_1, h_2, h_3, \ldots$ from $\mathbb{G}$ whose discrete logarithms with respect to g and each other are unknown. The sequence $\{h_j \in \mathbb{G} \mid j = 1, 2, \ldots\}$ can be generated by repeatedly hashing g while ensuring the result falls in the group $\mathbb{G}$. All the exchanges need to agree upon the specific sequence generation procedure. A Nummatus proof which is generated after the jth Quisquis block has appeared and before the $(j+1)$th block has appeared on the blockchain will use the jth element h_j. We will see that this sequence will be used to serve three purposes, namely, (1) to compute the commitment to the total reserves amount of the exchange, (2) to reveal collusion between exchanges sharing account to generate proof of reserves, and (3) to conceal the identity of the exchange's accounts across multiple Nummatus proofs.

Suppose an exchange is generating the Nummatus proof of reserves after the jth Quisquis block. We give a high-level description of the procedure followed by the exchange below (a more precise description is given in Section 5.2.1).

1. Let $\mathcal{A}_{\text{all}}$ be the set of all accounts and let $\mathcal{A}_{\text{own}} \subset \mathcal{A}_{\text{all}}$ be the accounts owned by the exchange.[¶] The exchange chooses a set of accounts $\mathcal{A}_{\text{oth}}$ not owned by it, i.e. $\mathcal{A}_{\text{oth}} \subset \mathcal{A}_{\text{all}} \backslash \mathcal{A}_{\text{own}}$. These other accounts are added to the set of exchange-owned accounts to form the *anonymity set* $\mathcal{A}_{\text{anon}} = \mathcal{A}_{\text{own}} \cup \mathcal{A}_{\text{oth}}$.

[¶]Sets $\mathcal{A}_{\text{all}}$ and $\mathcal{A}_{\text{own}}$ may change every time a new block is added to the Quisquis blockchain. Here we consider particular instances of these sets after the jth block is added to the blockchain.

2. Let $\mathcal{A}_{\text{anon}} = [\texttt{acct}_1, \texttt{acct}_2, \ldots, \texttt{acct}_n]$ where

$$\texttt{acct}_i = (a_i, b_i, c_i, d_i) = \left(a_i, a_i^{k_i}, a_i^{r_i}, g^{v_i} a_i^{k_i \cdot r_i}\right). \tag{5.10}$$

Here $k_i \in \mathbb{F}_p$ is the secret key, $v_i \in \mathbb{F}_p$ is the account balance, $r_i \in \mathbb{F}_p$ is an arbitrary scalar, and $a_i = g^{t_i}$ for an arbitrary $t_i \in \mathbb{F}_p$.

For each $\texttt{acct}_i$, the exchange creates a Pedersen commitment p_i and a non-interactive zero knowledge proof of knowledge (NIZKPoK, see Section 4.2 for an overview) σ_i which proves the *disjunction* of the following statements:

(a) Account $\texttt{acct}_i$ is owned by the exchange, i.e. the exchange knows the secret key k_i associated with $\texttt{acct}_i$, and p_i is a Pedersen commitment to the balance v_i in $\texttt{acct}_i$.

(b) Pedersen commitment p_i is a commitment to the zero amount.

Note that the proof σ_i proves that *one of these two statements* is true without revealing which one.

3. The exchange publishes the anonymity set $\mathcal{A}_{\text{anon}}$, Pedersen commitments $[p_1, \ldots, p_n]$, and NIZKPoKs $[\sigma_1, \sigma_2, \ldots, \sigma_n]$. It claims that $p_{\text{res}} = \prod_{i=1}^{n} p_i$ is a Pedersen commitment to the total reserves of the exchange.

To understand the different parts of the protocol, we need to look at the structure of the individual Pedersen commitments p_i. As discussed above, the discrete logarithm of h_j with respect to g is not known. A Pedersen commitment to an amount $v \in \mathbb{F}_p$ with respect to bases g and h_j is given by $g^v h_j^w$ where $w \in \mathbb{F}_p$ is the blinding factor.

When $\texttt{acct}_i \in \mathcal{A}_{\text{own}}$, the Nummatus protocol sets $p_i = g^{v_i} h_j^{k_i}$. So p_i is a commitment to the balance v_i of $\texttt{acct}_i$ and the secret key k_i is the blinding factor in this case. When $\texttt{acct}_i \notin \mathcal{A}_{\text{own}}$, the Nummatus protocol sets $p_i = h_j^{w_i}$ for a randomly chosen $w_i \in \mathbb{F}_p$, making p_i a commitment to the zero amount. Let $\mathcal{I}_{\text{own}}$ be the indices in $\{1, 2, \ldots, n\}$ corresponding to accounts in $\mathcal{A}_{\text{own}}$, i.e. $\texttt{acct}_i \in \mathcal{A}_{\text{own}}$ for all $i \in \mathcal{I}_{\text{own}}$. Let $\mathcal{I}_{\text{own}}^c$ denote those indices in $\{1, 2, \ldots, n\}$ which are not in $\mathcal{I}_{\text{own}}$. Then we have

$$p_{\text{res}} = \prod_{i=1}^{n} p_i = \prod_{i \in \mathcal{I}_{\text{own}}} g^{v_i} h_j^{k_i} \prod_{i \in \mathcal{I}_{\text{own}}^c} h_j^{w_i} = g^{v_{\text{res}}} h_j^{w_{\text{res}}}, \tag{5.11}$$

where

$$v_{\text{res}} = \sum_{i \in \mathcal{G}_{\text{own}}} v_i, \tag{5.12}$$

$$w_{\text{res}} = \sum_{i \in \mathcal{G}_{\text{own}}} k_i + \sum_{i \in \mathcal{G}^c_{\text{own}}} w_i. \tag{5.13}$$

Thus p_{res} is a Pedersen commitment to the exchange's total reserves v_{res}. If $p_{\text{liab}} = g^{v_{\text{liab}}} h_j^{w_{\text{liab}}}$ is a Pedersen commitment to the total liabilities of the exchange, then a proof of solvency reduces to showing that

$$p_{\text{res}} p_{\text{liab}}^{-1} = g^{v_{\text{res}} - v_{\text{liab}}} h_j^{w_{\text{res}} - w_{\text{liab}}} \tag{5.14}$$

is a commitment to a non-negative amount in the correct range.

If two exchanges share an account $\texttt{acct}_i$ while generating their respective proofs of reserves after the jth Quisquis block, then the account will appear in both the anonymity sets. But this is not enough to prove account sharing collusion between the exchanges. However, the commitment $p_i = g^{v_i} h_j^{k_i}$ corresponding to a shared account will appear in both lists of commitments, revealing the collusion.

The reason for choosing a different base h_j for generating the Pedersen commitments after each block is to prevent leaking the identity of exchange-owned accounts across multiple Nummatus proofs. Suppose the same base h is used to generate commitments in all Nummatus proofs given by an exchange. Then the commitments of exchange-owned accounts will remain same across proofs assuming the balances of the accounts (the v_is) remain same. However, the commitments of unknown accounts will be different in different proofs because of different w_is. Thus an observer will be able identify which accounts belong to the exchange.

A consequence of this design is that exchanges cannot use the same secret key for multiple accounts if they want to use the Nummatus proof of reserves protocol. This is not a serious restriction as the convenience afforded by having the same secret key for multiple accounts is negligible compared to the security provided by having different keys for different accounts. This issue is further discussed in Section 5.3.3.

Note that the proof σ_i proves that p_i is a Pedersen commitment of the form which is either $g^{v_i} h_j^{k_i}$ or $h_j^{w_i}$. If an exchange does not own the account $\texttt{acct}_i$, it will be forced to set p_i to the form $h_j^{w_i}$. When exchange does own the account, it can set p_i to be of the form $g^{v_i} h_j^{k_i}$. In the latter scenario, there is nothing stopping the exchange from setting

p_i to be of the form $h_j^{w_i}$. But this will mean that the balance v_i of account $\texttt{acct}_i$ is not counted in the total reserves amount v_{res}. In other words, the exchange is under-reporting the reserves it owns. This is not a problem as long as the reserves exceed the liabilities, since proving solvency is the final goal.

Due to the DDH assumption in the underlying group $\mathbb{G}$, the Nummatus proof of reserves protocol satisfies the following properties:

- *Inflation resistance*: No probabilistic polynomial time (PPT) exchange will be able to generate a commitment to an amount which exceeds the reserves it actually owns.
- *Collusion resistance*: If two exchanges share an account while generating their respective proofs of reserves (from the same blockchain state), then such collusion can be detected.
- *Privacy*: No PPT adversary will be able to distinguish whether an account in the anonymity set belongs to the exchange or not, estimate account balances or the total reserves amount.

5.2.1 Proof Generation

Suppose a Quisquis exchange wants to generate a Nummatus proof corresponding to its reserves after the jth Quisquis block. It performs the following procedure:

1. The exchange chooses the anonymity set of accounts $\mathcal{A}_{\text{anon}}$ from the set of all accounts $\mathcal{A}_{\text{all}}$ present on the blockchain after the jth Quisquis block has appeared and before the $(j+1)$th block appeared. The exchange owns a subset $\mathcal{A}_{\text{own}}$ of $\mathcal{A}_{\text{anon}} = [\texttt{acct}_1, \texttt{acct}_2, \ldots, \texttt{acct}_n]$.
2. For each $\texttt{acct}_i \in \mathcal{A}_{\text{anon}}$ such that $\texttt{acct}_i = \left(a_i, a_i^{k_i}, c_i, g^{v_i} c_i^{k_i}\right)$, the exchange generates a Pedersen commitment p_i of the form

$$p_i = \begin{cases} g^{v_i} h_j^{k_i} & \text{if } \texttt{acct}_i \in \mathcal{A}_{\text{own}}, \\ h_j^{w_i} & \text{if } \texttt{acct}_i \notin \mathcal{A}_{\text{own}}, \end{cases} \tag{5.15}$$

   where the w_is are chosen independently and uniformly from $\mathbb{F}_p$.

3. For each $\texttt{acct}_i \in \mathcal{A}_{\text{anon}}$ given by $\texttt{acct}_i = (a_i, b_i, c_i, d_i) = \left(a_i, a_i^{k_i}, c_i, g^{v_i} c_i^{k_i}\right)$, the exchange generates a NIZKPoK $\sigma_i = (e_{i,1}, e_{i,2}, s_{i,1}, s_{i,2}) \in \mathbb{F}_p^4$ of the form

$$\text{PoK}\left\{(\alpha, \beta) \;\middle|\; \left(b_i = a_i^{\alpha} \;\wedge\; p_i d_i^{-1} = \left(c_i^{-1} h_j\right)^{\alpha}\right) \vee \left(p_i = h_j^{\beta}\right)\right\}. \tag{5.16}$$

   The NIZKPoK σ_i proves that the exchange knows scalars α, β such that *either* $p_i = h_j^{\beta}$ *or* $b_i = a_i^{\alpha}$ and $p_i d_i^{-1} = \left(c_i^{-1} h_j\right)^{\alpha}$. The algorithm for generating σ_i is given in Appendix D.2.

4. The exchange publishes the base h_j, the anonymity set $\mathcal{A}_{\text{anon}}$, Pedersen commitments $[p_1, p_2, \ldots, p_n]$, and NIZKPoKs $[\sigma_1, \sigma_2, \ldots, \sigma_n]$. It claims that $p_{\text{res}} = \prod_{i=1}^{n} p_i$ is a Pedersen commitment to the total reserves of the exchange.

Equation (5.15) reflects the requirement that p_i is a commitment to the account balance for exchange-owned accounts and a commitment to the zero amount for other accounts. The choice of blinding factor in each case makes p_i a deterministic function of the secret key and the balance for exchange-owned accounts and a random group element for other accounts. The NIZKPoK condition in equation (5.16) in fact ensures that an exchange does not deviate from the constructions of p_i given in equation (5.15). It states that either p_i is a commitment to the zero amount or the following conditions (in italics) hold:

1. *The discrete logarithm of b_i with respect a_i is known to the party generating the proof σ_i.*

   As $b_i = a_i^{k_i}$, this condition implies that the secret key k_i is known to the exchange.

2. *The party generating the proof σ_i knows the discrete logarithm of $p_i d_i^{-1}$ with respect to $c_i^{-1} h_j$. Furthermore, the discrete logarithm is equal to the discrete logarithm of b_i with respect to a_i.*

   As $b_i = a_i^{k_i}$, from (5.16) we have

$$p_i d_i^{-1} = \left(c_i^{-1} h_j\right)^{k_i}. \tag{5.17}$$

   Since $d_i = g^{v_i} c_i^{k_i}$, from the above equation we get

$$p_i g^{-v_i} c_i^{-k_i} = c_i^{-k_i} h_j^{k_i} \implies p_i = g^{v_i} h_j^{k_i}. \tag{5.18}$$

   Thus p_i is a commitment to the balance v_i in the account $\texttt{acct}_i$ with blinding factor k_i as given in equation (5.15).

5.2.2 Proof Verification

Given a Nummatus proof of reserves from an exchange referring to the blockchain state after the jth Quisquis block, the verifier checks the following conditions:

1. All the accounts in the anonymity set $\mathcal{A}_{\text{anon}}$ must appear on the blockchain immediately after the jth block. If an account in $\mathcal{A}_{\text{anon}}$ does not appear on the blockchain, the proof is considered invalid.

2. For each i, the NIZKPoK σ_i must pass the verification procedure given in Appendix D.2.

3. The commitments $p_i, i = 1, 2, \ldots, n$, must not appear in another exchange's Nummatus proof. If the same commitments p_i appears in the Nummatus proofs of two different exchanges, then collusion is declared and the proof of reserves is considered invalid.

5.3 Nummatus Security Properties

Nummatus is equipped with three security properties namely inflation resistance, collusion resistance, and privacy. In this section we discuss each of them.

5.3.1 Inflation Resistance

Inflation resistance refers to the requirement that a PPT exchange can use the Nummatus protocol to generate a Pedersen commitment p_{res} to an amount v_{res} which is greater than the total reserves it owns except with a negligible probability. Suppose p_{res} is a commitment to v_{res} which is greater than $\sum_{i \in \mathcal{I}_{\text{own}}} v_i$. Then since

$$p_{\text{res}} = \prod_{i=1}^{n} p_i = \prod_{i \in \mathcal{I}_{\text{own}}} p_i \prod_{i \in \mathcal{I}^c_{\text{own}}} p_i, \tag{5.19}$$

it must be that either p_i is not a commitment to zero for some $i \in \mathcal{I}^c_{\text{own}}$ or p_i is a commitment to an amount larger than the account balance v_i for some $i \in \mathcal{I}_{\text{own}}$. For $i \in \mathcal{I}^c_{\text{own}}$, the exchange does not know the secret key k_i and consequently it must generate the NIZKPoK σ_i by setting p_i to be commitment to the zero amount. For $i \in \mathcal{I}_{\text{own}}$, if a PPT exchange sets p_i to be a commitment to a nonzero amount then σ_i forces this amount

to be the account balance v_i (see equation (5.18)). So the inflation resistance property of Nummatus follows from the unforgeability of the NIZKPoKs σ_i.

5.3.2 Collusion Resistance

The collusion resistance property ensures that two PPT exchanges can share the balance of a common account while generating Nummatus proofs except with a negligible probability of being undetected. When p_i is not a commitment to zero, the account $\texttt{acct}_i$ must be owned by the exchange as the private key k_i is needed to create p_i as $g^{v_i}h_j^{k_i}$. This form of p_i is a deterministic function of v_i and k_i. So two exchanges sharing $\texttt{acct}_i$ after the jth Quisquis block will produce same p_i in their proofs. If this happens, then account sharing collusion is immediately detected.

5.3.3 Privacy

In this section, we prove that the Nummatus protocol provides privacy to the exchange, even when multiple proofs are observed by an adversary. First, we make the following assumption.

The secret keys of all exchange-owned accounts in the anonymity set $\mathcal{A}_{anon}$ are all distinct. This is necessary as the Nummatus protocol cannot provide privacy without this constraint. To see why, suppose two accounts in $\mathcal{A}_{\text{anon}}$ share the same secret key k. Let their corresponding Pedersen commitments be $p_i = g^v h_j^k$ and $p_{i'} = g^{v'} h_j^k$ where i, i' are the account indices and v, v' are the account balances. An adversary can figure out that these two accounts are exchange-owned accounts by checking if the equality $p_i g^{-v_1} = p_{i'} g^{-v_2}$ holds for some $(v_1, v_2) \in V^2$ where V is the set of all possible amounts. As the size of V is small, this attack is practical. In fact, the receiver of a funds in a regular Quisquis transaction has to search through all possible values in V to figure out the amount received [10, Section 5.2.3].

We consider the scenario when a Quisquis exchange has generated $f(\lambda)$ Nummatus proofs following the above requirement, where $f(\lambda)$ denotes a polynomial of the security parameter λ. Let these $f(\lambda)$ proofs be denoted by $\mathsf{Num}_{\mathsf{act}} = \left(h_{j_i}, \mathcal{A}_{\text{anon}}^{(i)}, \mathbf{P}^{(i)}, \mathbf{\Sigma}^{(i)}\right)_{i=1}^{f(\lambda)}$,

where we have the following notation.

$$\mathcal{A}^{(i)}_{\text{anon}} = ((a_{i,k}, b_{i,k}, c_{i,k}, d_{i,k}))_{k=1}^{n_i}\,,$$
$$\mathbf{P}^{(i)} = (p_{i,1}, p_{i,2}, \ldots, p_{i,n_i})\,,$$
$$\mathbf{\Sigma}^{(i)} = (\sigma_{i,1}, \sigma_{i,2}, \ldots, \sigma_{i,n_i})\,,$$

where n_i is the size of the ith anonymity set $\mathcal{A}^{(i)}_{\text{anon}}$ and $p_{i,k}, \sigma_{i,k}$ are the Pedersen commitment and the NIZKPoK signature corresponding to the kth account in the ith anonymity set respectively.

Similar to previous chapters, we consider a simulator $\mathcal{S}_{\mathsf{Num}}$ which do not have any information of the exchange except having access to $\mathsf{Num}_{\mathsf{act}}$. To generate the simulated proofs, $\mathcal{S}_{\mathsf{Num}}$ consider the ith Pedersen commitment vector $\mathbf{P}^{(i)} = (p_{i,1}, p_{i,2}, \ldots, p_{i,n_i})$. She constructs the ith simulated Pedersen commitment vector $\hat{\mathbf{P}}^{(i)} = (\hat{p}_{i,1}, \hat{p}_{i,2}, \ldots, \hat{p}_{i,n_i})$ in the following manner. She chooses $q_{i,k} \xleftarrow{\$} \mathbb{F}_p$ and sets $\hat{p}_{i,k} = h_{j_i}^{q_{i,k}}$, for all $k \in [n_i]$. After this, she constructs the simulated signature $\hat{\sigma}_{i,k}$ using the knowledge of $q_{i,k}$. After doing this for all $i \in [f(\lambda)], k \in [n_i]$, she constructs $(\hat{\mathbf{\Sigma}}^{(i)})_{i=1}^{f(\lambda)}$ where $\hat{\mathbf{\Sigma}}^{(i)} = (\hat{\sigma}_{i,1}, \hat{\sigma}_{i,2} \ldots, \hat{\sigma}_{i,n_i})$. The simulated proof is $\mathsf{Num}_{\mathsf{sim}} = \left(h_{j_i}, \mathcal{A}^{(i)}_{\text{anon}}, \hat{\mathbf{P}}^{(i)}, \hat{\mathbf{\Sigma}}^{(i)}\right)_{i=1}^{f(\lambda)}$.

If no PPT distinguisher $\mathcal{D}_{\mathsf{Num}}$ can distinguish between $\mathsf{Num}_{\mathsf{act}}$ and $\mathsf{Num}_{\mathsf{sim}}$ except with a probability negligibly better than that of random guessing, then we can say $\mathsf{Num}_{\mathsf{act}}$ does not reveal any information about the exchange. In particular, we define the privacy experiment `NumPriv` for the Nummatus protocol as follows.

1. $\mathcal{S}_{\mathsf{Num}}$ sets $\mathsf{Num}_0 = \mathsf{Num}_{\mathsf{sim}}$ and $\mathsf{Num}_1 = \mathsf{Num}_{\mathsf{act}}$.
2. $\mathcal{S}_{\mathsf{Num}}$ chooses a bit $b \xleftarrow{\$} \{0, 1\}$ randomly.
3. $\mathcal{S}_{\mathsf{Num}}$ sends Num_b to $\mathcal{D}_{\mathsf{Num}}$.
4. $\mathcal{D}_{\mathsf{Num}}$ outputs a bit $\mathcal{D}_{\mathsf{Num}}(\mathsf{Num}_b)$ as a prediction of b.

Now we give the following definition.

Definition 5.1. *The Nummatus protocol is said to provide privacy if for every PPT $\mathcal{D}_{\mathsf{Num}}$ in the `NumPriv` experiment, there exists a negligible function $\mathsf{negl}(\lambda)$ of the security parameter λ such that,*

$$\left| \Pr[\mathcal{D}_{\mathsf{Num}}(\mathsf{Num}_b) = b] - \frac{1}{2} \right| \leq \mathsf{negl}(\lambda). \tag{5.20}$$

$\mathcal{A}_{\text{anon}}$ size	$\mathcal{A}_{\text{own}}$ size	Nummatus Proof Size	Nummatus Generat. Time	Nummatus Verification Time	Simplus Proof Size	Simplus Generat. Time	Simplus Verification Time
100	25	0.02 MB	1.15 s	1.15 s	0.006 MB	0.29 s	0.28 s
100	50	0.02 MB	1.16 s	1.16 s	0.011 MB	0.58 s	0.57 s
100	75	0.02 MB	1.19 s	1.19 s	0.017 MB	0.91 s	0.91 s
1000	250	0.29 MB	11.94 s	11.76 s	0.057 MB	3.00 s	2.98 s
1000	500	0.29 MB	11.92 s	11.77 s	0.114 MB	5.97 s	5.95 s
1000	750	0.29 MB	11.83 s	11.74 s	0.171 MB	8.92 s	8.74 s
10000	2500	2.93 MB	112.65 s	113.36 s	0.572 MB	28.99 s	28.06 s
10000	5000	2.93 MB	112.08 s	113.23 s	1.145 MB	56.40 s	56.63 s
10000	7500	2.93 MB	111.71 s	112.87 s	1.717 MB	85.07 s	85.72 s

Table 5.1: Proof Generation and Verification Performance of Nummatus and Simplus

We have the following theorem which is proved in Appendix D.1.

Theorem 5.1. *The Nummatus protocol provides privacy in the random oracle model under the DDH assumption.*

5.4 Performance

Nummatus is the first proof of reserves protocol for Quisquis exchanges which keeps the identities of the exchange accounts private. For benchmarking purposes, we compare Nummatus to a simple non-private protocol which we will call *Simplus*.[‖] In the Simplus protocol, the exchange reveals the accounts it owns, i.e. the set $\mathcal{A}_{\text{own}}$ is revealed. Like Nummatus, this protocol outputs a Pedersen commitment to the total reserves of the exchange. While the Simplus protocol does not provide privacy, it provides reserve amount privacy. The proof generation in Simplus proceeds as follows:

1. The exchange chooses a set $\mathcal{A}_{\text{own}} = \{\mathtt{acct}_1, \mathtt{acct}_2, \ldots, \mathtt{acct}_m\}$ of its own accounts which are sufficient to meet its liabilities. These accounts need to be present on the

[‖]Simplus is Latin for "simple" [58].

blockchain after the jth Quisquis block has appeared and before the $(j+1)$th block appeared.

2. For each $\texttt{acct}_i \in \mathcal{A}_{\text{own}}$ given by $\texttt{acct}_i = (a_i, b_i, c_i, d_i) = \left(a_i, a_i^{k_i}, c_i, g^{v_i} c_i^{k_i}\right)$, the exchange generates a Pedersen commitment $p_i := g^{v_i} h^{k_i}$ and a NIZKPoK $\psi_i = (e_{i,1}, s_{i,1}) \in \mathbb{F}_p^2$ of the form

$$\text{PoK}\left\{\alpha \;\middle|\; \left(b_i = a_i^{\alpha} \;\wedge\; p_i d_i^{-1} = \left(c_i^{-1} h\right)^{\alpha}\right)\right\}. \tag{5.21}$$

Note that in case of Nummatus, we need to use h_j as base of p_i to make exchange's accounts indistinguishable from accounts not owned by the exchange across multiple Nummatus proofs. But in case of Simplus, the exchange has already revealed the accounts owned by it. Therefore we can simply use h instead of h_j as a base of p_i. The algorithm for generating ψ_i is given in Appendix D.3.

3. The exchange publishes the set $\mathcal{A}_{\text{own}}$, Pedersen commitments $[p_1, p_2, \ldots, p_m]$, and NIZKPoKs $[\psi_1, \psi_2, \ldots, \psi_m]$. It claims that $p_{\text{res}} = \prod_{i=1}^{m} p_i$ is a Pedersen commitment to the total reserves of the exchange.

The NIZKPoK in equation (5.21) ensures that the exchange knows the private key k_i for each account $\texttt{acct}_i$. Furthermore, by the argument presented in the Nummatus protocol discussion, the NIZKPoK ensures that p_i is a commitment to the account balance v_i of $\texttt{acct}_i$. As the exchange's accounts are revealed in the Simplus protocol, collusion between exchanges can be detected if the same account appears in the own account lists of two different exchanges.

The Simplus proof verification proceeds as follows:

1. All the accounts in the set $\mathcal{A}_{\text{own}}$ must appear on the blockchain immediately after the jth block. If not, the proof is considered invalid.

2. For each i, the NIZKPoK ψ_i must pass the verification procedure given in Appendix D.3.

A Nummatus prover needs to compute $6n$ group exponentiations and $\mathcal{O}(n)$ point additions and field operations. A Nummatus verifier needs to compute $6n$ group exponentiations and $\mathcal{O}(n)$ point additions and field operations. The proof size is $(5n + 2)$ group elements and $4n$ scalars. Whereas a Simplus prover needs to compute $4n$ group

exponentiations and $\mathcal{O}(n)$ point additions and field operations. A simplus verifier needs to compute $4n$ group exponentiations and $\mathcal{O}(n)$ point additions and field operations. The proof size is $(5n + 1)$ group elements and $2n$ scalars.

The simulation code was implemented in Rust using the rust-secp256k1-zkp library [59] which has also been used for Numelio [60]. The performance of the Nummatus proof generation and verification algorithms is given in Table A.2 for anonymity list $\mathcal{A}_{\text{anon}}$ having sizes 100, 1000, and 10000. For each case, the percentage of accounts belonging to the exchange is either 25%, 50%, or 75%. Table A.2 also shows the performance of the Simplus protocol as a function of $\mathcal{A}_{\text{own}}$ size (the $\mathcal{A}_{\text{anon}}$ parameter is irrelevant here). The execution times were measured on single core of an Intel i7-7700 3.6 GHz CPU. The Nummatus protocol is at most 3 to 4 times slower and its proof size is at most 4 to 5 times larger compared to the Simplus protocol. The proof size and execution time of Nummatus protocol are practical and can be reduced further by parallel signature generations and verifications for different accounts in $\mathcal{A}_{\text{anon}}$. The higher values of performance parameters for Nummatus than that of Simplus can be considered as the price we are paying for privacy.

5.5 Conclusion

In this chapter, we described Nummatus, the first privacy preserving proof of reserves protocol for Quisquis [10] exchanges. Using Nummatus, a Quisquis exchange can prove that it holds more reserves than what it owes to its customers without revealing the reserves amount or the identity of owned accounts. Nummatus also detects the account sharing collusion between exchanges provided all exchanges generate their proofs after a particular block is added to the Quisquis blockchain. We give the performance comparison of Nummatus and a non-private proof of reserves protocol which we call Simplus. Our simulation shows that deployment of Nummatus is practical and feasible.

Chapter 6

Conclusion

In this chapter, we summarize the results of the thesis and describe some open problems.

6.1 Summary of Results

In this thesis, we have described proof of reserves protocols for three privacy focused cryptocurrency schemes, namely, Monero, MimbleWimble, and Quisquis. All the protocols are based on standard cryptographic assumptions like the hardness of solving the discrete logarithm and decisional Diffie-Hellman problems. The security proofs use the random oracle model. No trusted setup is required for any of the proposed protocols. All of them generate a Pedersen commitment to the total reserves amount of the exchange while concealing its owned addresses/outputs/accounts from the Monero/MimbleWimble/Quisquis blockchains in a larger anonymity set.

While the proposed proof of reserves protocols for Monero are collusion resistant and inflation resitant, they are unable to completely hide the identity of the exchange-owned addresses due to the explicit revelation of their key images. Furthermore, they are in general unable to preserve the untraceability and amount confidentiality properties of Monero. They do preserve the unlinkability property of Monero. The MProve+ protocol alleviates the drawback in the MProve/MProvisions protocols to some extent, but requires the exchange to carefully choose the transaction rings while spending from source addresses.

In addition to being collusion and inflation resistant, the proposed proof of reserves protocols for MimbleWimble and Quisquis are able to hide the identity of the exchange-

owned outputs/accounts in the anonymity set. Furthermore, they do not affect the privacy properties of the respective cryptocurrency schemes.

Below, we give a brief summary of each chapter.

- In Chapter 2, we described the MProvisions and MProve proof of reserves protocols for a Monero exchange. Both the protocols obfuscate the exchange-owned one-time addresses in a larger anonymity set. To prove that the exchange-owned addresses are unspent, the corresponding key images are published. For cover addresses in the anonymity sets, some dummy key images are also published. As a result, each one-time address in the anonymity set is associated with a real or a dummy key image. The published key images can be used to detect address sharing collusion between the exchanges. However, the key image associated with a source address appears again when the exchange spends from the source address in a future Monero transaction. The one-to-one association of the key images with the source addresses in the MProve/MProvisions proof makes a source spending transaction a zero-mixin transaction and affects the privacy of the exchange as well as the entire Monero network.

- To solve the drawback mentioned above, we proposed the MProve+ protocol in Chapter 3. MProve+ uses techniques of Bulletproofs [17] and Omniring [22] and publishes the key images of only the source addresses in the anonymity set. It appears to a PPT adversary that a published key image can be originated from any one-time address in the anonymity set. Thus the one-to-one association of the key images and the addresses in the MProve/MProvisions proofs becomes a many-to-many association in the MProve+ proofs. This prevents a source spending transaction to become a zero-mixin transaction provided the exchange chooses the ring of the transaction carefully. However, the fact that the exchange is spending in the source spending transaction is revealed for all three protocols. This drawback prevails because of revealing the key images of the source addresses to prove that they are not spent. Solving this drawback and proposing a fully privacy preserving proof of reserves protocol for Monero remains an open problem.

- In Chapters 4 and 5, we proposed the Revelio and Nummatus proof of reserves protocols for MimbleWimble and Quisquis cryptocurrency exchanges. In MimbleWimble,

there are no addresses. Coins are stored in Pedersen commitments called outputs. Quisquis is an account based model which solves the problem of monotonic growth of the UTXO set in the privacy focused cryptocurrencies like Monero. In both Revelio and Nummatus, we have used NIZKPoK signatures formalized by Camenisch and Stadler [46, 47]. The protocols preserve the privacy of the exchange. The simulation results show that they can be deployed in practice.

In the next section, we discuss the future scope of work.

6.2 Open Problems

The following is a list of open problems related to proof of reserves protocols.

- Even though Provisions was proposed for Bitcoin more than five years ago, there are still no proof of reserves protocols for Bitcoin which can handle multisig addresses or pay-to-script-hash (P2SH) addresses. Ideally, a protocol for such address types should involve no trusted setup or, at the very least, a universal trusted setup which allows participation from anyone in the world.

- As mentioned in the previous section, an interesting direction of future work is to propose a proof of reserves protocol for Monero which can (a) hide that the exchange is spending in a source spending transaction and (b) preserve the untraceability and amount confidentiality properties of Monero in general.

- The proof generation and verification times of the protocols proposed in this thesis are linear in the size of the anonymity set. While this is a bottleneck for exchanges who want to achieve better privacy by choosing large anonymity sets, it is not prohibitively expensive. However, linear verification times are inconvenient for customers, who would prefer constant verification times. Using zk-SNARKs and universal trusted setups to obtain proof of reserves protocols with improved verification times is another area for improvement.

List of Publications

[1] A. Dutta, S. Bagad, and S. Vijayakumaran, "MProve+: Privacy enhancing proof of reserves protocol for Monero," *IEEE Transactions on Information Forensics and Security*, 2021.

[2] A. Dutta and S. Vijayakumaran, "MProve: A proof of reserves protocol for Monero exchanges," in *2019 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, 2019, pp. 330–339.

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

[4] A. Dutta, A. Jana, and S. Vijayakumaran, "Nummatus: A privacy preserving proof of reserves protocol for Quisquis." in *In: Hao F., Ruj S., Sen Gupta S. (eds) Progress in Cryptology – INDOCRYPT 2019. INDOCRYPT 2019.Lecture Notes in Computer Science, vol 11898. Springer, Cham*, 2019, pp. 195–215.

Bibliography

[1] 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 (ACM CCS)*, New York, NY, USA, 2015, pp. 720–731.

[2] S. Nakamoto, "Bitcoin: A peer-to-peer electionic cash system (2008)," [Accessed 26-Sep-2019]. [Online]. Available: https://bitcoin.org/bitcoin.pdf

[3] S. Meiklejohn, M. Pomarole, G. Jordan, K. Levchenko, D. McCoy, G. M. Voelker, and S. Savage, "A fistful of bitcoins: Characterizing payments among men with no names," in *Proceedings of the 2013 Conference on Internet Measurement Conference*, ser. IMC '13. New York, NY, USA: Association for Computing Machinery, 2013, p. 127–140. [Online]. Available: https://doi.org/10.1145/2504730.2504747

[4] D. Ron and A. Shamir, "Quantitative analysis of the full bitcoin transaction graph," in *Financial Cryptography and Data Security*, A.-R. Sadeghi, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 6–24.

[5] F. Reid and M. Harrigan, *An Analysis of Anonymity in the Bitcoin System*. New York, NY: Springer New York, 2013, pp. 197–223. [Online]. Available: https://doi.org/10.1007/978-1-4614-4139-7_10

[6] Monero website. [Online]. Available: https://getmonero.org/

[7] Zcash website. [Online]. Available: https://z.cash/

[8] T. E. Jedusor, "Mimblewimble," 2016. [Online]. Available: https://download.wpsoftware.net/bitcoin/wizardry/mimblewimble.txt

[9] A. Poelstra, "Mimblewimble," 2016. [Online]. Available: https://download.wpsoftware.net/bitcoin/wizardry/mimblewimble.pdf

[10] P. Fauzi, S. Meiklejohn, R. Mercer, and C. Orlandi, "Quisquis: A new design for anonymous cryptocurrencies," Cryptology ePrint Archive, Report 2018/990, 2018, https://eprint.iacr.org/2018/990.

[11] IDEX blog. A complete list of cryptocurrency exchange hacks. [Accessed 27-MAY-2021]. [Online]. Available: https://blog.idex.io/all-posts/a-complete-list-of-cryptocurrency-exchange-hacks-updated

[12] "Cryptocurrency crime and anti-money laundering report," 2021 Report, CipherTrace Inc, Feb. 2021. [Online]. Available: https://ciphertrace.com/wp-content/uploads/2021/01/CipherTrace-Cryptocurrency-Crime-and-Anti-Money-Laundering-Report-012821.pdf

[13] C. Decker, J. Guthrie, J. Seidel, and R. Wattenhofer, "Making Bitcoin exchanges transparent," in *20th European Symposium on Research in Computer Security (ESORICS)*, 2015, pp. 561–576.

[14] Wikipedia contributors. Mt. Gox — Wikipedia, the free encyclopedia. [Accessed 27-MAY-2020]. [Online]. Available: https://en.bitcoin.it/wiki/Mt._Gox

[15] Proof-of-Reserves tool for Bitcoin. [Online]. Available: https://github.com/ElementsProject/reserves

[16] T. P. Pedersen, "Non-interactive and information-theoretic secure verifiable secret sharing," in *Advances in Cryptology — CRYPTO '91*. Springer, 1992, pp. 129–140.

[17] B. Bünz, J. Bootle, D. Boneh, A. Poelstra, P. Wuille, and G. Maxwell, "Bulletproofs: Short proofs for confidential transactions and more," in *2018 IEEE Symposium on Security and Privacy (IEEE S&P)*, May 2018, pp. 315–334.

[18] Z. Wilcox, "Proving your Bitcoin reserves," Bitcoin Talk Forum Post, May 2014. [Online]. Available: https://bitcointalk.org/index.php?topic=595180.0

[19] K. Chalkias, K. Lewi, P. Mohassel, and V. Nikolaenko, "Distributed auditing proofs of liabilities," Cryptology ePrint Archive, Report 2020/468, 2020, https://eprint.iacr.org/2020/468.

[20] Grin project website. [Online]. Available: https://grin.mw/

[21] "Beam project website." [Online]. Available: https://www.beam.mw/

[22] R. W. F. Lai, V. Ronge, T. Ruffing, D. Schröder, S. A. K. Thyagarajan, and J. Wang, "Omniring: Scaling up private payments without trusted setup - formal foundations and constructions of ring confidential transactions with log-size proofs," Cryptology ePrint Archive, Report 2019/580, 2019, https://eprint.iacr.org/2019/580.

[23] J. Camenisch, "Group signature schemes and payment systems based on the discrete logarithm problem," Ph.D. dissertation, ETH Zürich, 1998.

[24] J. Camenisch and M. Stadler, "Proof systems for general statements about discrete logarithms," Tech. Rep., 1997.

[25] Koe, K. M. Alonso, and S. Noether, "Zero to Monero: Second Edition," The Monero Project Library, April 2020. [Online]. Available: https://web.getmonero.org/library/Zero-to-Monero-2-0-0.pdf

[26] S. Noether. (2018) Reserve proof pull request. [Online]. Available: https://github.com/monero-project/monero/pull/3027

[27] N. v. Saberhagen, "CryptoNote v 2.0," White paper, 2013. [Online]. Available: https://cryptonote.org/whitepaper.pdf

[28] S. Noether and A. Mackenzie, "Ring confidential transactions," *Ledger*, vol. 1, pp. 1–18, 2016.

[29] J. K. Liu, V. K. Wei, and D. S. Wong, "Linkable spontaneous anonymous group signature for ad hoc groups," in *Australasian Conference on Information Security and Privacy*. Springer, 2004, pp. 325–335.

[30] J. Yu, M. H. A. Au, and P. Esteves-Verissimo, "Re-thinking untraceability in the CryptoNote-style blockchain," Cryptology ePrint Archive, Report 2019/186, 2019, https://eprint.iacr.org/2019/186.

[31] M. Möser, K. Soska, E. Heilman, K. Lee, H. Heffan, S. Srivastava, K. Hogan, J. Hennessey, A. Miller, A. Narayanan, and N. Christin, "An empirical analysis of traceability in the Monero blockchain," *Proceedings on Privacy Enhancing Technologies*, vol. 2018, no. 3, pp. 143–163, 2018.

[32] A. Kumar, C. Fischer, S. Tople, and P. Saxena, "A traceability analysis of Monero's blockchain," in *European Symposium on Research in Computer Security – ESORICS 2017*, 2017, pp. 153–173.

[33] Z. Yu, M. H. Au, J. Yu, R. Yang, Q. Xu, and W. F. Lau, "New empirical traceability analysis of CryptoNote-style blockchains," in *Financial Cryptography and Data Security*, I. Goldberg and T. Moore, Eds. Cham: Springer International Publishing, 2019, pp. 133–149.

[34] A. Hinteregger and B. Haslhofer, "An empirical analysis of Monero cross-chain traceability," *CoRR*, vol. abs/1812.02808, 2018. [Online]. Available: http://arxiv.org/abs/1812.02808

[35] A. Fiat and A. Shamir, "Witness indistinguishable and witness hiding protocols," ACM STOC, 1990.

[36] M. Abe, M. Ohkubo, and K. Suzuki, "1-out-of-n signatures from a variety of keys," in *Advances in Cryptology — ASIACRYPT 2002*. Springer, 2002, pp. 415–432.

[37] J. Katz and Y. Lindell, "Introduction to Modern Cryptography," CRC, 2nd edition, 2015.

[38] S. Bagad and S. Vijayakumaran, "On the confidentiality of amounts in Grin," in *2020 Crypto Valley Conference on Blockchain Technology (CVCBT)*, 2020, pp. 78–82.

[39] MProve simulation code. [Online]. Available: https://github.com/avras/monero/tree/v0.14.0.2-mprove/tests/mprove

[40] MProvisions simulation code. [Online]. Available: https://github.com/arijitdutta67/MProveAndMProvisions/tree/v0.13.0.4-mprove/tests/mprovisions

[41] R. W. F. Lai, V. Ronge, T. Ruffing, D. Schröder, S. A. K. Thyagarajan, and J. Wang, "Omniring: Scaling private payments without trusted setup," in *Proceedings of the*

2019 ACM SIGSAC Conference on Computer and Communications Security, ser. CCS '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 31–48. [Online]. Available: https://doi.org/10.1145/3319535.3345655

[42] R. Diestel, "Graph Theory," Springer, 3rd edition, 2005.

[43] Ristretto Encoding of Monero public key. [Online]. Available: https://github.com/suyash67/curve25519-dalek/blob/ddbffc9/src/edwards.rs#L1167

[44] MProve+ simulation code. [Online]. Available: https://github.com/suyash67/MProvePlus-Ristretto

[45] MProve simulation code. [Online]. Available: https://github.com/suyash67/MProve-Ristretto

[46] J. Camenisch and M. Stadler, "Proof systems for general statements about discrete logarithms," Dept. of Computer Science, ETH Zürich, Tech. Rep. 260, Mar 1997.

[47] J. Camenisch, "Group signature schemes and payment systems based on the discrete logarithm problem," Ph.D. dissertation, ETH Zurich, 1998.

[48] Certicom Research. (2010) SEC 2: Recommended Elliptic Curve Domain Parameters. [Online]. Available: http://www.secg.org/sec2-v2.pdf

[49] Pedersen commitment implementation in Grin. [Online]. Available: https://github.com/mimblewimble/secp256k1-zkp/

[50] C. P. Schnorr, "Efficient signature generation by smart cards," Journal of Cryptography, 4, pp. 161–174, 1991.

[51] R. Cramer, I. Damgård, and B. Schoenmakers, "Proofs of partial knowledge and simplified design of witness hiding protocols," in *Advances in Cryptology — CRYPTO '94*, 1994, pp. 174–187.

[52] Grin rust-secp256k1-zkp github repository. [Online]. Available: https://github.com/mimblewimble/secp256k1-zkp/

[53] Revelio simulation code. [Online]. Available: https://github.com/avras/revelio

[54] S. Bagad and S. Vijayakumaran, "Performance trade-offs in design of mimblewimble proofs of reserves," in *2020 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, 2020, pp. 367–377.

[55] Wiktionary contributors. quisquis — Wiktionary, the free dictionary. [Accessed 02-Aug-2019]. [Online]. Available: https://en.wiktionary.org/wiki/quisquis

[56] ——. nummatus — Wiktionary, the free dictionary. [Accessed 02-Aug-2019]. [Online]. Available: https://en.wiktionary.org/wiki/nummatus

[57] What are zk-SNARKs? [Accessed 02-Aug-2019]. [Online]. Available: https://z.cash/technology/zksnarks/

[58] Wiktionary contributors. simplus — Wiktionary, the free dictionary. [Accessed 02-Aug-2019]. [Online]. Available: https://en.wiktionary.org/wiki/simplus

[59] Grin rust-secp256k1-zkp github repository. [Online]. Available: https://github.com/mimblewimble/secp256k1-zkp/

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

[61] The size of the spent key images set in monero. [Online]. Available: https://monero.stackexchange.com/questions/12312/what-is-the-size-of-the-set-of-spent-key-images-in-monero

[62] Monero block explorer. [Online]. Available: https://moneroblocks.info/stats/transaction-stats

[63] F. Bao, R. H. Deng, and H. Zhu, "Variations of Diffie-Hellman problem," in *Information and Communications Security*, 2003, pp. 301–312.

[64] R. Cramer, I. Damgård, and B. Schoenmakers, "Proofs of partial knowledge and simplified design of witness hiding protocols," in *Advances in Cryptology — CRYPTO '94*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1994, pp. 174–187.

Appendix A

Proofs and Signatures Generation Procedures in Chapter 2

A.1 Proof of SHVZK Property of MProvisions

In this section, we prove that MProvisions is a perfect special-honest-verifier-zero-knowledge (SHVZK) protocol. The proof is similar to the proofs given in [1, Appendix C]. The completeness of the protocol can be checked by calculation. Proof of witness extended emulation and honest-verifier-zero-knowledge are as follows.

Claim 1. *Let P^* be an efficient prover. There exists a PPT algorithm (extractor $\mathcal{E}$) for the MProvisions protocol such that for each $i \in \{1, 2, \ldots, n\}$ and any pair of accepting transcripts with the same $A_i^{(\cdot)}$ and $c_i \neq c_i'$, $\mathcal{E}$ can output $(s_i \in \{0,1\}, k_i, e_i, f_i, \hat{x}_i)$ satisfying the statement in (2.26) for all $i \in \{1, 2, \ldots, n\}$.*

Proof. We give the extractor protocol as follows.
For $i \in \{1, 2, \ldots, n\}$,

1. Run P^* to obtain $A_i^{(1)}$, $A_i^{(2)}$, $A_i^{(3)}$, $A_i^{(4)}$, $A_i^{(5)}$.
2. Send $c_i \xleftarrow{\$} \mathbb{Z}_q$ to P^*.

3. P^* will publish r_{s_i}, r_{k_i}, r_{e_i}, r_{f_i}, $r_{\hat{x_i}}$ s.t.

$$\begin{aligned} r_{s_i}C_i + r_{k_i}G &= c_iB_i + A_i^{(1)} \\ r_{s_i}I_i + r_{e_i}H &= c_iM_i + A_i^{(2)} \\ r_{s_i}P_i + r_{f_i}H &= c_iN_i + A_i^{(3)} \\ r_{\hat{x_i}}\ H_p(P_i) + r_{e_i}H &= c_iM_i + A_i^{(4)} \\ r_{\hat{x_i}}G + r_{f_i}H &= c_iN_i + A_i^{(5)}. \end{aligned}$$

4. Rewind P^* to right after step 1 of the protocol.

5. Send $c_i' \xleftarrow{\$} \mathbb{Z}_q \setminus \{c_i\}$ to P^*.

6. P^* will publish r'_{s_i}, r'_{k_i}, r'_{e_i}, r'_{f_i}, $r'_{\hat{x_i}}$ s.t.

$$\begin{aligned} r'_{s_i}C_i + r'_{k_i}G &= c_i'B_i + A_i^{(1)} \\ r'_{s_i}I_i + r'_{e_i}H &= c_i'M_i + A_i^{(2)} \\ r'_{s_i}P_i + r'_{f_i}H &= c_i'N_i + A_i^{(3)} \\ r'_{\hat{x_i}}\ H_p(P_i) + r'_{e_i}H &= c_i'M_i + A_i^{(4)} \\ r'_{\hat{x_i}}G + r'_{f_i}H &= c_i'N_i + A_i^{(5)}. \end{aligned}$$

7. In both the cases, the equations in step (1d) of the MProvisions protocol hold (see Section 2.4.1). So output: $s_i = \frac{r_{s_i} - r'_{s_i}}{c_i - c_i'}$, $k_i = \frac{r_{k_i} - r'_{k_i}}{c_i - c_i'}$, $e_i = \frac{r_{e_i} - r'_{e_i}}{c_i - c_i'}$, $f_i = \frac{r_{f_i} - r'_{f_i}}{c_i - c_i'}$, $\hat{x_i} = \frac{r_{\hat{x_i}} - r'_{\hat{x_i}}}{c_i - c_i'}$.

□

Claim 2. ***(Honest-verifier-zero-knowledge)*** *There exists a PPT simulator $\mathcal{S}$ that, given $P_i, C_i, B_i, I_i, M_i, N_i$ for each $i \in \{1, 2, \ldots, n\}$, can produce a transcript that has the same distribution as a transcript between the prover and an honest verifier.*

Proof. We give the simulator ($\mathcal{S}$) protocol as follows.
For $i \in \{1, 2, \ldots, n\}$,

1. Choose r_{s_i}, r_{k_i}, r_{e_i}, r_{f_i}, $r_{\hat{x_i}}$ and the challenge c_i uniformly at random from $\mathbb{Z}_q$.

2. Let

$$
\begin{aligned}
A_i^{(1)} &= r_{s_i} C_i + r_{k_i} G - c_i B_i, \\
A_i^{(2)} &= r_{s_i} I_i + r_{e_i} H - c_i M_i, \\
A_i^{(3)} &= r_{s_i} P_i + r_{f_i} H - c_i N_i, \\
A_i^{(4)} &= r_{\hat{x}_i}\ H_p(P_i) + r_{e_i} H - c_i M_i, \\
A_i^{(5)} &= r_{\hat{x}_i} G + r_{f_i} H - c_i N_i.
\end{aligned}
$$

3. Publish $(A_i^{(1)}, A_i^{(2)}, A_i^{(3)}, A_i^{(4)}, A_i^{(5)}, c_i, r_{s_i}, r_{k_i}, r_{e_i}, r_{f_i}, r_{\hat{x}_i})$ as the transcript.

In the simulated transcript, the six scalars $c_i, r_{s_i}, r_{k_i}, r_{e_i}, r_{f_i}, r_{\hat{x}_i}$ are chosen uniformly and independently from $\mathbb{Z}_q$ and the remaining components of the transcript, namely $A_i^{(1)}$, $A_i^{(2)}$, $A_i^{(3)}, A_i^{(4)}, A_i^{(5)}$ are deterministic functions of these scalars. In the actual protocol, the five scalars $u_i^{(1)}, u_i^{(2)}, u_i^{(3)}, u_i^{(4)}, u_i^{(5)}$ are chosen uniformly and independently from $\mathbb{Z}_q$. By the calculation in step (1d) of the MProvisions protocol, the quantities $r_{s_i}, r_{k_i}, r_{e_i}, r_{f_i}, r_{\hat{x}_i}$ in the actual transcript are uniformly distributed, and independent of each other and c_i (which is itself uniformly distributed). The quantities $A_i^{(1)}$, $A_i^{(2)}$, $A_i^{(3)}, A_i^{(4)}, A_i^{(5)}$ in the actual transcript are fixed by the same deterministic functions (given in step (1e) of the MProvisions protocol) of $c_i, r_{s_i}, r_{k_i}, r_{e_i}, r_{f_i}, r_{\hat{x}_i}$ which fix these quantities in the simulated transcript. Hence the actual protocol transcript and the simulated transcript are identically distributed. □

Finally, the proof that $s_i \in \{0, 1\}$ for each $i \in \{1, 2, \ldots, n\}$, follows directly from the proof of Protocol 4, given in Appendix B of the Provisions [1] paper.

A.2 Ring Signature Generation in MProve

In this section, we describe the calculation of the regular ring signature γ_i corresponding to step 4 of the MProve proof generation procedure described in Section 2.4.2. The calculation is the same as the algorithm described in Section 2.4.2 with the public key list $\mathcal{Q}^i = (C_i', C_i' - C_i)$. We use different notation to differentiate the terms from those used in the linkable ring signature calculation of Appendix A.3.

1. For i such that $P_i \in P_{\text{known}}$, the private key z_i corresponding to the public key $C_i' = z_i G$ is used to create the regular ring signature $\gamma_i = (d_0^i, t_0^i, t_1^i)$ where

- Using randomly chosen β_i from $\mathbb{Z}_q$, d_1^i is calculated as $d_1^i = H_s(\mathcal{Q}^i, m, S_0^i)$ where $S_0^i = \beta_i G$.
- Using randomly chosen t_1^i from $\mathbb{Z}_q$, d_0^i is calculated as $d_0^i = H_s(\mathcal{Q}^i, m, S_1^i)$ where $S_1^i = t_1^i G + d_1^i(C_i' - C_i)$.
- The value t_0^i is set to $\beta_i - d_0^i z_i$.

2. For i such that $P_i \notin P_{\text{known}}$, the private key z_i corresponding to the public key $C_i' - C_i = z_i G$ is used to create the regular ring signature $\gamma_i = (d_0^i, t_0^i, t_1^i)$ where
   - Using randomly chosen β_i from $\mathbb{Z}_q$, d_0^i is calculated as $d_0^i = H_s(\mathcal{Q}^i, m, S_1^i)$ where $S_1^i = \beta_i G$.
   - Using randomly chosen t_0^i from $\mathbb{Z}_q$, d_1^i is calculated as $d_1^i = H_s(\mathcal{Q}^i, m, S_0^i)$ where $S_0^i = t_0^i G + d_0^i C_i'$.
   - The value t_1^i is set to $\beta_i - d_1^i z_i$.

A.3 Linkable Ring Signature Generation in MProve

In this section, we describe the calculation of the linkable ring signature σ_i corresponding to step 5 of the MProve proof generation procedure described in Section 2.4.2. The calculation is the same as the algorithm described in Section 2.2.2 with the public key list $\mathcal{R}^i = (P_i, C_i' - C_i)$.

Let $x_i \in \mathbb{Z}_q$ be the private key corresponding to P_i, i.e. $P_i = x_i G$.

1. For i such that $P_i \in P_{\text{known}}$, the private key x_i is used to create the linkable ring signature $\sigma_i = (I_i, c_0^i, s_0^i, s_1^i)$ where $I_i = x_i H_p(P_i)$ is the key image.
   - Using randomly chosen α_i from $\mathbb{Z}_q$, c_1^i is calculated as

$$\begin{aligned} L_0^i &= \alpha_i G, \;\; R_0^i = \alpha_i H_p(P_i), \\ c_1^i &= H_s(\mathcal{R}^i, m, L_0^i, R_0^i). \end{aligned} \tag{A.1}$$

   - Using randomly chosen s_1^i from $\mathbb{Z}_q$, c_0^i is calculated as

$$\begin{aligned} L_1^i &= s_1^i G + c_1^i(C_i' - C_i), \\ R_1^i &= s_1^i H_p(C_i' - C_i) + c_1^i I_i, \\ c_0^i &= H_s(\mathcal{R}^i, m, L_1^i, R_1^i). \end{aligned} \tag{A.2}$$

- The value s_0^i is set to $\alpha_i - c_0^i x_i$.

2. For i such that $P_i \notin P_{\text{known}}$, the private key z_i corresponding to the public key $C'_i - C_i = z_i G$ is used to create the linkable ring signature $\sigma_i = (I_i, c_0^i, s_0^i, s_1^i)$ where $I_i = z_i H_p(C'_i - C_i)$ is the key image.

   - Using randomly chosen α_i from $\mathbb{Z}_q$, c_0^i is calculated as

$$\begin{aligned} L_1^i &= \alpha_i G, \;\; R_1^i = \alpha_i H_p(C'_i - C_i), \\ c_0^i &= H_s(\mathcal{R}^i, m, L_1^i, R_1^i). \end{aligned} \tag{A.3}$$

   - Using randomly chosen s_0^i from $\mathbb{Z}_q$, c_1^i is calculated as

$$\begin{aligned} L_0^i &= s_0^i G + c_0^i P_i, \;\; R_0^i = s_0^i H_p(P_i) + c_0^i I_i, \\ c_1^i &= H_s(\mathcal{R}^i, m, L_0^i, R_0^i). \end{aligned} \tag{A.4}$$

   - The value s_1^i is set to $\alpha_i - c_1^i x_i$.

A.4 Proof of Theorem 2.2

Let $C_{\text{res}} = y_{\text{res}} G + a_{\text{res}} H$ and let $C_i = y_i G + a_i H$ be the Pedersen commitment corresponding to the ith one-time address in the anonymity set. As the exchange was successful in creating the linkable ring signatures σ_i, it knows $w_i \in \mathbb{Z}_q$ such that $w_i G = C'_i - C_i$ for $i \in \mathcal{I}_{\text{unknown}}$, i.e. $P_i \notin \mathcal{P}_{\text{own}}$, with overwhelming probability. This is because creating σ_i without knowing either of the private keys corresponding to the public keys $\{P_i, C'_i - C_i\}$ amounts to a forgery of the linkable ring signature which is possible only with negligible probability.

As the exchange was successful in creating the regular ring signatures γ_i, it knows $w_i \in \mathbb{Z}_q$ such that $w_i G$ is equal to either C'_i or $C'_i - C_i$ with overwhelming probability. Let $\mathcal{I}_1, \mathcal{I}_2$ be a partition of the set $\mathcal{I}_{\text{own}}$ such that $w_i G = C'_i$ for $i \in \mathcal{I}_1$ and $w_i G = C'_i - C_i$ for $i \in \mathcal{I}_2$.

As $\mathcal{I}_{\text{unknown}} \cup \mathcal{I}_1 \cup \mathcal{I}_2 = \{1, 2, \ldots, n\}$, we can rearrange the terms in equation (2.31)

to get

$$\sum_{i\in\mathcal{I}_1} C_i = C_{\text{res}} + \sum_{i\in\mathcal{I}_1} C'_i + \sum_{i\in\mathcal{I}_{\text{unknown}}\cup\mathcal{I}_2} (C'_i - C_i)$$

$$\implies \sum_{i\in\mathcal{I}_1} y_i G + \sum_{i\in\mathcal{I}_1} a_i H = y_{\text{res}} G + a_{\text{res}} H + \sum_{i=1}^{n} w_i G. \tag{A.5}$$

Then equation (A.5) implies that $a_{\text{res}} = \sum_{i\in\mathcal{I}_1} a_i$ as there are no other H terms in the RHS of equation (A.5). □

A.5 Proof of Theorem 2.3

We need to show that $\mathcal{D}_{\mathsf{MPe}}$ cannot distinguish between MPe_0 and MPe_1 with a probability that is non-negligibly better than that of random guessing. The proofs MPe_0 and MPe_1 have the elements $\left(\mathbf{P}^{(i)}, \mathbf{C}^{(i)}, m_i\right)_{i=1}^{f(\lambda)}$ common between them. In addition to these elements, the actual proof MPe_1 has $\left(\mathbf{C'}^{(i)}, \mathbf{\Gamma}^{(i)}, \mathbf{\Sigma}^{(i)}, C^{(i)}_{\text{res}}\right)_{i=1}^{f(\lambda)}$. Instead of having these elements, the simulated proof MPe_0 has $\left(\hat{\mathbf{C}}'^{(i)}, \hat{\mathbf{\Gamma}}^{(i)}, \hat{\mathbf{\Sigma}}^{(i)}, \hat{C}^{(i)}_{\text{res}}\right)_{i=1}^{f(\lambda)}$. Let us consider each of these elements.

Consider a particular element $C'_{i,k} \in \mathbf{C'}^{(i)}$ and $\hat{C}'_{i,k} \in \hat{\mathbf{C}}'^{(i)}$. The element $C'_{i,k} = z_{i,k}G$ when $P_{i,k} \in \mathcal{P}^{(i)}_{\text{own}}$ and $C'_{i,k} = C_{i,k} + z_{i,k}G$ when $P_{i,k} \notin \mathcal{P}^{(i)}_{\text{own}}$. Here $z_{i,k} = H_s(k_{\text{exch}}, P_{i,k})$, $k_{\text{exch}} \xleftarrow{\$} \mathbb{Z}_q$. Whereas $\hat{C}'_{i,k} = y_{i,k}G$ for some $y_{i,k} = H_s(k_{\text{sim}}, P_{i,k})$, $k_{\text{sim}} \xleftarrow{\$} \mathbb{Z}_q$. As $H_s(\cdot)$ is modeled as a random oracle, both $C'_{i,k}, \hat{C}'_{i,k}$ are uniformly distributed over $\mathbb{G}$ for all $i \in \{1, 2, \ldots, f(\lambda)\}$ and $k \in \{1, 2, \ldots, n_i\}$. Hence these elements do not aid $\mathcal{D}_{\mathsf{MPe}}$ in estimating b. As $C'_{i,k}$ and $\hat{C}'_{i,k}$ have identical distribution, both $C^{(i)}_{\text{res}}$ and $\hat{C}^{(i)}_{\text{res}}$, determined uniquely by the following equations, are also identically distributed,

$$C^{(i)}_{\text{res}} = \sum_{k=1}^{n_i} C_{i,k} - \sum_{k=1}^{n_i} C'_{i,k}, \tag{A.6}$$

$$\hat{C}^{(i)}_{\text{res}} = \sum_{k=1}^{n_i} C_{i,k} - \sum_{k=1}^{n_i} \hat{C}'_{i,k}. \tag{A.7}$$

Hence $\mathcal{D}_{\mathsf{MPe}}$ can discard them as well while estimating b.

Now consider a particular ring signature $\gamma_{i,k} = (d_0^{i,k}, t_0^{i,k}, t_1^{i,k}) \in \mathbf{\Gamma}^{(i)}$ and $\hat{\gamma}_{i,k} = (\hat{d}_0^{i,k}, \hat{t}_0^{i,k}, \hat{t}_1^{i,k}) \in \hat{\mathbf{\Gamma}}^{(i)}$. They also do not help $\mathcal{D}_{\mathsf{MPe}}$ in predicting the value of b. This is because the corresponding elements are either randomly chosen from $\mathbb{Z}_q$, uniquely determined by randomly chosen elements in $\mathbb{Z}_q$, or generated by the random oracle

$H_s(\cdot)$ (see Appendix A.2). Now consider a particular linkable ring signature $\sigma_{i,k} = (I_{i,k}, c_0^{i,k}, s_0^{i,k}, s_1^{i,k}) \in \mathbf{\Sigma}^{(i)}$ and $\hat{\sigma}_{i,k} = (\hat{I}_{i,k}, \hat{c}_0^{i,k}, \hat{s}_0^{i,k}, \hat{s}_1^{i,k}) \in \hat{\mathbf{\Sigma}}^{(i)}$. $\mathcal{D}_{\mathsf{MPe}}$ can discard $(c_0^{i,k}, s_0^{i,k}, s_1^{i,k})$ in $\sigma_{i,k}$ and $(\hat{c}_0^{i,k}, \hat{s}_0^{i,k}, \hat{s}_1^{i,k})$ in $\hat{\sigma}_{i,k}$ as they are again either randomly chosen from $\mathbb{Z}_q$, uniquely determined by randomly chosen elements in $\mathbb{Z}_q$, or generated by the random oracle $H_s(\cdot)$ (see Appendix A.3).

Now consider $I_{i,k}$ in $\sigma_{i,k}$ and $\hat{I}_{i,k}$ in $\hat{\sigma}_{i,k}$. Note that if a particular $P_{i,k}$ is an exchange-owned address, then the tuple $(P_{i,k}, H_p(P_{i,k}), I_{i,k})$ is a DDH triple. To see this, let $P_{i,k} = a_{i,k}G$ and $H_p(P_{i,k}) = b_{i,k}G$ for some $b_{i,k} \in \mathbb{Z}_q$. As $\sigma_{i,k}$ in this case is signed by $P_{i,k}$, $I_{i,k} = a_{i,k}H_p(P_{i,k}) = a_{i,k}b_{i,k}G$. However for any $P_{i,k}$, the tuple $(P_{i,k}, H_p(P_{i,k}), \hat{I}_{i,k})$ is not a DDH triple with an overwhelming probability. To see this, note that $\hat{\sigma}_{i,k}$ is always signed by $\hat{C'}_{i,k} - C_{i,k} = y_{i,k}G$. Hence, $\hat{I}_{i,k} = y_{i,k}H_p(\hat{C'}_{i,k} - C_{i,k}) = c_{i,k}G$ for some $c_{i,k} \in \mathbb{Z}_q$ and $c_{i,k}$ is unlikely to be equal to $a_{i,k}b_{i,k}$.

We consolidate all the distinct one-time address $P_{i,k}$ in the $f(\lambda)$ anonymity sets in the vector $\mathbf{P}$ as,

$$\mathbf{P} = \bigcup_{i=1}^{f(\lambda)} \mathbf{P}^{(i)} = (P_1, P_2, \ldots, P_N), \tag{A.8}$$

where N is the number of distinct one-time addresses in the $f(\lambda)$ anonymity sets. We define the following vectors containing the hashes of the one-time addresses P_i in $\mathbf{P}$, and the corresponding key images I_i and $\hat{I}_i$.

$$\mathbf{H}_p = (H_p(P_1), H_p(P_2), \ldots, H_p(P_N)), \tag{A.9}$$

$$\mathbf{I} = (I_1, I_2, \ldots, I_N), \tag{A.10}$$

$$\hat{\mathbf{I}} = (\hat{I}_1, \hat{I}_2, \ldots, \hat{I}_N). \tag{A.11}$$

Let there be s exchange-owned one-time addresses in $\mathbf{P}$. If $P_{j_1}, P_{j_2}, \ldots, P_{j_s} \in \mathbf{P}$ are exchange-owned addresses, then the tuple $(\mathbf{P}, \mathbf{H}_p, \mathbf{I})$ contains s DDH triples as follows,

$$\begin{gathered}(P_{j_1}, H_p(P_{j_1}), I_{j_1}),\\ (P_{j_2}, H_p(P_{j_2}), I_{j_2}),\\ \vdots\\ (P_{j_s}, H_p(P_{j_s}), I_{j_s}).\end{gathered}$$

However as discussed above, the tuple $\left(\mathbf{P}, \mathbf{H}_p, \hat{\mathbf{I}}\right)$ contains no DDH triple, except with a negligible probability. A PPT distinguisher $\mathcal{D}_{\mathsf{MPe}}$ who can successfully predict b in the

MProvePriv experiment with a probability which is non-neglibly better than $\frac{1}{2}$ must be able to distinguish between the tuples $(\mathbf{P}, \mathbf{H}_p, \mathbf{I})$ and $\left(\mathbf{P}, \mathbf{H}_p, \hat{\mathbf{I}}\right)$.

Now suppose in the MProvePriv experiment, $\mathscr{D}_{\mathsf{MPe}}$ is replaced by $\mathfrak{D}_{\mathsf{MPe}}$ which receives $(\mathbf{P}, \mathbf{H}_p, \mathbf{I}_b, j_1, \ldots, j_s)$ instead of MPe_b. Here $\mathbf{I}_0$ is $\hat{\mathbf{I}}$ and $\mathbf{I}_1$ is $\mathbf{I}$. Notice, now $\mathfrak{D}_{\mathsf{MPe}}$ has the number of the exchange-owned addresses and their locations in the vector $\mathbf{P}$ as additional and relevant information compared to $\mathscr{D}_{\mathsf{MPe}}$ when predicting the value of b is of concern. The additional information that $\mathscr{D}_{\mathsf{MPe}}$ had before does not help $\mathscr{D}_{\mathsf{MPe}}$ in predicting the value of b as discussed above. Hence for every PPT adversary $\mathscr{D}_{\mathsf{MPe}}$, there exists another PPT adversary $\mathfrak{D}_{\mathsf{MPe}}$ such that,

$$\left| \Pr[\mathscr{D}_{\mathsf{MPe}}(\mathsf{MPe}_b) = b] - \frac{1}{2} \right| \leq \left| \Pr[\mathfrak{D}_{\mathsf{MPe}}(\mathbf{P}, \mathbf{H}_p, \mathbf{I}_b, j_1, \ldots, j_s) = b] - \frac{1}{2} \right|. \tag{A.12}$$

We define the following notation.

$$\widetilde{\mathsf{MPe}}_b = \mathbf{P}, \mathbf{H}_p, \mathbf{I}_b, j_1, \ldots, j_s, \tag{A.13}$$

$$\widetilde{\mathsf{MPe}}_0 = \mathbf{P}, \mathbf{H}_p, \mathbf{I}_0, j_1, \ldots, j_s, \tag{A.14}$$

$$\widetilde{\mathsf{MPe}}_1 = \mathbf{P}, \mathbf{H}_p, \mathbf{I}_1, j_1, \ldots, j_s. \tag{A.15}$$

The RHS of the inequality (A.12) can be alternatively represented as,

$$\begin{aligned}
&\left| \Pr[\mathfrak{D}_{\mathsf{MPe}}(\widetilde{\mathsf{MPe}}_b) = b] - \frac{1}{2} \right| \\
&= \left| \Pr[b = 0] \Pr[\mathfrak{D}_{\mathsf{MPe}}(\widetilde{\mathsf{MPe}}_b) = b | b = 0] + \Pr[b = 1] \Pr[\mathfrak{D}_{\mathsf{MPe}}(\widetilde{\mathsf{MPe}}_b) = b | b = 1] - \frac{1}{2} \right|, \\
&= \left| \frac{1}{2} \Pr[\mathfrak{D}_{\mathsf{MPe}}(\widetilde{\mathsf{MPe}}_0) = 0] + \frac{1}{2} \Pr[\mathfrak{D}_{\mathsf{MPe}}(\widetilde{\mathsf{MPe}}_1) = 1] - \frac{1}{2} \right|, \\
&= \left| \frac{1}{2} \left(1 - \Pr[\mathfrak{D}_{\mathsf{MPe}}(\widetilde{\mathsf{MPe}}_0) = 1]\right) + \frac{1}{2} \Pr[\mathfrak{D}_{\mathsf{MPe}}(\widetilde{\mathsf{MPe}}_1) = 1] - \frac{1}{2} \right|, \\
&= \frac{1}{2} \left| \Pr[\mathfrak{D}_{\mathsf{MPe}}(\widetilde{\mathsf{MPe}}_0) = 1] - \Pr[\mathfrak{D}_{\mathsf{MPe}}(\widetilde{\mathsf{MPe}}_1) = 1] \right|, \\
&= \frac{1}{2} \left| \Pr[\mathfrak{D}_{\mathsf{MPe}}(\mathbf{P}, \mathbf{H}_p, \mathbf{I}_0, j_1, \ldots, j_s) = 1] - \Pr[\mathfrak{D}_{\mathsf{MPe}}(\mathbf{P}, \mathbf{H}_p, \mathbf{I}_1, j_1, \ldots, j_s) = 1] \right|.
\end{aligned} \tag{A.16}$$

From equation (A.16) and inequality (A.12), to prove Theorem 2.3, it is enough to prove the following claim.

Claim 3. *For every PPT $\mathfrak{D}_{\mathsf{MPe}}$ in the MProvePriv experiment, there exists a negligible function $\mathsf{negl}_1(\lambda)$ of the security parameter λ such that,*

$$\left| \Pr[\mathfrak{D}_{\mathsf{MPe}}(\mathbf{P}, \mathbf{H}_p, \mathbf{I}_0, j_1, \ldots, j_s) = 1] - \Pr[\mathfrak{D}_{\mathsf{MPe}}(\mathbf{P}, \mathbf{H}_p, \mathbf{I}_1, j_1, \ldots, j_s) = 1] \right| \leq \mathsf{negl}_1(\lambda). \tag{A.17}$$

We prove Claim 3 by contradiction. Suppose for a PPT distinguisher $\mathfrak{D}_{\mathsf{MPe}}$, there exists a polynomial $p(\lambda)$ such that,

$$\Big| \Pr[\mathfrak{D}_{\mathsf{MPe}}(\mathbf{P}, \mathbf{H}_p, \mathbf{I}_0, j_1, \ldots, j_s) = 1] - \Pr[\mathfrak{D}_{\mathsf{MPe}}(\mathbf{P}, \mathbf{H}_p, \mathbf{I}_1, j_1, \ldots, j_s) = 1] \Big| \geq \frac{1}{p(\lambda)}. \tag{A.18}$$

So $\mathfrak{D}_{\mathsf{MPe}}$ can distinguish between the two above scenarios with a non-negligible probability. We will show how to construct a DDH adversary $\mathcal{D}_{\mathsf{DDH}}$ using $\mathfrak{D}_{\mathsf{MPe}}$ as a subroutine. A DDH challenger $\mathcal{C}$ samples $d \xleftarrow{\$} \{0,1\}, x, y, z \xleftarrow{\$} \mathbb{Z}_q$. $\mathcal{C}$ sets $X = xG, Y = yG, Z = z_dG$, where $z_0 = z, z_1 = xy$. $\mathcal{C}$ sends (X, Y, Z) to $\mathcal{D}_{\mathsf{DDH}}$. $\mathcal{D}_{\mathsf{DDH}}$ outputs d' as the estimate of d. $\mathcal{D}_{\mathsf{DDH}}$ wins if $d' = d$. We construct $\mathcal{D}_{\mathsf{DDH}}$ using a *hybrid argument* [37].

Consider a hybrid simulator $\mathcal{S}^{(n)}_{\text{hyb}}$ which is given $(\mathbf{P}, \mathbf{H}_p, \mathbf{I}, \hat{\mathbf{I}}, j_1, \ldots, j_s)$, $n \in \{0, 1, 2, \ldots, s\}$ as input. $\mathcal{S}^{(n)}_{\text{hyb}}$ works as follows.

1. It produces a vector $\mathbf{I}^{(n)}$ of same length of $\mathbf{I}$ and $\hat{\mathbf{I}}$, i.e. N. It populates the j_1th, j_2th, $\ldots$, j_nth elements of $\mathbf{I}^{(n)}$ with the j_1th, j_2th, $\ldots$, j_nth elements of $\mathbf{I}$.
2. It populates the other elements of $\mathbf{I}^{(n)}$ with uniform elements from $\mathbb{G}$.
3. It outputs $(\mathbf{P}, \mathbf{H}_p, \mathbf{I}^{(n)}, j_1, \ldots, j_s)$.

Modelling $H_s(\cdot)$ as a random oracle* we have,

$$\Pr[\mathfrak{D}_{\mathsf{MPe}}(\mathbf{P}, \mathbf{H}_p, \mathbf{I}_1, j_1, \ldots, j_s) = 1] = \Pr[\mathfrak{D}_{\mathsf{MPe}}(\mathcal{S}^{(s)}_{\text{hyb}}) = 1], \tag{A.19}$$

$$\Pr[\mathfrak{D}_{\mathsf{MPe}}(\mathbf{P}, \mathbf{H}_p, \mathbf{I}_0, j_1, \ldots, j_s) = 1] = \Pr[\mathfrak{D}_{\mathsf{MPe}}(\mathcal{S}^{(0)}_{\text{hyb}}) = 1], \tag{A.20}$$

as $\mathbf{I}_0$ is $\hat{\mathbf{I}}$ and $\mathbf{I}_1$ is $\mathbf{I}$.

The construction of $\mathcal{D}_{\mathsf{DDH}}$ having (X, Y, Z) as input is as follows.

1. It queries $\mathfrak{D}_{\mathsf{MPe}}$ and obtains N, s, where $2 \leq N \leq f(\lambda), 1 \leq s \leq N - 1$.
2. It randomly chooses $k^* \xleftarrow{\$} \{1, 2, \ldots, s\}$ and $j_1, j_2, \ldots, j_s \xleftarrow{\$} \{1, 2, \ldots, N\}$.
3. It defines the following vectors,

$$\mathbf{P}^{\text{DDH}} = (P_1, \ldots, P_N),$$
$$\mathbf{H}_p^{\text{DDH}} = (Q_1, \ldots, Q_N),$$
$$\mathbf{I}^{\text{DDH}} = (R_1, \ldots, R_N).$$

*Note that the s in the subscript H_s denotes that the hash function is scalar-valued. It is not related to the number of exchange-owned addresses s.

4. It chooses $x_1, y_1, x_2, y_2, \ldots, x_{k^*-1}, y_{k^*-1} \xleftarrow{\$} \mathbb{Z}_q$. For all $l \in \{1, 2, \ldots, k^* - 1\}$, it sets,

$$P_{j_l} = x_{j_l} G, \quad Q_{j_l} = y_{j_l} G, \quad R_{j_l} = x_{j_l} y_{j_l} G.$$

   It also sets,

$$P_{j_{k^*}} = X, \quad Q_{j_{k^*}} = Y, \quad R_{j_{k^*}} = Z.$$

5. It populates the other elements of $\mathbf{P}^{\text{DDH}}$, $\mathbf{H}_p^{\text{DDH}}$, and $\mathbf{I}^{\text{DDH}}$ with uniform elements from $\mathbb{G}$.

6. It sends $\left(\mathbf{P}^{\text{DDH}}, \mathbf{H}_p^{\text{DDH}}, \mathbf{I}^{\text{DDH}}, j_1, \ldots, j_s\right)$ to $\mathfrak{D}_{\mathsf{MPe}}$ and receives b' as output. It sends b' as its response to the challenger $\mathcal{C}$.

Now we have,

$$\begin{aligned}
&\Pr\left[\mathscr{D}_{\mathsf{DDH}}(X, Y, Z) = 1 \;\middle|\; d = 0\right] \\
&= \sum_{m=1}^{s} \Pr[k^* = m] \Pr\left[\mathscr{D}_{\mathsf{DDH}}(X, Y, Z) = 1 \;\middle|\; d = 0 \wedge k^* = m\right] \\
&= \sum_{m=1}^{s} \Pr[k^* = m] \Pr\left[\mathfrak{D}_{\mathsf{MPe}}\left(\mathbf{P}^{\text{DDH}}, \mathbf{H}_p^{\text{DDH}}, \mathbf{I}^{\text{DDH}}, j_1, \ldots, j_s\right) = 1 \;\middle|\; d = 0 \wedge k^* = m\right], \\
&\overset{(1)}{=} \sum_{m=1}^{s} \frac{1}{s} \Pr\left[\mathfrak{D}_{\mathsf{MPe}}(\mathcal{S}_{\text{hyb}}^{(m-1)}) = 1\right], \\
&\overset{(2)}{=} \sum_{m=0}^{s-1} \frac{1}{s} \Pr\left[\mathfrak{D}_{\mathsf{MPe}}(\mathcal{S}_{\text{hyb}}^{(m)}) = 1\right].
\end{aligned} \tag{A.21}$$

Here equality (1) comes from the fact that when $d = 0$, $(P_{j_m}, Q_{j_m}, R_{j_m})$ is not a DDH triple. Equality (2) is obtained by simple changes in the indices of the summation. Similarly we obtain the following equation,

$$\begin{aligned}
&\Pr\left[\mathscr{D}_{\mathsf{DDH}}(X, Y, Z) = 1 \;\middle|\; d = 1\right] \\
&= \sum_{m=1}^{s} \Pr[k^* = m] \Pr\left[\mathfrak{D}_{\mathsf{MPe}}\left(\mathbf{P}^{\text{DDH}}, \mathbf{H}_p^{\text{DDH}}, \mathbf{I}^{\text{DDH}}, j_1, \ldots, j_s\right) = 1 \;\middle|\; d = 1 \wedge k^* = m\right], \\
&\overset{(3)}{=} \sum_{m=1}^{s} \frac{1}{s} \Pr\left[\mathfrak{D}_{\mathsf{MPe}}(\mathcal{S}_{\text{hyb}}^{(m)}) = 1\right].
\end{aligned} \tag{A.22}$$

Here equality (3) comes from the fact that when $d = 1$, $(P_{j_m}, Q_{j_m}, R_{j_m})$ is a DDH triple.

Now we have

$$\begin{aligned}
&\Big|\Pr\left[\mathscr{D}_{\mathsf{DDH}}(X,Y,Z)=1 \mid d=0\right]-\Pr\left[\mathscr{D}_{\mathsf{DDH}}(X,Y,Z)=1 \mid d=1\right]\Big| \\
&\overset{(4)}{=}\Big|\frac{1}{s}\left(\sum_{m=0}^{s-1}\Pr\left[\mathfrak{D}_{\mathsf{MPe}}(\mathcal{S}_{\text{hyb}}^{(m)})=1\right]-\sum_{m=1}^{s}\Pr\left[\mathfrak{D}_{\mathsf{MPe}}(\mathcal{S}_{\text{hyb}}^{(m)})=1\right]\right)\Big|, \\
&\overset{(5)}{=}\frac{1}{s}\Big|\Pr\left[\mathfrak{D}_{\mathsf{MPe}}(\mathcal{S}_{\text{hyb}}^{(0)})=1\right]-\Pr\left[\mathfrak{D}_{\mathsf{MPe}}(\mathcal{S}_{\text{hyb}}^{(s)})=1\right]\Big|, \\
&\overset{(6)}{=}\frac{1}{s}\Big|\Pr[\mathfrak{D}_{\mathsf{MPe}}(\mathbf{P},\mathbf{H}_p,\mathbf{I}_0,j_1,\ldots,j_s)=1]-\Pr[\mathfrak{D}_{\mathsf{MPe}}(\mathbf{P},\mathbf{H}_p,\mathbf{I}_1,j_1,\ldots,j_s)=1]\Big|, \\
&\overset{(7)}{\geq}\frac{1}{s}p(\lambda)=p_1(\lambda)\text{ (say)}.
\end{aligned}\tag{A.23}$$

Here the equality (4) comes from equations (A.21) and (A.22), the equality (5) comes from cancellations, the equality (6) comes from equations (A.20) and (A.19), and the inequality (7) comes from the assumption given in the inequality (A.18). However, the inequality (A.23) is a contradiction under the DDH assumption. So the inequality (A.18) cannot be true for any polynomial $p(\lambda)$. Hence Claim 3 is proved. □

A.6 Proof of Theorem 2.4

We need to show that $\mathscr{D}_{\mathsf{MPs}}$ cannot distinguish between MPs_0 and MPs_1 with a probability non-negligibly better than that of random guessing. The elements $(P_{i,j},C_{i,j})_{j=1,i=1}^{n_i,f(\lambda)}$ are common in MPs_0 and MPs_1. Now consider the elements $(B_{i,j},M_{i,j},N_{i,j},\mathsf{Tx}_{i,j})_{j=1,i=1}^{j=n_i,i=f(\lambda)}$ in MPs_1 and the elements $(\hat{B}_{i,j},\hat{M}_{i,j},\hat{N}_{i,j},\hat{\mathsf{Tx}}_{i,j})_{j=1,i=1}^{n_i,f(\lambda)}$ in MPs_0, where

$$\begin{aligned}
\mathsf{Tx}_{i,j}&=(A_{i,j}^{(1)},A_{i,j}^{(2)},A_{i,j}^{(3)},A_{i,j}^{(4)},A_{i,j}^{(5)},c_{i,j},r_{s_{i,j}},r_{k_{i,j}},r_{e_{i,j}},r_{f_{i,j}},r_{x_{i,j}^{\hat{}}}),\\
\hat{\mathsf{Tx}}_{i,j}&=(\hat{A}_{i,j}^{(1)},\hat{A}_{i,j}^{(2)},\hat{A}_{i,j}^{(3)},\hat{A}_{i,j}^{(4)},\hat{A}_{i,j}^{(5)},\hat{c}_{i,j},\hat{r}_{s_{i,j}},\hat{r}_{k_{i,j}},\hat{r}_{e_{i,j}},\hat{r}_{f_{i,j}},\hat{r}_{x_{i,j}^{\hat{}}}).
\end{aligned}$$

They are identically distributed in $\mathbb{G}$ and $\mathbb{Z}_q$ correspondingly.

When a one-time address P repeats across $f(\lambda)$ anonymity sets, the same key image I and the simulated key image $\hat{I}$ appear again because of the usage of the long term keys by the exchange as well as the simulator $\mathcal{S}_{\mathsf{MPs}}$. We consolidate all the distinct one-time address $P_{i,j}$ in the $f(\lambda)$ anonymity sets in the vector $\mathbf{P}$ as,

$$\mathbf{P}=\bigcup_{i=1,j=1}^{f(\lambda),n_i}\{P_{i,j}\}=(P_1,P_2,\ldots,P_N),\tag{A.24}$$

where N is the number of distinct one-time addresses in the $f(\lambda)$ anonymity sets. Similar to the proof for MProve given in Appendix A.5, we define the following vectors containing

the hashes of the one-time addresses P_i in $\mathbf{P}$, and the corresponding key images I_i and the simulated key image $\hat{I}_i$.

$$\mathbf{H}_p = (H_p(P_1), H_p(P_2), \ldots, H_p(P_N)), \tag{A.25}$$

$$\mathbf{I} = (I_1, I_2, \ldots, I_N), \tag{A.26}$$

$$\hat{\mathbf{I}} = (\hat{I}_1, \hat{I}_2, \ldots, \hat{I}_N). \tag{A.27}$$

Let there be s exchange-owned one-time addresses in $\mathbf{P}$. As discussed in Appendix A.5, if $P_{j_1}, P_{j_2}, \ldots, P_{j_s} \in \mathbf{P}$ are exchange-owned addresses, then the tuple $(\mathbf{P}, \mathbf{H}_p, \mathbf{I})$ contains s DDH triples as follows,

$$\begin{gathered}(P_{j_1}, H_p(P_{j_1}), I_{j_1}),\\ (P_{j_2}, H_p(P_{j_2}), I_{j_2}),\\ \vdots\\ (P_{j_s}, H_p(P_{j_s}), I_{j_s}).\end{gathered}$$

However the tuple $\left(\mathbf{P}, \mathbf{H}_p, \hat{\mathbf{I}}\right)$ contains no DDH triple with an overwhelming probability. The rest of the proof is identical to the proof given in Appendix A.5. □

Appendix B

Proofs and other Additional Aspects in Chapter 3

B.1 Difficulties in Hiding the Key Images of Source Addresses

UnspentProof. Consider a Monero user Bob who owns a one-time address P which is not spent. Bob wants to show that P is not spent without revealing the corresponding key image $I = H_p(P)^x$, where x is the secret key of P i.e. $P = G^x$. Suppose the public key pair of Bob is $(B_{\text{vk}}, B_{\text{sk}}) = (G^{b_{\text{vk}}}, G^{b_{\text{sk}}})$, where b_{vk} and b_{sk} are the secret view key and the secret spend key respectively. Let the Diffie-Hellman shared secret corresponding to P be B_{vk}^r, where $r \in \mathbb{Z}_q$. Then the secret key corresponding to P is $x = H(B_{\text{vk}}^r) + b_{\text{sk}}$ and we have,

$$I = H_p(P)^{H(B_{\text{vk}}^r)+b_{\text{sk}}}. \tag{B.1}$$

We say that a key image I is *originated* from a one-time address P if $P = G^x \wedge I = H_p(P)^x$ for the same private key x. In UnspentProof, the key images for all transactions where P has appeared as a ring member are tested. Suppose the set $\text{TX}(P) = \{txn_1, txn_2, \ldots, txn_n\}$ represents the set of all transactions where P has appeared as a ring member. Each transaction in $\text{TX}(P)$ has one or more key images. UnspentProof proves that each such key image in $\text{TX}(P)$ is not originated from P. In this way, P is proved to be unspent without revealing I. However, to execute the proof, the verifier must know the Diffie-Hellman shared secret B_{vk}^r. Let $I_?$ be a key image which has appeared in one

of the transactions in TX(P). Using $I_?$, B^r_{vk}, and P, the verifier computes the following quantity, termed as the *partial spend image*.

$$I_{?,s} = I_? H_p(P)^{-H(B^r_{\text{vk}})}. \tag{B.2}$$

From Equation (B.1) and (B.2), if $I = I_?$ i.e. P is spent in the transaction being tested, then we have,

$$I_{?,s} = H_p(P)^{b_{\text{sk}}}. \tag{B.3}$$

UnspentProof consists of two multi-base proof of knowledge signatures which are non-interactive Schnorr-like proofs [23, 24]. They involve sets of base elements and public keys. First, there is a three-base signature σ_3 with a base set $\{G, B_{\text{sk}}, I_{?,s}\}$. Let the set of public keys for σ_3 be $\{Q, R, S\}$. The signature σ_3 proves the knowledge of a secret scalar k such that the following holds,

$$Q = G^k \wedge R = B^k_{\text{sk}} \wedge S = I^k_{?,s}. \tag{B.4}$$

Here, σ_3 is signed with b_{sk} i.e. $k = b_{\text{sk}}$. So we have the set of public keys $\{Q, R, S\} = \{B_{\text{sk}}, B^{b_{\text{sk}}}_{\text{sk}}, I^{b_{\text{sk}}}_{?,s}\}$. There is also a two-base signature σ_2 with a base set $\{G, H_p(P)\}$. Signature σ_2 proves knowledge of a secret scalar k' such that,

$$X = G^{k'} \wedge Y = H_p(P)^{k'}, \tag{B.5}$$

where $\{X, Y\}$ is the set of public keys for σ_2. Here, σ_2 is signed with $b_{\text{sk}} * b_{\text{sk}}$ i.e. $k' = b_{\text{sk}} * b_{\text{sk}}$. So the set of public keys for σ_2 is $\{X, Y\} = \{B^{b_{\text{sk}}}_{\text{sk}}, H_p(P)^{b_{\text{sk}} * b_{\text{sk}}}\}$. When σ_3 is signed with b_{sk} and σ_2 is signed with $b_{\text{sk}} * b_{\text{sk}}$, we have $S = I^{b_{\text{sk}}}_{?,s}$ and $Y = H_p(P)^{b_{\text{sk}} * b_{\text{sk}}}$. From Equation (B.3), it follows $S = Y$ only if $I = I_?$. The UnspentProof protocol proceeds as follows.

1. Both the prover and the verifier compute the base sets of σ_3 and σ_2.

2. The prover generates the signatures σ_3 and σ_2 and sends them to the verifier along with the associated sets of public keys i.e. $\{Q, R, S\}$ and $\{X, Y\}$.

3. The verifier checks if (a) σ_3 and σ_2 are correct, and (b) $R = X$. The conditions (a) and (b) ensure that σ_3 is signed with b_{sk} and σ_2 is signed with $b_{\text{sk}} * b_{\text{sk}}$. If any of them does not hold, the verifier rejects the proof. Otherwise she proceeds to the next step.

4. If $S \neq Y$, the verifier declares that P is not spent in the transaction under test i.e. $I_? \neq I$. If $S = Y$, P is declared to be spent.

The same procedure is followed for every key image of every transaction in TX(P). In each case, the same σ_2 can be used whereas σ_3 changes every time. If the verification passes for every transaction in TX(P), then P is declared to be unspent. As any PPT adversary can forge a proof of knowledge signature only with a negligible probability, P is an unspent address with a probability overwhelmingly close to 1.

Hence by UnspentProof, we can show that a particular one-time address is not spent without revealing its key image. However, to execute UnspentProof, the prover must reveal the Diffie-Hellman shared secret (B_{vk}^r in the above example) to let the verifier calculate the partial spend image ($I_{?,s}$ in the above example). The verifier needs to calculate the partial spend image i.e. base of σ_3 herself. Suppose in the above example the verifier does not compute the partial spend image herself and the prover sends it to her. This allows the prover to cheat by sending any element other than $I_? H_p(P)^{-H(B_{\text{vk}}^r)}$ as $I_{?,s}$. Then even if P is spent in the transaction, σ_3 and σ_2 would be correct and $R = X$, $S \neq Y$ would hold. Hence the verifier will declare P is unspent in spite of P being spent in the transaction.

In spite of the novelty and the simplicity, UnspentProof has the following privacy concerns.

- To execute UnspentProof, revealing the Diffie-Hellman shared secret B_{vk}^r is unavoidable. However an entity which gets to know the Diffie-Hellman shared secret corresponding to a one-time address can easily obtain the amount associated with the one-time address using the method discussed in Section 2.2.3. If UnspentProof is to be used in the MProve+ protocol to prove that the source addresses are not spent, then the corresponding Diffie-Hellman shared secrets have to be revealed. Using those secrets, any adversary can calculate the total reserves amount.

- The base set of the signature σ_3 contains B_{sk}. Also to prove that the Diffie-Hellman shared secret B_{vk}^r is authentic, the prover Bob needs to produce a two-base signature with the secret key b_{vk}, the base set $\{G, R\}$ and the public key set $\{G^{b_{\text{vk}}}, R^{b_{\text{vk}}}\} = \{B_{\text{vk}}, B_{\text{vk}}^r\}$. This signature reveals B_{vk}. So to prove that Bob owns a unspent one-time address P using UnspentProof, Bob needs to reveal the public key pair $\{B_{\text{vk}}, B_{\text{sk}}\}$ cor-

responding to the one-time address P. This leads to the violation of the unlinkability that the exchange enjoys in Monero network.

The above mentioned reasons forbid the MProve+ protocol from using UnspentProof as a primitive.

Zero-knowledge set non-membership proof. Another possible way to avoid revealing key images in proof of reserves protocols is to propose a zero-knowledge set non-membership proof. Let the set of key images which have appeared on the Monero blockchain be $\mathcal{I}$. Recall that the set of key images of the exchange-owned source addresses is defined as $\mathbf{I}$ and the anonymity set of one-time addresses is defined as $\mathbf{P}$. We need a proof of reserves protocol which satisfies the following requirements.

1. The protocol needs a scheme that makes the set $\mathbf{I}$, a *zero-knowledge* set so that any information regarding its elements or size is not revealed. The verifier should be able to verify that no element in $\mathcal{I}$ is a member of this set $\mathbf{I}$.

2. For each key image $I_i \in \mathbf{I}$ and some $P_i \in \mathbf{P}$, the prover should be able to show the knowledge of the secret key $x_i \in \mathbb{Z}_q$ such that $P_i = G^{x_i} \wedge I_i = H_p(P_i)^{x_i}$ hold. The verifier also needs to ensure that all key images in $\mathbf{I}$ are distinct. This is to ensure that no source amount is used more than once in calculating the total reserves amount. So we need a zero-knowledge set non-membership proof and also a proof that the elements of the zero-knowledge set satisfy certain algebraic properties.

3. Since each transaction input in Monero has a corresponding key image, the set $\mathcal{I}$ is large and increases monotonically. As the non-membership proof has to be given for all elements in $\mathcal{I}$, the proof should be practical in terms of the size, the generation time, and the verification time.*

4. The security of Monero is based on the discrete logarithm assumption, the decisional Diffie-Hellman assumption, and the random oracle assumption for hash functions. Monero does not need any trusted setup. So it is desirable that a proof of reserves protocol does not use any primitive based on some other assumptions or a trusted setup.

*As of July 24, 2020, there are about 3.58×10^7 key images in the Monero blockchain [61]. Also, the average number of transactions per day (in the last one year) in Monero is ≈ 9000 [62]. Since each transaction contains 2 inputs on an average, the set $\mathcal{I}$ grows approximately by 18,000 every day.

We are not aware of any scheme which meets all the criteria mentioned above.

B.2 Proof of Theorem 3.3

Let the $f(\lambda)$ MProve+ proofs $\{\mathbf{P}^{(i)}, \mathbf{C}^{(i)}, \mathbf{H}_p^{(i)}, \mathbf{I}^{(i)}, C_{\text{res}}^{(i)}, \Pi_{\mathsf{MPP}}^{(i)}\}_{i=1}^{f(\lambda)}$ be denoted by $\mathsf{MPP}_{\mathsf{act}}$. We can extract the $f(\lambda)$ bipartite graphs $\left(\mathbf{P}^{(i)}, \mathbf{I}^{(i)}, \mathbf{P}^{(i)} \times \mathbf{I}^{(i)}\right)_{i=1}^{f(\lambda)}$ from the $f(\lambda)$ anonymity sets and the key image sets i.e. $\left\{\mathbf{P}^{(i)}, \mathbf{I}^{(i)}\right\}_{i=1}^{f(\lambda)}$. To prove that this is the sole information that can be extracted from $\mathsf{MPP}_{\mathsf{act}}$ by a PPT adversary, we construct a simulator $\mathcal{S}_{\mathsf{MPP}}$ as follows. $\mathcal{S}_{\mathsf{MPP}}$ is given only $\left\{\mathbf{P}^{(i)}, \mathbf{I}^{(i)}\right\}_{i=1}^{f(\lambda)}$. $\mathcal{S}_{\mathsf{MPP}}$ publishes $f(\lambda)$ simulated MProve+ proofs (say $\mathsf{MPP}_{\mathsf{sim}}$) keeping $\left\{\mathbf{P}^{(i)}\right\}_{i=1}^{f(\lambda)}$ as the anonymity sets. $\mathcal{S}_{\mathsf{MPP}}$ replaces the elements in $\left\{\mathbf{I}^{(i)}\right\}_{i=1}^{f(\lambda)}$ with uniform group elements $\left\{\mathbf{I}'^{(i)}\right\}_{i=1}^{f(\lambda)}$ keeping the structures of the bipartite graphs induced by $\mathsf{MPP}_{\mathsf{act}}$ as it is. We construct $\mathcal{S}_{\mathsf{MPP}}$ as follows.

1. From the given input $\left\{\mathbf{P}^{(i)}, \mathbf{I}^{(i)}\right\}_{i=1}^{f(\lambda)}$, $\mathcal{S}_{\mathsf{MPP}}$ calculates $\left(\{\mathbf{C}^{(i)}, \mathbf{H}_p^{(i)}\}_{i=1}^{f(\lambda)}\right)$ using the Monero blockchain and the hash functions used in Monero.

2. $\mathcal{S}_{\mathsf{MPP}}$ generates the simulated key images i.e. $\{\mathbf{I}'^{(i)}\}_{i=1}^{f(\lambda)}$ from $\{\mathbf{I}^{(i)}\}_{i=1}^{f(\lambda)}$ as follows. It chooses $\mathbf{I}'^{(1)}$ by sampling s_1 uniform group elements. Suppose in $\mathbf{I}'^{(1)}$, $I'_{1,k}$ is located in the position of $I_{1,k}$ in $\mathbf{I}^{(1)}$ $(k \in [s_1])$. If $I_{1,k}$ is repeated in $\mathbf{I}^{(j_1)}, \mathbf{I}^{(j_2)}, \ldots, \mathbf{I}^{(j_r)}$, $I'_{1,k}$ is placed in $\mathbf{I}'^{(j_1)}, \mathbf{I}'^{(j_2)}, \ldots, \mathbf{I}'^{(j_r)}$ in the same position where $I_{1,k}$ is located in those vectors. This is followed for all such k for which $I_{1,k}$ is repeated in the subsequent proofs. Similar procedure is followed sequentially from $i = 2$ to $f(\lambda)$. For example, $\mathbf{I}^{(j)}$ is filled only after filling $\mathbf{I}^{(1)}, \mathbf{I}^{(2)}, \ldots, \mathbf{I}^{(j-1)}$. Uniform group elements are placed in the locations of $\mathbf{I}^{(j)}$ which are not filled up already due to repetition. If any $I_{j,k} \in \mathbf{I}^{(j)} (k \in [s_j])$ repeats in subsequent proofs, $I'_{j,k}$ is placed in those positions.

3. $\mathcal{S}_{\mathsf{MPP}}$ sets $C'^{(i)}_{\text{res}} \xleftarrow{\$} \mathbb{G}$ for each $i \in [f(\lambda)]$.

4. For each $i \in [f(\lambda)]$, $\mathcal{S}_{\mathsf{MPP}}$ computes $(u_i, v_i, w_i, y_i, z_i, x_i) \xleftarrow{\$} \mathbb{Z}_q$ and sets $\mathsf{stmt}^{(i)} = \left(\mathbf{P}^{(i)}, \mathbf{C}^{(i)}, \mathbf{H}_p^{(i)}, \mathbf{I}'^{(i)}, C'^{(i)}_{\text{res}}\right)$ and the challenges as $(u_i, v_i, w_i, y_i, z_i, x_i)$. $\mathcal{S}_{\mathsf{MPP}}$ samples $A_{\mathsf{S}}^{(i)}, T_{2,\mathsf{S}}^{(i)} \xleftarrow{\$} \mathbb{G}, \boldsymbol{\ell}_{\mathsf{S}}^{(i)}, \boldsymbol{\imath}_{\mathsf{S}}^{(i)} \xleftarrow{\$} \mathbb{Z}_q^m, \tau_{\mathsf{S}}^{(i)}, r_{\mathsf{S}}^{(i)} \xleftarrow{\$} \mathbb{Z}_q$. It computes $\hat{t}_{\mathsf{S}}^{(i)} = \langle \boldsymbol{\ell}_{\mathsf{S}}^{(i)}, \boldsymbol{\imath}_{\mathsf{S}}^{(i)} \rangle$. It then

computes $S_{\mathsf{S}}^{(i)}$ and $T_{1,\mathsf{S}}^{(i)}$ as follows.

$$S_{\mathsf{S}}^{(i)} = ((F)^{-r} A_{\mathsf{S}}^{(i)} \mathbf{G}_{w_i}^{\boldsymbol{\alpha}^{(i)} - \boldsymbol{\ell}_{\mathsf{S}}^{(i)}} \mathbf{H}^{\boldsymbol{\beta}^{(i)} - \boldsymbol{\theta}^{(i)\circ -1} \circ \boldsymbol{\tau}_{\mathsf{S}}^{(i)}})^{-\frac{1}{x_i}}, \tag{B.6}$$

$$T_{1,\mathsf{S}}^{(i)} = (H^{\delta_i - \hat{t}_{\mathsf{S}}^{(i)}} G^{-\tau_{\mathsf{S}}^{(i)}} C'^{(i)}_{\text{res}}{}^{-z_i^2} T_{2,\mathsf{S}}^{(i)x_i^2})^{-\frac{1}{x_i}}, \tag{B.7}$$

where the values of $\boldsymbol{\alpha}^{(i)}, \boldsymbol{\beta}^{(i)}, \boldsymbol{\theta}^{(i)}$, and δ_i are calculated following the definitions given in Figure 3.3.

$\mathcal{S}_{\mathsf{MPP}}$ obtains the ith transcript as $\Pi_{\mathsf{MPP,S}}^{(i)} = (A_{\mathsf{S}}^{(i)}, S_{\mathsf{S}}^{(i)}, T_{1,\mathsf{S}}^{(i)}, T_{2,\mathsf{S}}^{(i)}, \boldsymbol{\ell}_{\mathsf{S}}^{(i)}, \boldsymbol{\tau}_{\mathsf{S}}^{(i)}, \hat{t}_{\mathsf{S}}^{(i)}, \tau_{\mathsf{S}}^{(i)}, r_{\mathsf{S}}^{(i)})$.

5. $\mathcal{S}_{\mathsf{MPP}}$ outputs the simulated proof as $\mathsf{MPP}_{\mathsf{sim}} = \{\mathbf{P}^{(i)}, \mathbf{C}^{(i)}, \mathbf{H}_p^{(i)}, \mathbf{I}'^{(i)}, C'^{(i)}_{\text{res}}, \Pi_{\mathsf{MPP,S}}^{(i)}\}_{i=1}^{f(\lambda)}$.

Because of step 2 of the construction of $\mathcal{S}_{\mathsf{MPP}}$, the $f(\lambda)$ bipartite graphs which can be extracted from $\left\{\mathbf{P}^{(i)}, \mathbf{I}^{(i)}\right\}_{i=1}^{f(\lambda)}$ and $\left\{\mathbf{P}^{(i)}, \mathbf{I}'^{(i)}\right\}_{i=1}^{f(\lambda)}$ are the same except having one set of different disjoint vertices (key images). To show that this is the sole information that can be extracted from $\mathsf{MPP}_{\mathsf{act}}$, we propose the following privacy experiment `MPPPriv` for the MProve+ scheme as follows.

1. $\mathcal{S}_{\mathsf{MPP}}$ sets $\mathsf{MPP}_0 = \mathsf{MPP}_{\mathsf{sim}}$ and $\mathsf{MPP}_1 = \mathsf{MPP}_{\mathsf{act}}$.
2. $\mathcal{S}_{\mathsf{MPP}}$ chooses a bit $b \xleftarrow{\$} \{0, 1\}$ randomly.
3. $\mathcal{S}_{\mathsf{MPP}}$ sends MPP_b to $\mathcal{D}_{\mathsf{MPP}}$.
4. $\mathcal{D}_{\mathsf{MPP}}$ outputs a bit $\mathcal{D}_{\mathsf{MPP}}(\mathsf{MPP}_b)$ as a prediction of b.

Note that MPP_0 is computed using no other information in MPP_1 except the $f(\lambda)$ bipartite graphs extracted from $\left\{\mathbf{P}^{(i)}, \mathbf{I}^{(i)}\right\}_{i=1}^{f(\lambda)}$. If no PPT adversary in the above experiment can successfully predict b with a probability non-negligibly better than $\frac{1}{2}$, then Theorem 3.3 holds. Now we make the following claim and prove it.

Claim 4. *For every PPT $\mathcal{D}_{MPP}$ in the `MPPPriv` experiment, there exists a negligible function $negl(\lambda)$ of the security parameter λ such that,*

$$\left| \Pr[\mathcal{D}_{MPP}(MPP_b) = b] - \frac{1}{2} \right| \leq negl(\lambda). \tag{B.8}$$

The LHS of the inequality (B.8) can be alternatively represented as,

$$
\begin{aligned}
&\left| \Pr[\mathscr{D}_{\mathsf{MPP}}(\mathsf{MPP}_b) = b] - \frac{1}{2} \right| \\
&= \left| \Pr[b=0] \Pr[\mathscr{D}_{\mathsf{MPP}}(\mathsf{MPP}_b) = b | b = 0] + \Pr[b=1] \Pr[\mathscr{D}_{\mathsf{MPP}}(\mathsf{MPP}_b) = b | b = 1] - \frac{1}{2} \right|, \\
&= \left| \frac{1}{2} \Pr[\mathscr{D}_{\mathsf{MPP}}(\mathsf{MPP}_0) = 0] + \frac{1}{2} \Pr[\mathscr{D}_{\mathsf{MPP}}(\mathsf{MPP}_1) = 1] - \frac{1}{2} \right|, \\
&= \left| \frac{1}{2} \left(1 - \Pr[\mathscr{D}_{\mathsf{MPP}}(\mathsf{MPP}_0) = 1]\right) + \frac{1}{2} \Pr[\mathscr{D}_{\mathsf{MPP}}(\mathsf{MPP}_1) = 1] - \frac{1}{2} \right|, \\
&= \frac{1}{2} \left| \Pr[\mathscr{D}_{\mathsf{MPP}}(\mathsf{MPP}_0) = 1] - \Pr[\mathscr{D}_{\mathsf{MPP}}(\mathsf{MPP}_1) = 1] \right|, \\
&= \frac{1}{2} \left| \Pr[\mathscr{D}_{\mathsf{MPP}}(\mathsf{MPP}_{\mathsf{act}}) = 1] - \Pr[\mathscr{D}_{\mathsf{MPP}}(\mathsf{MPP}_{\mathsf{sim}}) = 1] \right|. \qquad \text{(B.9)}
\end{aligned}
$$

By equation (B.9), Claim 4 can be alternatively represented as,

Claim 5. *For any PPT distinguisher $\mathscr{D}_{MPP}$, there exists a negligible function $negl_1(\lambda)$, such that*

$$\left| \Pr[\mathscr{D}_{MPP}(MPP_{act}) = 1] - \Pr[\mathscr{D}_{MPP}(MPP_{sim}) = 1] \right| \leq negl_1(\lambda). \qquad \text{(B.10)}$$

Let us consider the elements of $\mathsf{MPP}_{\mathsf{act}}$ and $\mathsf{MPP}_{\mathsf{sim}}$. For each $i \in f(\lambda)$, $\mathbf{P}^{(i)}, \mathbf{C}^{(i)}, \mathbf{H}_p^{(i)}$ are common in both of them. $C_{\text{res}}^{(i)}$ and $C'^{(i)}_{\text{res}}$ are distributed uniformly in $\mathbb{G}$. By observation, the elements of the transcript $\Pi^{(i)}_{\mathsf{MPP,S}}$ and $\Pi^{(i)}_{\mathsf{MPP}}$ are identically distributed[†].

Let us consolidate all the distinct elements of $\bigcup_{i=1}^{f(\lambda)} \mathbf{I}'^{(i)}$ and $\bigcup_{i=1}^{f(\lambda)} \mathbf{I}^{(i)}$ in the vectors $\mathbf{I}_c$ and $\mathbf{I}'_c$ respectively in a lexicographic order. Let

$$\mathbf{I}_c = \{I_1, I_2, \ldots, I_N\}, \qquad \text{(B.11)}$$

$$\mathbf{I}'_c = \{I'_1, I'_2, \ldots, I'_N\}, \qquad \text{(B.12)}$$

where $N = \left| \bigcup_{i=1}^{f(\lambda)} \mathbf{I}^{(i)} \right|$. Let us define the following sets,

$$\mathscr{P}_{\text{orig}} = \{\mathscr{P}_{\text{orig}}(I_1), \mathscr{P}_{\text{orig}}(I_2), \ldots, \mathscr{P}_{\text{orig}}(I_N)\}, \qquad \text{(B.13)}$$

$$\mathbf{H}_p(\mathscr{P}_{\text{orig}}) = \{\mathbf{H}_p(\mathscr{P}_{\text{orig}}(I_1)), \mathbf{H}_p(\mathscr{P}_{\text{orig}}(I_2)), \ldots, \mathbf{H}_p(\mathscr{P}_{\text{orig}}(I_N))\}, \qquad \text{(B.14)}$$

where the vector $\mathbf{H}_p(\mathscr{P}_{\text{orig}}(I_k)), k \in [N]$ contains the hashes of the elements of the set $\mathscr{P}_{\text{orig}}(I_k)$. Because of the way $\mathbf{I}'^{(i)}$s are populated (discussed in step 2 of the construction of $\mathsf{MPP}_{\mathsf{sim}}$), $(\mathscr{P}_{\text{orig}}, \mathbf{H}_p(\mathscr{P}_{\text{orig}}))$ are the same for both $\mathbf{I}_c$ and $\mathbf{I}'_c$. We also have,

$$\mathscr{P}_{\text{orig}}(I_k) = \mathscr{P}_{\text{orig}}(I'_k), \quad \forall I_k \in \mathbf{I}_c, I'_k \in \mathbf{I}'_c, k \in [N]. \qquad \text{(B.15)}$$

[†]The elements are either uniformly distributed in $\mathbb{G}$ and $\mathbb{Z}_q$ or fixed by the verification equations (V1),(V2), and (V3).

From the above discussion, it is clear that to prove Claim 5, it is enough[‡] to prove that for every PPT distinguisher $\mathcal{D}_{\mathsf{MPP}}$, there exists a negligible function $\mathsf{negl}'(\lambda)$ such that,

$$\Big| \Pr[\mathcal{D}_{\mathsf{MPP}}(\mathscr{P}_{\text{orig}}, \mathbf{H}_p(\mathscr{P}_{\text{orig}}), \mathbf{I}_c) = 1] - \Pr[\mathcal{D}_{\mathsf{MPP}}(\mathscr{P}_{\text{orig}}, \mathbf{H}_p(\mathscr{P}_{\text{orig}}), \mathbf{I}'_c) = 1] \Big| \leq \mathsf{negl}'(\lambda). \tag{B.16}$$

Consider the set $\mathscr{P}_{\text{orig}}(I_j), \mathscr{P}_{\text{orig}}(I'_j) \in \mathscr{P}_{\text{orig}}$, where $I_j \in \mathbf{I}_c$ and $I'_j \in \mathbf{I}'_c$ ($\mathscr{P}_{\text{orig}}(I_j) = \mathscr{P}_{\text{orig}}(I'_j)$ as discussed above). Let the set $\mathscr{P}_{\text{orig}}(I_j)$ be,

$$\mathscr{P}_{\text{orig}}(I_j) = (P_1, P_2, \ldots, P_{o_j}), \quad o_j \in \mathbb{Z}_q. \tag{B.17}$$

There exists a secret index $m_j \in [o_j]$ for which the following equation holds.

$$P_{m_j} = G^{x_{m_j}} \wedge I_j = H_p(P_{m_j})^{x_{m_j}}. \tag{B.18}$$

Let $H_p(P_{m_j}) = G^{y_j}$ for some $y_j \in \mathbb{Z}_q$. Then we have the following decisional Diffie-Hellman (DDH) triple from Equation (B.18),

$$\left(P_{m_j} = G^{x_{m_j}}, \quad H_p(P_{m_j}) = G^{y_j}, \quad I_j = G^{x_{m_j} y_j}\right). \tag{B.19}$$

However when I_j is replaced by $I'_j = G^{z_j}$ (say), $z_j \in \mathbb{Z}_q$, then the triple $(P_{m_j}, H_p(P_{m_j}), I'_j)$ is not a DDH triple for any $m_j \in [o_j]$. Hence the collection $(\mathscr{P}_{\text{orig}}(I_j), \mathbf{H}_p(\mathscr{P}_{\text{orig}}(I_j)), I_j)$ contains a single DDH triple for a secret combination $(P_{m_j}, H_p(m_j), I_j)$ for all $I_j \in \mathbf{I}_c$. So there are N such DDH triples in $(\mathscr{P}_{\text{orig}}, \mathbf{H}_p(\mathscr{P}_{\text{orig}}), \mathbf{I}_c)$. However, with an overwhelming probability there is no such DDH triple in $(\mathscr{P}_{\text{orig}}, \mathbf{H}_p(\mathscr{P}_{\text{orig}}), \mathbf{I}'_c)$. Suppose for a PPT distinguisher $\mathcal{D}_{\mathsf{MPP}}$, there exists a polynomial $p(\lambda)$ such that,

$$\Big| \Pr[\mathcal{D}_{\mathsf{MPP}}(\mathscr{P}_{\text{orig}}, \mathbf{H}_p(\mathscr{P}_{\text{orig}}), \mathbf{I}_c) = 1] - \Pr[\mathcal{D}_{\mathsf{MPP}}(\mathscr{P}_{\text{orig}}, \mathbf{H}_p(\mathscr{P}_{\text{orig}}), \mathbf{I}'_c) = 1] \Big| \geq \frac{1}{p(\lambda)}. \tag{B.20}$$

So $\mathcal{D}_{\mathsf{MPP}}$ can distinguish between the two above scenarios with a non-negligible probability. We will show how to construct a DDH adversary $\mathcal{D}_{\mathsf{DDH}}$ using $\mathcal{D}_{\mathsf{MPP}}$ as a subroutine. A DDH challenger $\mathcal{C}$ samples $b \xleftarrow{\$} \{0,1\}, x, y, z \xleftarrow{\$} \mathbb{Z}_q$. $\mathcal{C}$ sets $X = G^x, Y = G^y, Z = G^{z_b}$, where $z_0 = z, z_1 = xy$. $\mathcal{C}$ sends (X, Y, Z) to $\mathcal{D}_{\mathsf{DDH}}$. $\mathcal{D}_{\mathsf{DDH}}$ outputs b' as the estimate of b. $\mathcal{D}_{\mathsf{DDH}}$ wins if $b' = b$. We construct $\mathcal{D}_{\mathsf{DDH}}$ using a hybrid argument.

[‡]Some of the vertices containing one-time addresses and edges of the bipartite graphs are removed while constructing the originating sets. The removed vertices and edges are the same for both the cases. Hence we can ignore them as well.

Consider a hybrid simulator $\mathcal{S}_{\text{hyb}}^{(n)}$ which is given $(\mathscr{P}_{\text{orig}}, \mathbf{H}_p(\mathscr{P}_{\text{orig}}), \mathbf{I}_c)$, $n \in \{0, 1, 2, \ldots, N\}$. $\mathcal{S}_{\text{hyb}}^{(n)}$ works as follows.

1. It keeps the first n elements of $\mathbf{I}_c$ as it is.
2. It sets the $n+1$ to N elements of $\mathbf{I}_c$ as uniform elements of $\mathbb{G}$. Let the modified $\mathbf{I}_c$ be $\mathbf{I}_c^{(n)}$.
3. It outputs $(\mathscr{P}_{\text{orig}}, \mathbf{H}_p(\mathscr{P}_{\text{orig}}), \mathbf{I}_c^{(n)})$.

By observation we have,

$$\Pr[\mathcal{D}_{\mathsf{MPP}}(\mathscr{P}_{\text{orig}}, \mathbf{H}_p(\mathscr{P}_{\text{orig}}), \mathbf{I}_c) = 1] = \Pr[\mathcal{D}_{\mathsf{MPP}}(\mathcal{S}_{\text{hyb}}^{(N)}) = 1], \tag{B.21}$$

$$\Pr[\mathcal{D}_{\mathsf{MPP}}(\mathscr{P}_{\text{orig}}, \mathbf{H}_p(\mathscr{P}_{\text{orig}}), \mathbf{I}'_c) = 1] = \Pr[\mathcal{D}_{\mathsf{MPP}}(\mathcal{S}_{\text{hyb}}^{(0)}) = 1]. \tag{B.22}$$

The construction of $\mathcal{D}_{\mathsf{DDH}}$ having (X, Y, Z) as input is as follows.

1. It queries $\mathcal{D}_{\mathsf{MPP}}$ and obtains N where $2 \leq N \leq \sum_{i=1}^{f(\lambda)} n_i$.
2. It randomly chooses $k^*, o_1, o_2, \ldots, o_N \xleftarrow{\$} [N]$. It then chooses $m_1 \xleftarrow{\$} [o_1], m_2 \xleftarrow{\$} [o_2], \ldots, m_{k^*} \xleftarrow{\$} [o_{k^*}]$.
3. It defines the following sets for all $j \in [N]$,

$$\mathscr{P}_{\text{orig}}(I_j) = \{P_{j,1}, P_{j,2}, \ldots, P_{j,o_j}\},$$
$$\mathbf{H}_p(\mathscr{P}_{\text{orig}}(I_j)) = \{Q_{j,1}, Q_{j,2}, \ldots, Q_{j,o_j}\}.$$

4. It chooses $x_1, y_1, x_2, y_2, \ldots, x_{k^*-1}, y_{k^*-1} \xleftarrow{\$} \mathbb{Z}_q$. For all $j \in [k^* - 1]$, it sets,

$$P_{j,m_j} = G^{x_j}, \quad Q_{j,m_j} = G^{y_j}, \quad I_j = G^{x_j y_j}.$$

   It also sets,

$$P_{k^*,m_{k^*}} = X, \quad Q_{k^*,m_{k^*}} = Y, \quad I_{k^*} = Z.$$

5. It sets $I_{k^*+1}, I_{k^*+2}, \ldots, I_N$ as uniform elements from $\mathbb{G}$. It populates other elements of $(\mathscr{P}_{\text{orig}}(I_j), \mathbf{H}_p(\mathscr{P}_{\text{orig}}(I_j)))$ with uniform elements from $\mathbb{G}$ for all $j \in [N]$.

6. It defines the following sets,

$$\mathbf{I}_{\mathrm{DDH}} = \{I_1, \ldots, I_N\}$$
$$\mathscr{P}_{\mathrm{orig,DDH}} = \{\mathscr{P}_{\mathrm{orig}}(I_1), \ldots, \mathscr{P}_{\mathrm{orig}}(I_N)\}$$
$$\mathbf{H}_p(\mathscr{P}_{\mathrm{orig,DDH}}) = \{\mathbf{H}_p(\mathscr{P}_{\mathrm{orig}}(I_1)), \ldots, \mathbf{H}_p(\mathscr{P}_{\mathrm{orig}}(I_N))\}.$$

7. It sends $(\mathscr{P}_{\mathrm{orig,DDH}}, \mathbf{H}_p(\mathscr{P}_{\mathrm{orig,DDH}}), \mathbf{I}_{\mathrm{DDH}})$ to $\mathscr{D}_{\mathsf{MPP}}$ and receives b' as output. It sends b' as its response to the challenger $\mathcal{C}$.

Now we have,

$$\begin{aligned}
&\Pr\left[\mathscr{D}_{\mathsf{DDH}}(X,Y,Z) = 1 \;\middle|\; b = 0\right] = \sum_{l=1}^{N} \Pr[k^* = l] \Pr\left[\mathscr{D}_{\mathsf{DDH}}(X,Y,Z) = 1 \;\middle|\; b = 0 \wedge k^* = l\right], \\
&= \sum_{l=1}^{N} \Pr[k^* = l] \Pr\left[\mathscr{D}_{\mathsf{MPP}}(\mathscr{P}_{\mathrm{orig,DDH}}, \mathbf{H}_p(\mathscr{P}_{\mathrm{orig,DDH}}), \mathbf{I}_{\mathrm{DDH}}) = 1 \;\middle|\; b = 0 \wedge k^* = l\right], \\
&\overset{(1)}{=} \sum_{l=1}^{N} \frac{1}{N} \Pr\left[\mathscr{D}_{\mathsf{MPP}}(\mathcal{S}_{\mathrm{hyb}}^{(l-1)}) = 1\right], \\
&\overset{(2)}{=} \sum_{l=0}^{N-1} \frac{1}{N} \Pr\left[\mathscr{D}_{\mathsf{MPP}}(\mathcal{S}_{\mathrm{hyb}}^{(l)}) = 1\right]. \qquad \text{(B.23)}
\end{aligned}$$

Here equality (1) comes from the fact that when $b = 0$, $(P_{l,m_l}, Q_{l,m_l}, I_l)$ is not a DDH triple. Equality (2) is obtained by simple changes in the indices of the summation. Similarly we obtain the following equation,

$$\begin{aligned}
&\Pr\left[\mathscr{D}_{\mathsf{DDH}}(X,Y,Z) = 1 \;\middle|\; b = 1\right] = \sum_{l=1}^{N} \Pr[k^* = l] \Pr\left[\mathscr{D}_{\mathsf{DDH}}(X,Y,Z) = 1 \;\middle|\; b = 1 \wedge k^* = l\right], \\
&= \sum_{l=1}^{N} \Pr[k^* = l] \Pr\left[\mathscr{D}_{\mathsf{MPP}}(\mathscr{P}_{\mathrm{orig,DDH}}, \mathbf{H}_p(\mathscr{P}_{\mathrm{orig,DDH}}), \mathbf{I}_{\mathrm{DDH}}) = 1 \;\middle|\; b = 1 \wedge k^* = l\right], \\
&= \sum_{l=1}^{N} \frac{1}{N} \Pr\left[\mathscr{D}_{\mathsf{MPP}}(\mathcal{S}_{\mathrm{hyb}}^{(l)}) = 1\right]. \qquad \text{(B.24)}
\end{aligned}$$

We have,

$$
\begin{aligned}
&\Big|\Pr\left[\mathscr{D}_{\mathsf{DDH}}(X,Y,Z)=1 \;\middle|\; b=0\right]-\Pr\left[\mathscr{D}_{\mathsf{DDH}}(X,Y,Z)=1 \;\middle|\; b=1\right]\Big| \\
&\overset{(3)}{=}\Big|\frac{1}{N}\left(\sum_{l=0}^{N-1}\Pr\left[\mathscr{D}_{\mathsf{MPP}}(\mathcal{S}_{\text{hyb}}^{(l)})=1\right]-\sum_{l=1}^{N}\Pr\left[\mathscr{D}_{\mathsf{MPP}}(\mathcal{S}_{\text{hyb}}^{(l)})=1\right]\right)\Big|, \\
&\overset{(4)}{=}\frac{1}{N}\Big|\Pr\left[\mathscr{D}_{\mathsf{MPP}}(\mathcal{S}_{\text{hyb}}^{(0)})=1\right]-\Pr\left[\mathscr{D}_{\mathsf{MPP}}(\mathcal{S}_{\text{hyb}}^{(N)})=1\right]\Big|, \\
&\overset{(5)}{=}\frac{1}{N}\Big|\Pr[\mathscr{D}_{\mathsf{MPP}}(\mathscr{P}_{\text{orig}},\mathbf{H}_p(\mathscr{P}_{\text{orig}}),\mathbf{I}'_c)=1]-\Pr[\mathscr{D}_{\mathsf{MPP}}(\mathscr{P}_{\text{orig}},\mathbf{H}_p(\mathscr{P}_{\text{orig}}),\mathbf{I}_c)=1]\Big|, \\
&\overset{(6)}{\geq}\frac{1}{N}p(\lambda), \\
&=p_1(\lambda)\text{ (say)}.
\end{aligned}
\tag{B.25}
$$

Here the equality (3) comes from equations (B.23) and (B.24), the equality (4) comes from cancellations, the equality (5) comes from equations (B.22) and (B.21), and the inequality (6) comes from the assumption given in the inequality (B.20). However, the inequality (B.25) is a contradiction under the DDH assumption. So the inequality (B.20) cannot be true for any polynomial $p(\lambda)$. Hence there exists a negligible function $\mathsf{negl}'(\lambda)$ such that the inequality (B.16) holds. □

Appendix C

Proof of Theorem 4.1

We need to prove that no PPT distinguisher $\mathcal{D}_{\mathsf{Rev}}$ in the `RevPriv` experiment can distinguish between $\mathsf{Rev}_0 = (\mathbf{C}^{(i)}_{\text{anon}}, \hat{\mathbf{I}}^{(i)}, \hat{\mathbf{\Sigma}}^{(i)})_{i=1}^{f(\lambda)}$ and $\mathsf{Rev}_1 = (\mathbf{C}^{(i)}_{\text{anon}}, \mathbf{I}^{(i)}, \mathbf{\Sigma}^{(i)})_{i=1}^{f(\lambda)}$ with a probability non-negligibly better than $\frac{1}{2}$, i.e. inequality (4.25) holds for every PPT distinguisher $\mathcal{D}_{\mathsf{Rev}}$. First consider $(\mathbf{\Sigma}^{(i)})_{i=1}^{f(\lambda)}$ in Rev_1 and $(\hat{\mathbf{\Sigma}}^{(i)})_{i=1}^{f(\lambda)}$ in Rev_0. The elements in these two NIZKPoK signatures are identically distributed and give $\mathcal{D}_{\mathsf{Rev}}$ no advantage in predicting b. Next, note that there might be many outputs in $(\mathbf{C}^{(i)}_{\text{anon}})_{i=1}^{f(\lambda)}$ which appear across multiple proofs. If they are owned by the exchange, then the corresponding key images in $(\mathbf{I}^{(i)})_{i=1}^{f(\lambda)}$ will be the same. If they are not owned by the exchange, then also the corresponding key images will be the same in $(\mathbf{I}^{(i)})_{i=1}^{f(\lambda)}$ because of using the long term secret key k_{exch}. Whenever an output repeats in $(\mathbf{C}^{(i)}_{\text{anon}})_{i=1}^{f(\lambda)}$, the same simulated key image appears in $(\hat{\mathbf{I}}^{(i)})_{i=1}^{f(\lambda)}$ because of the long term key k_{sim} used by the simulator $\mathcal{S}_{\mathsf{Rev}}$. After discarding the duplicate elements in $(\mathbf{C}^{(i)}_{\text{anon}}, \mathbf{I}^{(i)}, \hat{\mathbf{I}}^{(i)})_{i=1}^{f(\lambda)}$, we obtain the following vectors.

$$\mathbf{C} = (C_1, C_2, \ldots, C_N), \tag{C.1}$$

$$\mathbf{I} = (I_1, I_2, \ldots, I_N), \tag{C.2}$$

$$\hat{\mathbf{I}} = (\hat{I}_1, \hat{I}_2, \ldots, \hat{I}_N), \tag{C.3}$$

where N is the number of distinct outputs in $(\mathbf{C}^{(i)}_{\text{anon}})_{i=1}^{f(\lambda)}$. Note that $N \leq \sum_{i=1}^{f(\lambda)} n_i$ where n_i is the size of the ith anonymity set. Let $\mathbf{v} = (v_1, v_2, \ldots, v_N)$ denote the amounts corresponding to the outputs in $\mathbf{C}$. Also let $C_{j_1}, C_{j_2}, \ldots, C_{j_s} \in \mathbf{C}$ be s exchange-owned outputs in $\mathbf{C}$. We define a vector $\mathbf{vH} = (v_1 H, v_2 H, \ldots, v_N H)$ and $\mathbf{G'} = (\underbrace{G', G' \ldots, G'}_{N \text{ times}})$. We observe that there are s DDH triples in the tuple $(\mathbf{C} - \mathbf{vH}, \mathbf{G'}, \mathbf{I} - \mathbf{vH})$. In particular, consider the j_lth elements in $\mathbf{C}$ and $\mathbf{I}$, $l \in [s]$. They are respectively $C_{j_l} = k_{j_l} G + v_{j_l} H$

and $I_{j_l} = k_{j_l}G' + v_{j_l}H$, as C_{j_l} is an exchange-owned output. Hence the j_lth elements of $\mathbf{C} - \mathbf{vH}$ and $\mathbf{I} - \mathbf{vH}$ are $k_{j_l}G$ and $k_{j_l}G'$ respectively. If $G' = k'G$ for some $k' \in \mathbb{Z}_n$, then we have the following DDH triples in the tuple $(\mathbf{C} - \mathbf{vH}, \mathbf{G}', \mathbf{I} - \mathbf{vH})$.

$$\begin{aligned}(C_{j_1} - v_{j_1}H, G', I_{j_1} - v_{j_1}H) &= (k_{j_1}G, k'G, k_{j_1}k'G)\\ (C_{j_2} - v_{j_2}H, G', I_{j_2} - v_{j_2}H) &= (k_{j_2}G, k'G, k_{j_2}k'G),\\ &\vdots\\ (C_{j_s} - v_{j_s}H, G', I_{j_s} - v_{j_s}H) &= (k_{j_s}G, k'G, k_{j_s}k'G).\end{aligned}$$

However in the other locations in $(\mathbf{C} - \mathbf{vH}, \mathbf{G}', \mathbf{I} - \mathbf{vH})$, there are no DDH triples (except with a negligible probability). To see this, consider an index $j \in [N], j \notin \{j_1, j_2, \ldots, j_s\}$. As C_j is not an exchange-owned output, $I_j = y_jG'$ for some $y_j \in \mathbb{Z}_n$. Hence $I_j - v_jH = y_jG' - v_jH = k'_jG$ for some k'_j which is unlikely to be equal to $k'k_j$. As $\hat{I}_j = q_jG'$ for all $j \in [N]$, there are no DDH triples in the tuple $(\mathbf{C} - \mathbf{vH}, \mathbf{G}', \hat{\mathbf{I}} - \mathbf{vH})$ (except with a negligible probability).

Now suppose in the `RevPriv` experiment, $\mathscr{D}_{\mathsf{Rev}}$ is replaced by $\mathfrak{D}_{\mathsf{Rev}}$ which receives $(\mathbf{C}, \mathbf{v}, \mathbf{G}', \mathbf{I}_b, j_1, \ldots, j_s)$ instead of Rev_b. Here $\mathbf{I}_0$ is $\hat{\mathbf{I}}$ and $\mathbf{I}_1$ is $\mathbf{I}$. Notice, now $\mathfrak{D}_{\mathsf{Rev}}$ has the locations of the exchange-owned outputs and the amounts of all the outputs as additional and relevant information compared to $\mathscr{D}_{\mathsf{Rev}}$ when predicting the value of b is of concern. The additional information $((\mathbf{\Sigma}^{(i)})_{i=1}^{f(\lambda)}$, duplicate values in $\mathbf{C}, \mathbf{I}, \hat{\mathbf{I}})$ that $\mathscr{D}_{\mathsf{Rev}}$ has compared to $\mathfrak{D}_{\mathsf{Rev}}$ does not help $\mathscr{D}_{\mathsf{Rev}}$ in predicting the value of b as discussed above. Hence for every PPT adversary $\mathscr{D}_{\mathsf{Rev}}$, there exists another PPT adversary $\mathfrak{D}_{\mathsf{Rev}}$ such that,

$$\left| \Pr[\mathscr{D}_{\mathsf{Rev}}(\mathsf{Rev}_b) = b] - \frac{1}{2} \right| \leq \left| \Pr[\mathfrak{D}_{\mathsf{Rev}}(\mathbf{C}, \mathbf{v}, \mathbf{G}', \mathbf{I}_b, j_1, \ldots, j_s) = b] - \frac{1}{2} \right|. \tag{C.4}$$

We define the following notation.

$$\widetilde{\mathsf{Rev}}_b = \mathbf{C}, \mathbf{v}, \mathbf{G}', \mathbf{I}_b, j_1, \ldots, j_s, \tag{C.5}$$

$$\widetilde{\mathsf{Rev}}_0 = \mathbf{C}, \mathbf{v}, \mathbf{G}', \mathbf{I}_0, j_1, \ldots, j_s, \tag{C.6}$$

$$\widetilde{\mathsf{Rev}}_1 = \mathbf{C}, \mathbf{v}, \mathbf{G}', \mathbf{I}_1, j_1, \ldots, j_s. \tag{C.7}$$

The RHS of the inequality (C.4) can be alternatively represented as,

$$\begin{aligned}
&\Big| \Pr[\mathfrak{D}_{\mathsf{Rev}}(\widetilde{\mathsf{Rev}}_b) = b] - \frac{1}{2} \Big| \\
=&\Big| \Pr[b=0] \Pr[\mathfrak{D}_{\mathsf{Rev}}(\widetilde{\mathsf{Rev}}_b) = b | b=0] + \Pr[b=1] \Pr[\mathfrak{D}_{\mathsf{Rev}}(\widetilde{\mathsf{Rev}}_b) = b | b=1] - \frac{1}{2} \Big|, \\
=&\Big| \frac{1}{2} \Pr[\mathfrak{D}_{\mathsf{Rev}}(\widetilde{\mathsf{Rev}}_0) = 0] + \frac{1}{2} \Pr[\mathfrak{D}_{\mathsf{Rev}}(\widetilde{\mathsf{Rev}}_1) = 1] - \frac{1}{2} \Big|, \\
=&\Big| \frac{1}{2} \Big(1 - \Pr[\mathfrak{D}_{\mathsf{Rev}}(\widetilde{\mathsf{Rev}}_0) = 1]\Big) + \frac{1}{2} \Pr[\mathfrak{D}_{\mathsf{Rev}}(\widetilde{\mathsf{Rev}}_1) = 1] - \frac{1}{2} \Big|, \\
=& \frac{1}{2} \Big| \Pr[\mathfrak{D}_{\mathsf{Rev}}(\widetilde{\mathsf{Rev}}_0) = 1] - \Pr[\mathfrak{D}_{\mathsf{Rev}}(\widetilde{\mathsf{Rev}}_1) = 1] \Big|, \\
=& \frac{1}{2} \Big| \Pr[\mathfrak{D}_{\mathsf{Rev}}(\mathbf{C}, \mathbf{v}, \mathbf{G}', \mathbf{I}_0, j_1, \ldots, j_s) = 1] - \Pr[\mathfrak{D}_{\mathsf{Rev}}(\mathbf{C}, \mathbf{v}, \mathbf{G}', \mathbf{I}_1, j_1, \ldots, j_s) = 1] \Big|.
\end{aligned} \tag{C.8}$$

From equation (C.8) and inequality (C.4) to prove Theorem 4.1, it is enough to prove the following claim.

Claim 6. *For every PPT $\mathfrak{D}_{\mathsf{Rev}}$ in the* `RevPriv` *experiment, there exists a negligible function $\mathsf{negl}_1(\lambda)$ of the security parameter λ such that,*

$$\Big| \Pr[\mathfrak{D}_{\mathsf{Rev}}(\mathbf{C}, \mathbf{v}, \mathbf{G}', \mathbf{I}_0, j_1, \ldots, j_s) = 1] - \Pr[\mathfrak{D}_{\mathsf{Rev}}(\mathbf{C}, \mathbf{v}, \mathbf{G}', \mathbf{I}_1, j_1, \ldots, j_s) = 1] \Big| \leq \mathsf{negl}_1(\lambda). \tag{C.9}$$

We prove Claim 6 by contradiction. Suppose for a PPT distinguisher $\mathfrak{D}_{\mathsf{Rev}}$, there exists a polynomial $p(\lambda)$ such that,

$$\Big| \Pr[\mathfrak{D}_{\mathsf{Rev}}(\mathbf{C}, \mathbf{v}, \mathbf{G}', \mathbf{I}_0, j_1, \ldots, j_s) = 1] - \Pr[\mathfrak{D}_{\mathsf{Rev}}(\mathbf{C}, \mathbf{v}, \mathbf{G}', \mathbf{I}_1, j_1, \ldots, j_s) = 1] \Big| \geq \frac{1}{p(\lambda)}. \tag{C.10}$$

So $\mathfrak{D}_{\mathsf{Rev}}$ can distinguish between the two above scenarios with a non-negligible probability. We will show how to construct a DDH adversary $\mathcal{D}_{\mathsf{DDH}}$ using $\mathfrak{D}_{\mathsf{Rev}}$ as a subroutine. A DDH challenger $\mathcal{C}$ samples $d \xleftarrow{\$} \{0,1\}, x, y, z \xleftarrow{\$} \mathbb{Z}_n$. $\mathcal{C}$ sets $X = xG, Y = yG, Z = z_dG$, where $z_0 = z, z_1 = xy$. $\mathcal{C}$ sends (X, Y, Z) to $\mathcal{D}_{\mathsf{DDH}}$. $\mathcal{D}_{\mathsf{DDH}}$ outputs d' as the estimate of d. $\mathcal{D}_{\mathsf{DDH}}$ wins if $d' = d$. We construct $\mathcal{D}_{\mathsf{DDH}}$ using a hybrid argument.

Consider a hybrid simulator $\mathcal{S}^{(n)}_{\text{hyb}}$ which is given $(\mathbf{C}, \mathbf{v}, \mathbf{G}', \mathbf{I}, \hat{\mathbf{I}}, j_1, \ldots, j_s)$ as input where $n \in \{0, 1, 2, \ldots, s\}$. $\mathcal{S}^{(n)}_{\text{hyb}}$ works as follows.

1. It produces a vector $\mathbf{I}^{(n)}$ of same length of $\mathbf{I}$ and $\hat{\mathbf{I}}$, i.e. N. It populates the j_1th, j_2th, $\ldots$, j_nth elements of $\mathbf{I}^{(n)}$ with the j_1th, j_2th, $\ldots$, j_nth elements of $\mathbf{I}$.

2. It populates the other elements of $\mathbf{I}^{(n)}$ with uniform elements from $\mathbb{G}$.

3. It outputs $(\mathbf{C}, \mathbf{v}, \mathbf{G}', \mathbf{I}^{(n)}, j_1, \ldots, j_s)$.

Modeling $\mathcal{H}(\cdot)$ a random oracle we have,

$$\Pr[\mathfrak{D}_{\mathsf{Rev}}((\mathbf{C}, \mathbf{v}, \mathbf{G}', \mathbf{I}_0, j_1, \ldots, j_s) = 1] = \Pr[\mathfrak{D}_{\mathsf{Rev}}(\mathcal{S}_{\text{hyb}}^{(0)}) = 1], \tag{C.11}$$

$$\Pr[\mathfrak{D}_{\mathsf{Rev}}(\mathbf{C}, \mathbf{v}, \mathbf{G}', \mathbf{I}_1, j_1, \ldots, j_s) = 1] = \Pr[\mathfrak{D}_{\mathsf{Rev}}(\mathcal{S}_{\text{hyb}}^{(s)}) = 1], \tag{C.12}$$

as $\mathbf{I}_0$ is $\hat{\mathbf{I}}$ and $\mathbf{I}_1$ is $\mathbf{I}$.

The construction of $\mathcal{D}_{\mathsf{DDH}}$ having (X, Y, Z) as input is as follows.

1. It queries $\mathfrak{D}_{\mathsf{Rev}}$ and obtains N, s, where $2 \leq N \leq f(\lambda), 1 \leq s \leq N-1$.

2. It randomly chooses $k^* \xleftarrow{\$} [s], j_1, j_2, \ldots, j_s \xleftarrow{\$} [N]$.

3. It defines the following vectors,

$$\begin{aligned}
\mathbf{C}^{\text{DDH}} &= (C_1, C_2, \ldots, C_N), \\
\mathbf{I}^{\text{DDH}} &= (I_1, I_2, \ldots, I_N), \\
\mathbf{v}^{\text{DDH}} &= (v_1, v_2, \ldots, v_N).
\end{aligned}$$

4. It populates the elements in $\mathbf{v}^{\text{DDH}}$ by uniformly sampling the set of allowed mounts.

5. It chooses $x_1, x_2, \ldots, x_{k^*-1} \xleftarrow{\$} \mathbb{Z}_n$. It sets $G' = Y$. For all $l \in [k^* - 1]$, it sets,

$$C_{j_l} = x_l G + v_{j_l} H, \quad I_{j_l} = x_l G' + v_{j_l} H.$$

It also sets,

$$C_{j_{k^*}} = X, \quad I_{j_{k^*}} = Z.$$

6. It populates other elements of $\mathbf{C}^{\text{DDH}}$, $\mathbf{I}^{\text{DDH}}$ with uniform elements from $\mathbb{G}$.

7. It sends $\left(\mathbf{C}^{\text{DDH}}, \mathbf{v}^{\text{DDH}}, \mathbf{G}', \mathbf{I}^{\text{DDH}}, j_1, \ldots, j_s\right)$ to $\mathfrak{D}_{\mathsf{Rev}}$ and receives b' as output. It sends b' as its response to the challenger $\mathcal{C}$.

Now we have,

$$
\begin{aligned}
&\Pr\left[\mathscr{D}_{\mathsf{DDH}}(X,Y,Z)=1 \;\middle|\; d=0\right] \\
&=\sum_{m=1}^{s}\Pr[k^*=m]\Pr\left[\mathscr{D}_{\mathsf{DDH}}(X,Y,Z)=1 \;\middle|\; d=0\wedge k^*=m\right] \\
&=\sum_{m=1}^{s}\Pr[k^*=m]\Pr\left[\mathfrak{D}_{\mathsf{Rev}}\left(\mathbf{C}^{\mathrm{DDH}},\mathbf{v}^{\mathrm{DDH}},\mathbf{G}',\mathbf{I}^{\mathrm{DDH}},j_1,\ldots,j_s\right)=1 \;\middle|\; d=0\wedge k^*=m\right], \\
&\overset{(1)}{=}\sum_{m=1}^{s}\frac{1}{s}\Pr\left[\mathfrak{D}_{\mathsf{Rev}}(\mathcal{S}_{\mathrm{hyb}}^{(m-1)})=1\right], \\
&\overset{(2)}{=}\sum_{m=0}^{s-1}\frac{1}{s}\Pr\left[\mathfrak{D}_{\mathsf{Rev}}(\mathcal{S}_{\mathrm{hyb}}^{(m)})=1\right]. \qquad \text{(C.13)}
\end{aligned}
$$

Here equality (1) comes from the fact that when $d=0$, $(C_{j_m}-v_{j_m}H, G', I_{j_m}-v_{j_m}H)$ is not a DDH triple. Equality (2) is obtained by simple changes in the indices of the summation. Similarly we obtain the following equation,

$$
\begin{aligned}
&\Pr\left[\mathscr{D}_{\mathsf{DDH}}(X,Y,Z)=1 \;\middle|\; d=1\right] \\
&=\sum_{m=1}^{s}\Pr[k^*=m]\Pr\left[\mathfrak{D}_{\mathsf{Rev}}\left(\mathbf{C}^{\mathrm{DDH}},\mathbf{v}^{\mathrm{DDH}},\mathbf{G}',\mathbf{I}^{\mathrm{DDH}},j_1,\ldots,j_s\right)=1 \;\middle|\; d=1\wedge k^*=m\right], \\
&\overset{(3)}{=}\sum_{m=1}^{s}\frac{1}{s}\Pr\left[\mathfrak{D}_{\mathsf{Rev}}(\mathcal{S}_{\mathrm{hyb}}^{(m)})=1\right]. \qquad \text{(C.14)}
\end{aligned}
$$

Here equality (3) comes from the fact that when $d=1$, $(C_{j_m}-v_{j_m}H, G', I_{j_m}-v_{j_m}H)$ is a DDH triple. Now we have

$$
\begin{aligned}
&\left|\Pr\left[\mathscr{D}_{\mathsf{DDH}}(X,Y,Z)=1 \;\middle|\; d=0\right]-\Pr\left[\mathscr{D}_{\mathsf{DDH}}(X,Y,Z)=1 \;\middle|\; d=1\right]\right| \\
&\overset{(4)}{=}\left|\frac{1}{s}\left(\sum_{m=0}^{s-1}\Pr\left[\mathfrak{D}_{\mathsf{Rev}}(\mathcal{S}_{\mathrm{hyb}}^{(m)})=1\right]-\sum_{m=1}^{s}\Pr\left[\mathfrak{D}_{\mathsf{Rev}}(\mathcal{S}_{\mathrm{hyb}}^{(m)})=1\right]\right)\right|, \\
&\overset{(5)}{=}\frac{1}{s}\left|\Pr\left[\mathfrak{D}_{\mathsf{Rev}}(\mathcal{S}_{\mathrm{hyb}}^{(0)})=1\right]-\Pr\left[\mathfrak{D}_{\mathsf{Rev}}(\mathcal{S}_{\mathrm{hyb}}^{(s)})=1\right]\right|, \\
&\overset{(6)}{=}\frac{1}{s}\left|\Pr[\mathfrak{D}_{\mathsf{Rev}}(\mathbf{C},\mathbf{v},\mathbf{G}',\mathbf{I}_0,j_1,\ldots,j_s)=1]-\Pr[\mathfrak{D}_{\mathsf{Rev}}(\mathbf{C},\mathbf{v},\mathbf{G}',\mathbf{I}_1,j_1,\ldots,j_s)=1]\right|, \\
&\overset{(7)}{\geq}\frac{1}{s}p(\lambda)=p_1(\lambda)\ \text{(say)}. \qquad \text{(C.15)}
\end{aligned}
$$

Here the equality (4) comes from equations (C.13) and (C.14), the equality (5) comes from cancellations, the equality (6) comes from equations (C.11) and (C.12), and the inequality (7) comes from the assumption given in the inequality (C.10). However, the inequality (C.15) is a contradiction under the DDH assumption. So the inequality (C.10) cannot be true for any polynomial $p(\lambda)$. Hence Claim 6 is proved. □

Appendix D

Proofs and Signature Generation Procedures in Chapter 5

D.1 Proof of Theorem 5.1

We need to prove that no PPT distinguisher $\mathscr{D}_{\mathsf{Num}}$ in the `NumPriv` experiment can distinguish between $\mathsf{Num}_1 = \left(h_{j_i}, \mathscr{A}^{(i)}_{\text{anon}}, \mathbf{P}^{(i)}, \mathbf{\Sigma}^{(i)}\right)_{i=1}^{f(\lambda)}$ and $\mathsf{Num}_0 = \left(h_{j_i}, \mathscr{A}^{(i)}_{\text{anon}}, \hat{\mathbf{P}}^{(i)}, \hat{\mathbf{\Sigma}}^{(i)}\right)_{i=1}^{f(\lambda)}$ with a probability non-negligibly better than $\frac{1}{2}$, i.e. inequality (5.20) holds for every PPT distinguisher $\mathscr{D}_{\mathsf{Num}}$.

The anonymity sets $\left\{\mathscr{A}^{(i)}_{\text{anon}}\right\}_{i=1}^{f(\lambda)}$ are identical in Num_0 and Num_1. Also the elements in $(\mathbf{\Sigma}^{(i)})_{i=1}^{f(\lambda)}$ in Num_1 and $(\hat{\mathbf{\Sigma}}^{(i)})_{i=1}^{f(\lambda)}$ in Num_0 are identically distributed. Hence these elements do not provide any advantage to $\mathscr{D}_{\mathsf{Num}}$ in predicting b. Also consider a Pedersen commitment corresponding to an account not owned by the exchange. They are identically distributed i.e. $\hat{p}_{i,k} = h_{j_i}^{q_{i,k}}$ in Num_0 and $p_{i,k} = h_{j_i}^{w_{i,k}}$ in Num_1, where $q_{i,k}, w_{i,k} \xleftarrow{\$} \mathbb{F}_p$. However, for an exchange-owned account the Pedersen commitments are,

$$p_{i,k} = g^{v_{i,k}} h_{j_i}^{k_{i,k}} \text{ in } \mathsf{Num}_1 \text{ and,}$$
$$\hat{p_{i,k}} = h_{j_i}^{q_{i,k}} \text{ in } \mathsf{Num}_0.$$

As mentioned, in a particular anonymity set $\mathscr{A}^{(i)}_{\text{anon}}$, all the exchange-owned accounts should have distinct secret keys. However, a particular secret key k_l can appear $f_l(\lambda)$ times across $f(\lambda)$ Nummatus proofs, where $f_l(\lambda) \in [f(\lambda)]$. Suppose, $\mathsf{Num}_{\mathsf{act}}$ has N number of such distinct keys $k_1, k_2, \ldots, k_N$ used by the exchange across $f(\lambda)$ proofs. Suppose for a particular secret key k_l, the set $\{p_{l,1}, p_{l,2}, \ldots, p_{l,f_l(\lambda)}\} \in \left(\mathbf{P}^{(i)}\right)_{i=1}^{f(\lambda)}$ is the set of all Pedersen

Table D.1: Various Quantities Used in the Proof of Theorem 5.1

Notation	Meaning
$p_{i,j} \in \mathbf{P}^{(i)}$	Pedersen commitment in the jth account in the ith anonymity set.
$p_{l,j} \in \mathbf{M}^{(l)}$	Pedersen commitment in the jth exchange-owned account in the set $\bigcup_{i=1}^{f(\lambda)} \mathbf{P}^{(i)}$ where k_l is used.
$\hat{p}_{i,j} \in \mathbf{P}^{(i)}$	Simulated Pedersen commitment in the jth account in the ith anonymity set.
$\hat{p}_{l,j} \in \mathbf{M}^{(l)}$	Simulated Pedersen commitment in the jth exchange-owned account in the set $\bigcup_{i=1}^{f(\lambda)} \hat{\mathbf{P}}^{(i)}$ where k_l is used in the original proof.
h_{j_i}	Base element used in the ith anonymity set.
$h_{l,j}$	Base element used for the Pedersen commitment $p_{l,j}$.
$f_l(\lambda)$	The number of times the exchange uses the key k_l in the $f(\lambda)$ anonymity sets.
N	The total number of distinct keys the exchange has used in the $f(\lambda)$ anonymity sets.

commitments where k_l is used as a blinding factor. For each k_l, $l \in [N]$, we define the following vectors.

$$\begin{aligned} \mathbf{M}^{(l)} &= (h_{l,j}, p_{l,j}, v_{l,j})_{j=1}^{f_l(\lambda)}, \\ \hat{\mathbf{M}}^{(l)} &= (h_{l,j}, \hat{p}_{l,j}, v_{l,j})_{j=1}^{f_l(\lambda)}, \end{aligned} \tag{D.1}$$

where $v_{l,j}$ is the balance of the particular account where k_l has been used in the jth proof, $h_{l,j}$ is the corresponding base, and $\hat{p}_{l,j}$ is the corresponding simulated Pedersen commitment. Table D.1 lists the various quantities used so far.

Now we notice the following sequence in the vector $\mathbf{M}^{(l)}$,

$$\begin{aligned} &h_{l,1}, h_{l,2}, \ldots, h_{l,f_l(\lambda)}, p_{l,1}g^{-v_{l,1}}, p_{l,2}g^{-v_{l,2}}, \ldots, p_{l,f_l(\lambda)}g^{-v_{l,f_l(\lambda)}} \\ &= h_{l,1}, h_{l,2}, \ldots, h_{l,f_l(\lambda)}, h_{l,1}^{k_l}, h_{l,2}^{k_l}, \ldots, h_{l,f_l(\lambda)}^{k_l}. \end{aligned}$$

In the vector $\hat{\mathbf{M}}^{(l)}$, the corresponding sequence will be,

$$\begin{aligned} &h_{l,1}, h_{l,2}, \ldots, h_{l,f_l(\lambda)}, \hat{p}_{l,1}g^{-v_{l,1}}, \hat{p}_{l,2}g^{-v_{l,2}}, \ldots, \hat{p}_{l,f_l(\lambda)}g^{-v_{l,f_l(\lambda)}} \\ &= h_{l,1}, h_{l,2}, \ldots, h_{l,f_l(\lambda)}, h_{l,1}^{q_{l,1}}g^{-v_{l,1}}, h_{l,2}^{q_{l,2}}g^{-v_{l,2}}, \ldots, h_{l,f_l(\lambda)}^{q_{l,f_l(\lambda)}}g^{-v_{l,f_l(\lambda)}}, \\ &= h_{l,1}, h_{l,2}, \ldots, h_{l,f_l(\lambda)}, u_{l,1}, u_{l,2}, \ldots, u_{l,f_l(\lambda)}, \end{aligned}$$

where $u_{l,k}$ are uniform group elements for $k \in [f_(\lambda)]$.

Hence distinguishing between the vectors $\mathbf{M}^{(l)}$ and $\hat{\mathbf{M}}^{(l)}$ is equivalent to solving the generalized DDH problem [63] described below.

Suppose there is a generalized DDH challenger $\mathcal{C}_{\mathsf{GDDH}}$ and a PPT distinguisher $\mathcal{D}_{\mathsf{GDDH}}$. The challenger $\mathcal{C}_{\mathsf{GDDH}}$ calculates the following quantities.

$$g_1, g_2, \ldots, g_{f'(\lambda)}, u_1, u_2, \ldots, u_{f'(\lambda)} \xleftarrow{\$} \mathbb{G}, d \xleftarrow{\$} \{0,1\}, k \xleftarrow{\$} \mathbb{F}_p.$$

If $d = 0$, $\mathcal{C}_{\mathsf{GDDH}}$ sets

$$a_1 = g_1, a_2 = g_2, \ldots, a_{f'(\lambda)} = g_{f'(\lambda)}, b_{0,1} = u_1, b_{0,2} = u_2, \ldots, b_{0,f'(\lambda)} = u_{f'(\lambda)}.$$

If $d = 1$, $\mathcal{C}_{\mathsf{GDDH}}$ sets

$$a_1 = g_1, a_2 = g_2, \ldots, a_{f'(\lambda)} = g_{f'(\lambda)}, b_{1,1} = g_1^k, b_{1,2} = g_2^k, \ldots, b_{1,f'(\lambda)} = g_{f'(\lambda)}^k.$$

$\mathcal{C}_{\mathsf{GDDH}}$ sends $(a_1, a_2, \ldots, a_{f'(\lambda)}, b_{d,1}, b_{d,2}, \ldots, b_{d,f'(\lambda)})$ to $\mathcal{D}_{\mathsf{GDDH}}$ and $\mathcal{D}_{\mathsf{GDDH}}$ outputs a bit $\hat{d}$. $\mathcal{D}_{\mathsf{GDDH}}$ wins if $\hat{d} = d$.

We call the tuple sent by $\mathcal{C}_{\mathsf{GDDH}}$ when $d = 1$, a generalized DDH tuple. There are N such tuples in $(\mathbf{M}^{(l)})_{l=1}^N$. However, with an overwhelming probability there is no such tuple in $(\hat{\mathbf{M}}^{(l)})_{l=1}^N$. A PPT distinguisher $\mathcal{D}_{\mathsf{Num}}$ who can successfully predict b in the `NumPriv` experiment, must be able to distinguish between these two scenarios.

Now suppose in the `NumPriv` experiment, $\mathcal{D}_{\mathsf{Num}}$ is replaced by $\mathfrak{D}_{\mathsf{Num}}$ which receives $\mathbf{M}_b$ instead of Num_b. Here $\mathbf{M}_0$ is $(\hat{\mathbf{M}}^{(l)})_{l=1}^N$ and $\mathbf{M}_1$ is $(\mathbf{M}^{(l)})_{l=1}^N$. Notice that $\mathfrak{D}_{\mathsf{Num}}$ has the amounts of the accounts and the sets of all accounts where the same secret key has been used as additional and relevant information compared to $\mathcal{D}_{\mathsf{Num}}$ when predicting the value of b is of concern. The additional information ($(\mathbf{\Sigma}^{(i)})_{i=1}^{f(\lambda)}$, accounts in the anonymity sets, Pedersen commitments corresponding to cover accounts) that $\mathcal{D}_{\mathsf{Num}}$ has compared to $\mathfrak{D}_{\mathsf{Num}}$ does not help $\mathcal{D}_{\mathsf{Num}}$ in predicting the value of b as discussed above. Hence we make the following claim.

Claim 7. *For every PPT distinguisher $\mathcal{D}_{\mathsf{Num}}$ in the experiment `NumPriv`, there exist another PPT distinguisher $\mathfrak{D}_{\mathsf{Num}}$ such that the following inequality holds.*

$$\left| \Pr[\mathcal{D}_{\mathsf{Num}}(\mathsf{Num}_b) = b] - \frac{1}{2} \right| \leq \left| \Pr[\mathfrak{D}_{\mathsf{Num}}(\mathbf{M}_b) = b] - \frac{1}{2} \right|. \tag{D.2}$$

The RHS of the inequality (D.2) can be alternatively represented as,

$$\begin{aligned}
&\Big|\ \Pr[\mathfrak{D}_{\mathsf{Num}}(\mathbf{M}_b) = b] - \frac{1}{2}\ \Big| \\
=&\Big|\ \Pr[b=0]\Pr[\mathfrak{D}_{\mathsf{Num}}(\mathbf{M}_b) = b | b=0] + \Pr[b=1]\Pr[\mathfrak{D}_{\mathsf{Num}}(\mathbf{M}_b) = b | b=1] - \frac{1}{2}\ \Big|, \\
=&\Big|\ \frac{1}{2}\Pr[\mathfrak{D}_{\mathsf{Num}}(\mathbf{M}_0) = 0] + \frac{1}{2}\Pr[\mathfrak{D}_{\mathsf{Num}}(\mathbf{M}_1) = 1] - \frac{1}{2}\ \Big|, \\
=&\Big|\ \frac{1}{2}\left(1 - \Pr[\mathfrak{D}_{\mathsf{Num}}(\mathbf{M}_0) = 1]\right) + \frac{1}{2}\Pr[\mathfrak{D}_{\mathsf{Num}}(\mathbf{M}_1) = 1] - \frac{1}{2}\ \Big|, \\
=&\ \frac{1}{2}\ \Big|\ \Pr[\mathfrak{D}_{\mathsf{Num}}(\mathbf{M}_0) = 1] - \Pr[\mathfrak{D}_{\mathsf{Num}}(\mathbf{M}_1) = 1]\ \Big|,
\end{aligned} \tag{D.3}$$

From equation (D.3) and inequality (D.2), to prove Theorem 5.1, it is enough to prove the following claim.

Claim 8. *For every PPT $\mathfrak{D}_{Num}$ in the* ***NumPriv*** *experiment, there exists a negligible function $\mathsf{negl}_1(\lambda)$ of the security parameter λ such that,*

$$\Big|\ \Pr[\mathfrak{D}_{Num}(\mathbf{M}_0) = 1] - \Pr[\mathfrak{D}_{Num}(\mathbf{M}_1) = 1]\ \Big| \leq \mathsf{negl}_1(\lambda). \tag{D.4}$$

We prove Claim 8 by contradiction. Suppose for a PPT distinguisher $\mathfrak{D}_{\mathsf{Num}}$, there exists a polynomial $p(\lambda)$ such that,

$$\Big|\ \Pr[\mathfrak{D}_{\mathsf{Num}}(\mathbf{M}_0) = 1] - \Pr[\mathfrak{D}_{\mathsf{Num}}(\mathbf{M}_1) = 1]\ \Big| \geq \frac{1}{p(\lambda)}. \tag{D.5}$$

Now we construct $\mathscr{D}_{\mathsf{GDDH}}$ using $\mathfrak{D}_{\mathsf{Num}}$ as subroutine and a hybrid argument similar to previous chapters.

Consider a hybrid simulator $\mathcal{S}_{\text{hyb}}^{(n)}$ which is given $\mathbf{M}_0, \mathbf{M}_1$, $n \in \{0, 1, 2, \ldots, N\}$ as input. $\mathcal{S}_{\text{hyb}}^{(n)}$ works as follows.

1. It produces a vector $\mathbf{M}^{(n)}$ of same length of $\mathbf{M}_0$ and $\mathbf{M}_1$ i.e. N. It populates the first n elements of $\mathbf{M}^{(n)}$ with the first n elements of $\mathbf{M}_1$.

2. It populates the $(n+1)$th element to the Nth element of $\mathbf{M}^{(n)}$ with the $(n+1)$th element to the Nth element of $\mathbf{M}_0$.

3. It outputs $\mathbf{M}^{(n)}$.

We have,

$$\Pr[\mathfrak{D}_{\mathsf{Num}}((\mathbf{M}_0) = 1] = \Pr[\mathfrak{D}_{\mathsf{Num}}(\mathcal{S}_{\text{hyb}}^{(0)}) = 1], \tag{D.6}$$

$$\Pr[\mathfrak{D}_{\mathsf{Num}}(\mathbf{M}_1) = 1] = \Pr[\mathfrak{D}_{\mathsf{Num}}(\mathcal{S}_{\text{hyb}}^{(N)}) = 1]. \tag{D.7}$$

The construction of $\mathscr{D}_{\mathsf{GDDH}}$ having $(a_1, a_2, \ldots, a_{f'(\lambda)}, b_{d,1}, b_{d,2}, \ldots, b_{d,f'(\lambda)})$ as input is as follows.

1. It queries $\mathfrak{D}_{\mathsf{Num}}$ and obtains N, where $2 \leq N \leq f(\lambda)$.
2. It randomly chooses $s^* \xleftarrow{\$} [N], f_1(\lambda), f_2(\lambda), \ldots, f_{s^*-1}(\lambda), f_{s^*+1}(\lambda), \ldots, f_N(\lambda) \xleftarrow{\$} [f(\lambda)]$, $k_1, k_2, \ldots, k_{s^*-1} \xleftarrow{\$} \mathbb{F}_p$. Sets $f_{s^*}(\lambda) = f'(\lambda)$.
3. For each $i \in [N]$, it defines the vector,
$$\mathbf{M}^{(i)}_{\mathrm{GDDH}} = (h'_{i,j}, p'_{i,j}, v'_{i,j})_{j=1}^{f_i(\lambda)}.$$
4. It populates $v'_{i,j}$ by sampling uniformly the set of allowed amounts V, for all $i \in [N], j \in [f_i(\lambda)]$.
5. For $i \in [1, 2, \ldots, s^* - 1]$, it populates the random oracle output $h'_{i,j}$ with uniform elements from $\mathbb{G}$. It also sets $p'_{i,j} = {h'}_{i,j}^{k_i} g^{v'_{i,j}}$ for all $j \in [f(\lambda)], i \in [1, 2, \ldots, s^* - 1]$.
6. For $i = s^*$, it sets random oracle output $h'_{s^*,j} = a_j$ and $p'_{s^*,j} = b_j g^{v'_{s^*,j}}$ for $j \in [f'(\lambda)]$.
7. For $i = s^* + 1$ to N, it populates $h'_{i,j}$, $p'_{i,j}$ by choosing uniform group elements for all $j \in [f_i(\lambda)]$.
8. It sends $\left(\mathbf{M}^{(i)}_{\mathrm{GDDH}}\right)_{i=1}^{N}$ to $\mathfrak{D}_{\mathsf{Num}}$ and receives b' as output. It sends b' as its response to the challenger $\mathcal{C}_{\mathsf{GDDH}}$.

Now we have,

$$\begin{aligned}
&\Pr\left[\mathscr{D}_{\mathsf{GDDH}}(a_1, a_2, \ldots, a_{f'(\lambda)}, b_{d,1}, b_{d,2}, \ldots, b_{d,f'(\lambda)}) = 1 \;\middle|\; d = 0\right] \\
&= \sum_{m=1}^{N} \Pr[s^* = m] \Pr\left[\mathscr{D}_{\mathsf{GDDH}}(a_1, a_2, \ldots, a_{f'(\lambda)}, b_{d,1}, b_{d,2}, \ldots, b_{d,f'(\lambda)}) = 1 \;\middle|\; d = 0 \wedge s^* = m\right], \\
&= \sum_{m=1}^{N} \Pr[s^* = m] \Pr\left[\mathfrak{D}_{\mathsf{Num}}\left((\mathbf{M}^{(i)}_{\mathrm{GDDH}})_{i=1}^{N}\right) = 1 \;\middle|\; d = 0 \wedge s^* = m\right], \\
&\overset{(1)}{=} \sum_{m=1}^{N} \frac{1}{N} \Pr\left[\mathfrak{D}_{\mathsf{Num}}(\mathcal{S}^{(m-1)}_{\mathrm{hyb}}) = 1\right], \\
&\overset{(2)}{=} \sum_{m=0}^{N-1} \frac{1}{N} \Pr\left[\mathfrak{D}_{\mathsf{Num}}(\mathcal{S}^{(m)}_{\mathrm{hyb}}) = 1\right].
\end{aligned} \tag{D.8}$$

Here equality (1) comes from the fact that when $d = 0$, $\mathbf{M}^{(m)}_{\text{GDDH}}$ does not contain any generalized DDH tuple. Equality (2) is obtained by simple changes in the indices of the summation. Similarly we obtain the following equation,

$$\begin{aligned}
&\Pr\left[\mathcal{D}_{\mathsf{GDDH}}(a_1, a_2, \ldots, a_{f'(\lambda)}, b_{d,1}, b_{d,2}, \ldots, b_{d,f'(\lambda)}) = 1 \;\middle|\; d = 1\right] \\
&= \sum_{m=1}^{N} \Pr[s^* = m] \Pr\left[\mathfrak{D}_{\mathsf{Num}}\left((\mathbf{M}^{(i)}_{\text{GDDH}})_{i=1}^{N}\right) = 1 \;\middle|\; d = 1 \wedge s^* = m\right], \\
&\overset{(3)}{=} \sum_{m=1}^{N} \frac{1}{N} \Pr\left[\mathfrak{D}_{\mathsf{Num}}(\mathcal{S}^{(m)}_{\text{hyb}}) = 1\right]. \qquad \text{(D.9)}
\end{aligned}$$

Here equality (3) comes from the fact that when $d = 1$, $\mathbf{M}^{(m)}_{\text{GDDH}}$ contains a generalized DDH tuple. Now we have

$$\begin{aligned}
&\Big| \Pr\left[\mathcal{D}_{\mathsf{GDDH}}(a_1, a_2, \ldots, a_{f'(\lambda)}, b_{d,1}, b_{d,2}, \ldots, b_{d,f'(\lambda)}) = 1 \;\middle|\; d = 0\right] - \\
&\qquad \Pr\left[\mathcal{D}_{\mathsf{GDDH}}(a_1, a_2, \ldots, a_{f'(\lambda)}, b_{d,1}, b_{d,2}, \ldots, b_{d,f'(\lambda)}) = 1 \;\middle|\; d = 1\right] \Big| \\
&\overset{(4)}{=} \Big| \frac{1}{N} \left(\sum_{m=0}^{N-1} \Pr\left[\mathfrak{D}_{\mathsf{Num}}(\mathcal{S}^{(m)}_{\text{hyb}}) = 1\right] - \sum_{m=1}^{N} \Pr\left[\mathfrak{D}_{\mathsf{Num}}(\mathcal{S}^{(m)}_{\text{hyb}}) = 1\right] \right) \Big|, \\
&\overset{(5)}{=} \frac{1}{N} \left| \Pr\left[\mathfrak{D}_{\mathsf{Num}}(\mathcal{S}^{(0)}_{\text{hyb}}) = 1\right] - \Pr\left[\mathfrak{D}_{\mathsf{Num}}(\mathcal{S}^{(N)}_{\text{hyb}}) = 1\right] \right|, \\
&\overset{(6)}{=} \frac{1}{N} \left| \Pr[\mathfrak{D}_{\mathsf{Num}}((\mathbf{M}_0) = 1] - \Pr[\mathfrak{D}_{\mathsf{Num}}((\mathbf{M}_1) = 1] \right|, \\
&\overset{(7)}{\geq} \frac{1}{N} p(\lambda) = p_1(\lambda) \text{ (say)}. \qquad \text{(D.10)}
\end{aligned}$$

Here the equality (4) comes from equations (D.8) and (D.9), the equality (5) comes from cancellations, the equality (6) comes from equations (D.6) and (D.7), and the inequality (7) comes from the assumption given in the inequality (D.5). However, the inequality (D.10) is a contradiction under the generalized DDH assumption. So the inequality (D.5) cannot be true for any polynomial $p(\lambda)$. Hence Claim 8 is proved. □

D.2 Nummatus NIZKPoK Generation and Verification Algorithms

In this section, we describe the procedure for generating and verifying the NIZKPoK σ_i that is used in Nummatus. In the notation proposed by Camenisch and Stadler [23], [24],

the NIZKPoK is of the form

$$\text{PoK}\left\{(\alpha,\beta)\;\middle|\;\left(b_i = a_i^{\alpha}\ \wedge\ p_i d_i^{-1} = \left(c_i^{-1}h_j\right)^{\alpha}\right) \vee \left(p_i = h_j^{\beta}\right)\right\}.$$

The above proof is for a disjunction of two statements. We motivate the structure of σ_i by first describing methods to prove these two statements individually. Then the method first proposed by Cramer *et al.* [64] is used to generate a proof for the disjunction of the two statements.

Let $\mathcal{H} : \{0,1\}^* \mapsto \mathbb{F}_p$ be a cryptographic hash function which is modeled as a random oracle. Let $\|$ denote the bitstring concatenation operator. For notational convenience, we write $\mathcal{H}(x,y,z)$ to denote $\mathcal{H}(x\|y\|z)$ where x, y, z are group elements represented as bitstrings.

Definition D.1. *An ordered pair $(e,s) \in \mathbb{F}_p^2$ is a NIZKPoK of the discrete logarithm of a group element p_i with respect to a base h_j if*

$$e = \mathcal{H}(h_j, p_i, h_j^s p_i^e). \tag{D.11}$$

The pair (e,s) is said to be of the form PoK$\{\beta \mid p_i = h_j^{\beta}\}$.

The proof (e,s) is generated by first choosing a scalar r uniformly from $\mathbb{F}_p$ and calculating $e = \mathcal{H}(h_j, p_i, h_j^r)$. The second element of the pair is calculated as $s = r - e\beta$ where β is the discrete logarithm of p_i with respect to h_j, which is known to the prover. It now follows that

$$e = \mathcal{H}(h_j, p_i, h_j^r) = \mathcal{H}(h_j, p_i, h_j^{s+e\beta}) = \mathcal{H}(h_j, p_i, h_j^s p_i^e). \tag{D.12}$$

The verification of the proof (e,s) simply consists of checking the equality in equation (D.11).

Definition D.2. *An ordered pair $(e,s) \in \mathbb{F}_p^2$ is a NIZKPoK of*

1. *the knowledge of the discrete logarithms of the group elements b_i with respect to base a_i,* ***and***

2. *the knowledge of discrete logarithms of the group element $p_i d_i^{-1}$ with respect to base $c_i^{-1}h_j$,* ***and***

3. *the equality of the discrete logarithm of b_i with respect to a_i and of $p_i d_i^{-1}$ with respect to $c_i^{-1} h_j$,*

if it satisfies

$$e = \mathcal{H}\left(\mathtt{stmt}_i, a_i^s b_i^e, \left(c_i^{-1} h_j\right)^s \left(p_i d_i^{-1}\right)^e\right). \tag{D.13}$$

where $\mathtt{stmt}_i = (h_j, a_i, b_i, c_i, d_i, p_i)$ is the tuple of group elements appearing in the statement being proved. The ordered pair (e, s) is said to be of the form

$$PoK\left\{\alpha \;\middle|\; b_i = a_i^\alpha \;\wedge\; p_i d_i^{-1} = \left(c_i^{-1} h_j\right)^\alpha\right\}.$$

A prover who knows α can generate the proof (e, s) as follows:

- The prover chooses scalars r uniformly from $\mathbb{F}_p$ and calculates

$$e = \mathcal{H}\left(\mathtt{stmt}_i, a_i^r, \left(c_i^{-1} h_j\right)^r\right). \tag{D.14}$$

- The second scalar in the proof is calculated as

$$s = r - e\alpha \tag{D.15}$$

It follows that

$$\begin{aligned} e &= \mathcal{H}\left(\mathtt{stmt}_i, a_i^r, \left(c_i^{-1} h_j\right)^r\right) \\ &= \mathcal{H}\left(\mathtt{stmt}_i, a_i^{s+e\alpha}, \left(c_i^{-1} h_j\right)^{s+e\alpha}\right) \\ &= \mathcal{H}\left(\mathtt{stmt}_i, a_i^s b_i^e, \left(c_i^{-1} h_j\right)^s \left(p_i d_i^{-1}\right)^e\right). \end{aligned} \tag{D.16}$$

The verification of the proof (e, s) simply consists of checking the equality in equation (D.13).

The NIZKPoK σ_i in Nummatus is a proof of the disjunction of the two statements proved above.

Definition D.3. *The tuple $\sigma_i = (e_1, e_2, s_1, s_2) \in \mathbb{F}_p^4$ is a NIZKPoK of the knowledge of the discrete logarithm of a group element p_i with respect to base h_j* ***or***

- *the knowledge of the discrete logarithm of the group element b_i with respect to base a_i,* ***and***

- *the knowledge of discrete logarithm of the group element $p_i d_i^{-1}$ with respect to base $c_i^{-1} h_j$,* ***and***

- *the equality of the discrete logarithm of b_i with respect to a_i and of $p_i d_i^{-1}$ with respect to $c_i^{-1} h_j$,*

  if it satisfies

$$e_1 + e_2 = \mathcal{H}\left(\mathtt{stmt}_i, a_i^{s_1} b_i^{e_1}, \left(c_i^{-1} h_j\right)^{s_1} \left(p_i d_i^{-1}\right)^{e_1}, h_j^{s_2} p_i^{e_2}\right). \tag{D.17}$$

  where $\mathtt{stmt}_i = (h_j, a_i, b_i, c_i, d_i, p_i)$ is the tuple of group elements appearing in the statement being proved. The tuple (e_1, e_2, s_1, s_2) is said to be of the form

$$PoK\left\{(\alpha, \beta) \;\middle|\; \left(b_i = a_i^{\alpha} \;\wedge\; p_i d_i^{-1} = \left(c_i^{-1} h_j\right)^{\alpha}\right) \vee \left(p_i = h_j^{\beta}\right)\right\}.$$

Suppose the prover know the discrete logarithm β of p_i with respect to base h_j. Then she can create the proof σ_i as follows:

1. She chooses scalars r_2, e_1, s_1 uniformly and independently from $\mathbb{F}_p$. She calculates e_2 as

$$e_2 = \mathcal{H}\left(\mathtt{stmt}_i, a_i^{s_1} b_i^{e_1}, \left(c_i^{-1} h_j\right)^{s_1} \left(p_i d_i^{-1}\right)^{e_1}, h_j^{r_2}\right) - e_1. \tag{D.18}$$

2. Using her knowledge of β, she calculates s_2 as

$$s_2 = r_2 - e_2 \beta. \tag{D.19}$$

It follows that

$$\begin{aligned} e_1 + e_2 &= \mathcal{H}\left(\mathtt{stmt}_i, a_i^{s_1} b_i^{e_1}, \left(c_i^{-1} h_j\right)^{s_1} \left(p_i d_i^{-1}\right)^{e_1}, h_j^{s_2 + e_2 \beta}\right) \\ &= \mathcal{H}\left(\mathtt{stmt}_i, a_i^{s_1} b_i^{e_1}, \left(c_i^{-1} h_j\right)^{s_1} \left(p_i d_i^{-1}\right)^{e_1}, h_j^{s_2} p_i^{e_2}\right). \end{aligned} \tag{D.20}$$

On the other hand, if the prover knows α such that $b_i = a_i^{\alpha}$, and $p_i d_i^{-1} = \left(c_i^{-1} h_j\right)^{\alpha}$, then she can create the proof σ_i as follows:

1. She chooses scalars r_1, e_2, s_2 uniformly and independently from $\mathbb{F}_p$. She calculates e_1 as

$$e_1 = \mathcal{H}\left(\mathtt{stmt}_i, a_i^{r_1}, \left(c_i^{-1} h_j\right)^{r_1}, h_j^{s_2} p_i^{e_2}\right) - e_2. \tag{D.21}$$

2. Using her knowledge of α, she calculates s_1 as

$$s_1 = r_1 - e_1 \alpha. \tag{D.22}$$

It follows that

$$\begin{aligned} e_1 + e_2 &= \mathcal{H}\left(\mathtt{stmt}_i, a_i^{r_1}, \left(c_i^{-1}h_j\right)^{r_1}, h_j^{s_2}p_i^{e_2}\right) \\ &= \mathcal{H}\left(\mathtt{stmt}_i, a_i^{s_1+e_1\alpha}, \left(c_i^{-1}h_j\right)^{s_1+e_1\alpha}, h_j^{s_2}p_i^{e_2}\right) \\ &= \mathcal{H}\left(\mathtt{stmt}_i, a_i^{s_1}b_i^{e_1}, \left(c_i^{-1}h_j\right)^{s_1}\left(p_i d_i^{-1}\right)^{e_1}, h_j^{s_2}p_i^{e_2}\right). \end{aligned} \tag{D.23}$$

In both cases, the verification of the proof (e_1, e_2, s_1, s_2) consists of checking the equality in equation (D.17). In the proof of disjunction of statements, the prover has one degree of freedom as only the sum $e_1 + e_2$ has to be equal to the hash function output (whose argument contains the scalars). This freedom is exploited to choose which knowledge is used to prove the disjunction.

D.3 Simplus NIZKPoK Generation and Verification Algorithms

Compared to the NIZKPoK in the Nummatus protocol, the NIZKPoK ψ_i in the Simplus protocol is simpler to compute.

Definition D.4. *An ordered pair $\psi_i = (e, s) \in \mathbb{F}_p^2$ is a NIZKPoK of*

1. *the knowledge of the discrete logarithm of the group element b_i with respect to base a_i,* ***and***
2. *the knowledge of discrete logarithm of the group element $p_i d_i^{-1}$ with respect to base $c_i^{-1}h$,* ***and***
3. *the equality of the discrete logarithm of b_i with respect to a_i and of $p_i d_i^{-1}$ with respect to $c_i^{-1}h$,*

if it satisfies

$$e = \mathcal{H}\left(\mathtt{stmt}_i, a_i^s b_i^e, \left(c_i^{-1}h\right)^s \left(p_i d_i^{-1}\right)^e\right). \tag{D.24}$$

where $\mathtt{stmt}_i = (h, a_i, b_i, c_i, d_i, p_i)$ is the tuple of group elements appearing in the statement being proved. The ordered pair (e, s) is said to be of the form

$$PoK\left\{\alpha \;\middle|\; b_i = a_i^{\alpha} \;\wedge\; p_i d_i^{-1} = \left(c_i^{-1}h\right)^{\alpha}\right\}.$$

A prover who knows α can generate the proof (e, s) as follows:

- The prover chooses scalars r uniformly from $\mathbb{F}_p$ and calculates

$$e = \mathcal{H}\left(\texttt{stmt}_i, a_i^r, \left(c_i^{-1}h\right)^r\right). \tag{D.25}$$

- The second scalar in the proof is calculated as

$$s = r - e\alpha \tag{D.26}$$

It follows that

$$\begin{aligned} e &= \mathcal{H}\left(\texttt{stmt}_i, a_i^r, \left(c_i^{-1}h\right)^r\right) \\ &= \mathcal{H}\left(\texttt{stmt}_i, a_i^{s+e\alpha}, \left(c_i^{-1}h\right)^{s+e\alpha}\right) \\ &= \mathcal{H}\left(\texttt{stmt}_i, a_i^s b_i^e, \left(c_i^{-1}h\right)^s \left(p_i d_i^{-1}\right)^e\right). \end{aligned} \tag{D.27}$$

The verification of the proof (e, s) simply consists of checking the equality in equation (D.24).