MProve-Nova: A Privacy-Preserving Proof of Reserves Protocol for Monero

Varun Thakore[*]
Department of Electrical Engineering
Indian Institute of Technology Bombay
Mumbai, Maharashtra, India
varunt@ee.iitb.ac.in

Saravanan Vijayakumaran
Department of Electrical Engineering
Indian Institute of Technology Bombay
Mumbai, Maharashtra, India
sarva@ee.iitb.ac.in

Abstract

A proof of reserves (PoR) protocol enables a cryptocurrency exchange to prove to its users that it owns a certain amount of coins, as a first step towards proving that it is solvent. We present the design, implementation, and security analysis of MProve-Nova, a PoR protocol for Monero that leverages the Nova recursive SNARK to achieve two firsts (without requiring any trusted setup). It is the first Monero PoR protocol that reveals *only* the number of outputs owned by an exchange; no other information about the outputs or their key images is revealed. It is also the first Monero PoR protocol where the proof size and proof verification time are *constant*, i.e. they are *independent* of the number of outputs on the Monero blockchain and the number of outputs owned by the exchange. To achieve constant verification times, MProve-Nova requires a preprocessing step which creates two Merkle trees from all the outputs and key images on the Monero blockchain.

MProve-Nova consists of two Nova-based subprotocols, a *reserves commitment generator (RCG) protocol* used to compute a commitment to the total reserves owned by an exchange and a *non-collusion (NC) protocol* used to prove non-collusion between two exchanges. For the RCG protocol, we observed proof sizes of about 28 KB and verification times of 4.3 seconds. For the NC protocol, we observed proof sizes of about 24 KB and verification times of 0.2 seconds. Proving times for both protocols increase linearly with the number of outputs owned by the exchange but remain independent of the number of outputs on the Monero blockchain. On average, the RCG protocol required about 42 minutes per 1000 outputs and the NC protocol required about 5 minutes per 1000 outputs.

Keywords

Cryptocurrency, Monero, Proof of Reserves, Nova.

1 Introduction

Cryptocurrency exchanges provide a user-friendly platform for buying, selling, and trading of cryptocurrencies. While customers can transfer their coins from exchanges to non-custodial wallets, many of them prefer to keep their coins on exchanges to avoid the hassles and risks of managing private keys. This leaves customer funds at the risk of being stolen from the exchange due to security breaches or internal fraud.

Some examples include the bankruptcy of Mt. Gox [58] and the collapse of FTX [57]. The total funds lost due to exchange hacks alone is estimated to be at least \$2.4 billion as of April 2023 [12]. While losses due to security breaches can be avoided by hardening the protocols involving the exchange's private keys, internal fraud by exchange operators cannot be completely prevented. But such fraud can be detected early (and hence deterred) if exchanges are required to regularly publish *proofs of solvency*.

A *proof of reserves (PoR)* is one half of a proof of solvency protocol, with a *proof of liabilities (PoL)* being the other half. A PoR protocol enables an exchange to prove that it owns a certain amount of a cryptocurrency, i.e. it holds a certain amount of assets. For this reason, a PoR is sometimes also called a *proof of assets*. A PoL protocol enables an exchange to prove that the total amount of assets it is storing on behalf of all its customers (its liabilities) equals a certain amount. If an exchange's assets exceeds its liabilities, it is considered solvent.

In this paper, we focus solely on PoR protocols for Monero [37], a privacy-focused cryptocurrency based on the CryptoNote protocol [51]. We present the design, implementation, and security analysis of MProve-Nova, a PoR protocol for Monero that achieves constant proof sizes and verification times while revealing only the number of outputs owned by an exchange.

Paper organization. In Section 2, we describe the challenges involved in designing Monero PoR protocols. Our contributions are listed in Section 3. Section 4 describes related work. In Section 5, we briefly cover aspects of the Nova recursive SNARK required to describe MProve-Nova. In Section 6, we provide a high-level motivation of MProve-Nova. Sections 7 and 8 describe RCG and NC protocols. Section 9 contains the security analysis of MProve-Nova. We provide implementation details and performance results in Section 10 followed by conclusions in Section 11.

2 Challenges in Designing a Monero PoR Protocol

To describe the challenges involved in designing Monero PoR protocols, we first introduce some terminology. See Appendix A for a more detailed overview of Monero.

Let $\mathbb{G}$ be the prime-order subgroup of the ed25519 elliptic curve that contains Monero public keys and commitments.[1] The destination of coin transfers in a Monero transaction is called an *output*. Each output is characterized by a pair $(P, C) \in \mathbb{G}^2$, where P is a one-time address and C is a Pedersen commitment to the number of coins stored in the output. When we refer to the *private key* or *key image*[2] *of an output*, we mean the private key and key image corresponding to the one-time address P. Knowledge of the private key of an output implies *ownership* of the output.

[*] Current affiliation: ZkSecurity. Work done while at IIT Bombay.

[1] A list of frequently used symbols and their definitions is presented in Appendix K.

[2] Key images are one-way functions of one-time addresses that are generated as part of the Monero linkable ring signatures to prevent double spending.

Outputs containing Pedersen commitments are created in a Monero transaction type called *ring confidential transaction (RingCT)* [41]. Monero made the RingCT type of transactions mandatory in September 2017 [38].

In our design of MProve-Nova, we only consider RingCT outputs. In case an exchange owns a non-RingCT output, they can use a Pedersen commitment with zero blinding factor, i.e. a commitment of the form $C(a, 0) = aH$, to represent the output.

Many cryptocurrency protocols (including Bitcoin) have the notion of an *unspent transaction output (UTXO).* As the name suggests, this corresponds to an output having coins that have not been spent by their owner. For such cryptocurrencies, a PoR protocol can restrict its attention to the UTXO set and ignore all spent transaction outputs.

Since Monero hides the identity of the spending key in a transaction, one *cannot* partition the output set into spent and unspent outputs. While some transaction graph analysis techniques have been able to categorize a large percentage of non-RingCT outputs as spent [31, 39], it is not possible to know if any of the RingCT outputs (except for 5 of them[3]) have been spent [52, 60]. Consequently, the set of unspent outputs in Monero keeps on growing. Hence it is desirable to design PoR protocols where the proving and verification times are *independent* of the number of outputs on the Monero blockchain.

Any Monero PoR protocol must prove two statements:

(1) The prover knows the private keys corresponding to some outputs on the Monero blockchain. The sum of the coins in the output commitments will add up to the reserves owned by the prover.
(2) The outputs contributing to the prover's reserves have not been already spent in a past transaction.

While a PoR protocol could be executed by anyone, it is most useful when the prover is a cryptocurrency exchange which is trying to convince its customers that it is solvent.

We are interested in *publicly-verifiable privacy-preserving* PoR protocols. By publicly-verifiable, we mean that the PoR can be verified by any party, not just a trusted auditor. By privacy-preserving, we mean that the protocols do not reveal the specific outputs owned by the exchange.

Without the privacy requirement, it is trivial to construct a PoR protocol for Monero. The exchange can generate signatures proving ownership of a set of outputs and non-membership proofs proving that the outputs have not been spent. The latter would involve proving that the outputs' key images have not appeared in any past transaction.

Privacy is especially important in Monero PoR protocols because Monero transactions contain linkable ring signatures, with the output being spent hidden among decoy outputs sampled from the Monero blockchain. If some outputs are identified as *unspent* outputs belonging to an exchange, they can be marked as decoys in all the previous transaction rings they appear in. This reduces the effective ring size of such transactions. The implication is that a non-private Monero PoR protocol can negatively impact the privacy of other Monero users, in addition to impacting the privacy of the exchange generating the proof.

When the identities of an exchange's outputs are hidden by a PoR protocol, it opens up the possibility of *collusion* between exchanges. Collusion refers to the situation when the same output is used by two exchanges to generate their respective proofs of reserves. This is a form of double spending, where one exchange could bribe another to contribute to the former's PoR. It is desirable to have privacy-preserving PoR protocols that are also *collusion-resistant.*

Finally, it is desirable to have PoR protocols with short proofs that can be verified quickly on personal computers. This will lower the bar for proof verifiers, make it more likely that the proofs of reserves are verified by customers, and hence reduce the likelihood of exchanges in engaging in activities that could render them insolvent.

Previous Monero PoR protocols, MProve [22] and MProve+ [21], partially addressed the above challenges. Due to scalability issues, they were able to hide the exchange-owned outputs in an anonymity set that was much smaller than the set of all outputs. They proved that the exchange-owned outputs were unspent by revealing their key images. While the revealed key images also allowed detection of collusion between exchanges, revealing them negatively impacted the privacy of the exchange and the privacy of regular Monero users.

A more detailed description of MProve and MProve+ (including their drawbacks) is given in Appendices B and C.

3 Our Contributions

We describe the design, implementation, and security analysis of MProve-Nova, a publicly-verifiable privacy-preserving PoR protocol for Monero. Our design is based on Nova [30], a recursive SNARK that does not require a trusted setup. MProve-Nova consists of two subprotocols: *a reserves commitment generator (RCG) protocol* to generate a Pedersen commitment to the total amount of unspent Monero coins owned by an exchange and *a non-collusion (NC) protocol* to prove non-collusion between two exchanges. They have the following features:

(1) The RCG and NC protocols both have *constant* proof sizes and proof verification times, i.e. these two metrics are *independent* of the number of outputs on the Monero blockchain and the number of outputs owned by the exchange. This makes MProve-Nova the first Monero PoR protocol with constant proof sizes and proof verification times. However, the RCG protocol does require a preprocessing step which creates two Merkle trees from all the outputs and key images on the Monero blockchain.
(2) While the RCG protocol does not reveal any information about the exchange-owned outputs in the random oracle model, the NC protocol requires an exchange to send hashes of its private keys to another exchange. These hashes only reveal the *number of outputs* owned by the sender exchange to the receiver exchange (in the random oracle model). The MProve-Nova protocol does not reveal any other information about the exchange-owned outputs to polynomial-time adversaries. Particularly, MProve-Nova is the first Monero

[3] In 2017, Wijaya et al. [56] deliberately created a transaction [55] in Monero block 1468439 which spent from the *same* transaction ring containing 5 RingCT outputs exactly 5 times. This meant that all the outputs in this transaction ring could be marked as spent. No other cases like this have been observed [52].

PoR protocol which does not reveal the key images of the exchange-owned outputs to polynomial-time adversaries.

We formally prove that MProve-Nova proofs only reveal the number of outputs to polynomial-time adversaries in the random oracle model. We analyze the soundness of MProve-Nova and show that it is *inflation-resistant* and *collusion-resistant*. Inflation resistance means that a computationally bounded exchange cannot generate a valid RCG protocol proof that outputs a Pedersen commitment to a number of coins that is greater than the actual number of coins it owns. Collusion resistance means that a pair of computationally bounded exchanges cannot generate an valid NC protocol proof if they used the same output while generating their RCG protocol proofs at a certain block height.

We implemented MProve-Nova in Rust (our implementation is available on GitHub [47]). We present simulation results to demonstrate the feasibility of using MProve-Nova in a real-world setting.

4 Related Work

Provisions [20], proposed by Dagher *et al.*, is one of the first privacy-preserving proof of solvency protocols for Bitcoin exchanges. It consists of three subprotocols: proof of reserves, proof of liabilities and proof of solvency. In the proof of reserves, the exchange generates a Pedersen commitment [42] C_{res} to the total assets corresponding to a subset of owned addresses $\mathcal{P}_{\text{own}}$ from a larger anonymity set $\mathcal{P}$. The exchange submits a proof that it included only those amounts in C_{res} for which it knows the private keys corresponding to the addresses in $\mathcal{P}$. In the proof of liabilities, the exchange generates a Pedersen commitment to each bit of the balance amount owned by the customer. These commitments are combined to calculate the total liabilities C_{liab} of the exchange. In the proof of solvency, the exchange computes $C_{\text{diff}} = C_{\text{res}} - C_{\text{liab}}$ and proves that C_{diff} commits to a non-negative amount. There is also a fourth protocol to prove non-collusion but it reveals the number of addresses owned by the exchange and is presented as an optional protocol. Provisions is specific to Bitcoin and cannot be applied to privacy-preserving cryptocurrencies such as Monero.

In 2019, Blockstream [44, 45] released a tool to generate proof of reserves which involves generating an invalid transaction using all the UTXOs of an exchange and an invalid input so that exchange's funds are not spent. This technique does not preserve address privacy since all the UTXOs owned by the exchange are revealed.

Stoffu Noether implemented a technique for generating proof of reserves for Monero which was added to the official Monero client in 2018 [46]. It takes a target amount as input and finds the smallest set of addresses owned by the prover whose total amount exceeds the target amount. Then the set of addresses and their corresponding key images and amounts are revealed as part of the proof of reserves. Thus this technique does not provide any privacy to the prover.

Dutta *et al.* [22] proposed MProve, a proof of reserves protocol for Monero exchanges. MProve+ [21] was later proposed which enhanced the privacy of MProve by using techniques from Bulletproofs [11] and Omniring [34]. The details of MProve and MProve+ are given in Appendices B and C.

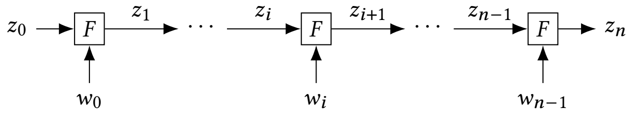

Figure 1: Incrementally Verifiable Computation

Some notable work on proof of reserves include, gOTzilla [7] proposed by Baldimtsi *et al.*, which is an interactive zero-knowledge protocol for proving disjunctive statements. Additionally, Chatzigiannis and Chalkias [16] proposed proof of assets for account-based blockchains such as Diem, formerly known as Libra [1].

There has been significant work in proof of liabilities (PoL), most notable ones being DAPOL [14] and DAPOL+ [24]. MProve-Nova can be used with DAPOL+ to provide a proof of solvency. Chalkias *et al.* [13] highlighted vulnerabilities in the implementation of PoL used in production. Chatzigiannis *et al.* [15] evaluated and systematized several distributed payment systems which offer auditability. Their work provides a comparison between different proof of assets and proof of liabilities schemes on the basis of their efficiency and privacy properties.

5 Nova

In this section, we cover aspects of the Nova recursive SNARK [30, 40] required to describe the MProve-Nova protocol. Nova introduced a non-interactive folding scheme for committed relaxed rank-1 constraint systems (R1CS) [30] (see Appendix D for a definition). It consists of two main components: an *incrementally verifiable computation (IVC)* [50] scheme and a zkSNARK to prove knowledge of valid IVC proofs.

5.1 IVC Scheme

An IVC scheme allows a prover to prove that for some function F and public values z_0 and z_n, it knows auxiliary input values $w_0, w_1, \ldots, w_{n-1}$ such that

$$z_n = F\left(F\left(\ldots F\left(F\left(F\left(z_0, w_0\right), w_1\right), w_2\right), \ldots\right), w_{n-1}\right).$$

Such a proof is generated by proving the execution of a series of incremental computations of the form $z_{i+1} = F(z_i, w_i)$, for each $i \in \{0, 1, \ldots, n-1\}$, where z_i and z_{i+1} are the public input and output in the ith step, respectively. See Figure 1.

The Nova IVC scheme uses a non-interactive folding scheme for committed relaxed R1CS. The step function F needs to be expressed using R1CS constraints. At each step i, the variables z_i, z_{i+1}, and w_i define an R1CS instance (see definition in Appendix D). This instance is folded into a running committed relaxed R1CS instance which represents the correct execution of steps $0, 1, \ldots, i-1$.

The IVC prover gives a proof Π_{i+1} at each step i, which attests that $z_{i+1} = F(z_i, w_i)$ was computed correctly and the folding of the two committed relaxed R1CS instances is valid. The IVC proof Π_{i+1} attests to the correct execution of steps $0, 1, \ldots, i$. The IVC proof generation time for n steps is $O\left(n|F|\right)$, where $|F|$ is the number of R1CS constraints needed to express the computation of the step function F.

The Nova IVC scheme satisfies completeness and knowledge-soundness. Due to space constraints, we present these definitions in Appendix E. For more details on the Nova IVC scheme, we refer the reader to Section 5 of the Nova paper [30].

5.2 zkSNARK of an IVC Proof

After n steps, the IVC prover produces a proof Π_n that attests to the correct execution of steps $0, 1, \ldots, n-1$. The IVC prover can send this proof to the verifier, but this does not satisfy zero-knowledge since the proof Π_n does not hide the prover's auxiliary inputs.

Instead, Nova uses a zero-knowledge SNARK (zkSNARK) that satisfies knowledge-soundness (see Appendix F for definitions) to prove knowledge of a valid IVC proof Π_n. While the original Nova proposal did not hide the number of steps n, we use an implementation by Angel *et al.* [2, 3] that does hide n.

(1) The prover $\mathcal{P}_{zk}$ and verifier $\mathcal{V}_{zk}$ of the zkSNARK are given the instance (F, z_0, z_n). The proving key pk and verification key vk are derived from the R1CS constraints expressing F.

(2) The prover $\mathcal{P}_{zk}$ uses the proving key pk, IO values z_0, z_n, and IVC proof Π_n to produce the proof π.

$$\pi \leftarrow \mathcal{P}_{zk}(\mathsf{pk}, z_0, z_n, \Pi_n).$$

(3) The verifier $\mathcal{V}_{zk}$ takes the verification key vk, proof π, and z_0, z_n as inputs. It either accepts the proof or rejects it.

$$0/1 \leftarrow \mathcal{V}_{zk}(\mathsf{vk}, z_0, z_n, \pi).$$

If the zkSNARK is based on a Pedersen commitment scheme for vectors, then the proof size is $O(\log |F|)$ and the proof generation and verification times are both $O(|F|)$. Note that these metrics are independent of the number of steps n used to generate the IVC proof Π_n.

6 The Design of MProve-Nova

In this section, we provide a high-level motivation of MProve-Nova. Previous Monero PoR protocols, MProve and MProve+ [21, 22], suffer from the following drawbacks (see Appendices B, C).

- A PoR instance in both MProve and MProve+ contains a set of one-time addresses $\mathcal{P}_{\text{anon}}$ called the anonymity set. The exchange-owned addresses $\mathcal{P}_{\text{known}}$ are a subset of $\mathcal{P}_{\text{anon}}$. To verify an MProve/MProve+ proof, the verifier has to first download $\mathcal{P}_{\text{anon}}$. While a larger $\mathcal{P}_{\text{anon}}$ allows an exchange to hide its addresses in a larger anonymity set, it also increases the download requirement for verifiers.
- Proof sizes increase either linearly with the size of $\mathcal{P}_{\text{anon}}$ (in the case of MProve) or linearly with the size of $\mathcal{P}_{\text{known}}$ (in the case of MProve+).
- Proof verification times increase linearly with the size of $\mathcal{P}_{\text{anon}}$, raising the bar for verifiers.
- The key images of the exchange-owned addresses are revealed, negatively impacting the privacy of both exchanges and regular Monero users.

To reduce the size of the PoR protocol instance, we need to compress the information about the addresses on the Monero blockchain into a succinct form. MProve-Nova achieves instance compression by taking as input the root of a Merkle tree having the Monero outputs as leaves. This tree is called the *transaction outputs tree (TXOT).* The advantage of this approach is that the size of the TXOT root is independent of the number of outputs on the Monero blockchain. This allows MProve-Nova to have an anonymity set which is the set of all outputs, instead of only a small subset of the outputs like in MProve and MProve+.

To prove ownership of an output, MProve-Nova gives a privacy-preserving proof of knowledge of a Merkle path to an output leaf in the TXOT tree and a proof of knowledge of the corresponding private key. This strategy was pioneered by ZCash [8, 17] and has been used in projects like Tornado Cash [59] and Semaphore [43].

In addition to proving ownership of an output, a Monero PoR protocol has to also prove that the output is unspent. MProve and MProve+ achieved this requirement by explicitly revealing the key images of the exchange-owned outputs. MProve-Nova takes as an additional input the root of an indexed Merkle tree [49] having all the key images on the Monero blockchain as leaves. This tree is called the *key images tree (KIT)*. It then gives privacy-preserving proofs of non-membership of the key images of exchange-owned outputs in this tree.

The key images revealed by MProve and MProve+ also helped to prove that exchanges did not collude, i.e. they did not share an output while generating their respective proofs of reserves. If two exchanges had shared an output, its key image would have appeared in both their proofs. Non-collusion can be easily verified by checking that the key image sets revealed by the exchanges have no elements in common. Since MProve-Nova uses privacy-preserving proofs of non-membership of output key images in the KIT, one can no longer verify non-collusion by simply inspecting the proofs. To solve this issue, an MProve-Nova proof outputs a Merkle root of a tree, called the *double spend tree (DST)*, containing only the Monero outputs owned by an exchange. Using their respective DST roots, two exchanges can prove non-collusion using a series of non-membership proofs. Essentially, one of the exchanges will prove that none of the leaves from its own DST are present in the other exchange's DST. This procedure is called the *non-collusion (NC) protocol* in MProve-Nova.

To reduce proof sizes and verification times, MProve-Nova uses the Nova recursive SNARK to break down the task of proving ownership of multiple unspent outputs into smaller steps, where each step proves the ownership of a single unspent output. Similarly, the task of proving non-collusion is divided into steps, where each step proves that an output from the one exchange's DST is absent in the other exchange's DST. In Nova, proof sizes and verification times depend only on the size of the step circuit and not on the number of steps [30]. Consequently, the MProve-Nova proof size and verification time is independent of the number of outputs on the Monero blockchain and the number of exchange-owned outputs.

In an MProve-Nova step, the amount of coins associated with an unspent output is accumulated in a Pedersen commitment. After the last step, the final commitment is a commitment to the exchange's reserves. In addition to its role in the NC protocol, the DST also helps ensure that an exchange does not repeat an unspent output across steps, in order to inflate its reserves. This kind of "double spending" is prevented via a non-membership proof of the current step's output in the previous step's DST (justifying the DS in DST).

7 Reserves Commitment Generator Protocol

Both MProve and MProve+ output a Pedersen commitment C_{res} to the total amount of coins owned by the exchange (its reserves). MProve-Nova follows the same strategy, with the reserves commitment generator (RCG) protocol responsible for generating C_{res}.

Pedersen commitments to the reserves amount enable an exchange to hide the amount while allowing it to prove solvency. If the exchange wants to prove that its reserves are greater than or equal to an amount a, it can give a range proof on $C_{\text{res}} - aH$ proving that it is a commitment to a non-negative amount. The amount a may correspond to the total liabilities of the exchange towards its customers. If the exchange wants to hide the amount a, it can use a proof of liabilities protocol that outputs only a Pedersen commitment C_{liab} to the amount a. Then the exchange can prove solvency by giving a range proof on $C_{\text{res}} - C_{\text{liab}}$ proving that it is a commitment to a non-negative amount.

Recall from Section 2 that a Monero output consists of a pair (P, C) where P is a one-time address and C is a Pedersen commitment to the number of coins associated with P.

- Let $\mathcal{T}_{\text{bh}} = [(P_1, C_1), (P_2, C_2), \ldots, (P_{N_{\text{bh}}}, C_{N_{\text{bh}}})]$ be the vector of all transaction outputs that have appeared in Monero blocks up to height bh, sorted in the order of their appearance on the Monero blockchain.
- Let $\mathcal{I}_{\text{bh}}$ be the set of all key images that have appeared in Monero blocks up to height bh.
- Let $\mathcal{J}_{\text{known}} \subset \{1, 2, \ldots, N_{\text{bh}}\}$ be the set of output indices such that an exchange knows the private keys corresponding the address P_i for all $i \in \mathcal{J}_{\text{known}}$.
- Let $\mathcal{J}_{\text{unspent}} \subset \{1, 2, \ldots, N_{\text{bh}}\}$ be the set of output indices corresponding to unspent outputs. In other words, for each $i \in \mathcal{J}_{\text{unspent}}$ the key image of P_i has not appeared in $\mathcal{I}_{\text{bh}}$.

If C_i is a commitment to the amount a_i, then the total amount of coins owned by the exchange is

$$a_{\text{tot}} = \sum_{i \in \mathcal{J}_{\text{known}} \bigcap \mathcal{J}_{\text{unspent}}} a_i$$

Even though the exchange owns a_{tot} coins, its liabilities may be lower. So it may only want to prove that it owns a_{res} coins, where $a_{\text{res}} < a_{\text{tot}}$. The RCG protocol allows an exchange to generate a Pedersen commitment C_{res} to any amount a_{res} that satisfies

$$a_{\text{res}} = \sum_{i \in \mathcal{J}_{\text{res}}} a_i, \tag{1}$$

where $\mathcal{J}_{\text{res}}$ is a non-empty subset of $\mathcal{J}_{\text{known}} \bigcap \mathcal{J}_{\text{unspent}}$.

7.1 Transaction Outputs and Key Images Merkle Trees

To efficiently prove and verify that the amount in commitment C_{res} has contributions only from unspent outputs owned by the exchange, it is necessary to convert the information about transaction outputs and key images into a succinct form. To this end, we construct a regular Merkle tree from the transactions outputs in $\mathcal{T}_{\text{bh}}$ and an indexed Merkle tree from the key images set $\mathcal{I}_{\text{bh}}$. Indexed Merkle trees were introduced by Tzialla *et. al* [49] to generate efficient non-membership proofs inside a SNARK.

Both trees are constructed using the Poseidon hash function [23] to reduce the number of R1CS constraints[4] in the computation that will eventually be proved using Nova. Let $H_{pos} : \{0,1\}^* \mapsto \mathbb{F}_s$ be the Poseidon hash function where $\mathbb{F}_s$ is the scalar field used to express the R1CS constraints. The two trees are constructed as follows.

(1) **Transaction Outputs Tree TXOT**: Let $H_{\text{p}} : \mathbb{G} \mapsto \mathbb{G}$ be the cryptographic hash function used to compute the key image of a one-time address $P = xG$ as $xH_{\text{p}}(P)$. Let $\|$ denote the string concatenation operator.
The transactions outputs tree TXOT is a regular Merkle tree constructed using the sequence of leaves
$$\left\{H_{pos}\left(P_i \| C_i \| H_{\text{p}}(P_i)\right) \mid i = 1, 2, \ldots, N_{\text{bh}}\right\}.$$
We omit the dependence of TXOT on bh for simplicity. The motivation for choosing this particular leaf structure is described in Section 7.2.

(2) **Key Images Tree KIT**: This is an indexed Merkle tree constructed using the leaves $\{H_{pos}(I) \mid I \in \mathcal{I}_{\text{bh}}\}$. Once again, we omit the dependence of KIT on bh for simplicity.

For a Monero block height bh, the trees TXOT and KIT are uniquely defined and can be constructed by any party who has a copy of the Monero blockchain.

7.2 RCG Protocol Instance and Witness

Definition 7.1. An instance of the RCG protocol is given by the tuple

$$\mathsf{inst}_{\text{RCG}} = (\text{bh}, \text{root(TXOT)}, \text{root(KIT)}, C_{\text{res}}),$$

where C_{res} is a Pedersen commitment and root (TXOT), root (KIT) are the root hashes of the TXOT and KIT trees, respectively.

Let $\mathsf{MerkleProof}(\text{root}, v, i)$ denote a Merkle proof that the ith leaf in a Merkle tree with root hash equal to root has value v. Let $\mathsf{NonMemberProof}(\text{root}, v)$ denote a proof of non-membership of a leaf having value v in an indexed Merkle tree with root hash equal to root. Both these types of proofs consist of a list of sibling nodes along the path from a leaf to the tree root.

Definition 7.2. A witness for an RCG protocol instance $\mathsf{inst}_{\text{RCG}}$ is a tuple

$$\mathsf{wit}_{\text{RCG}} = \left(\mathcal{J}_{\text{res}}, \mathbf{x}, \mathbf{P}', \mathbf{C}', \mathbf{H}, \pi_{\text{mp}}, \pi_{\text{nmp}}, r_{\text{res}}\right),$$

whose components satisfy the following properties.

(i) $\mathcal{J}_{res} = \{i_1, i_2, \ldots, i_n\} \subset \{1, 2, \ldots, N_{\text{bh}}\}$ is a set of n *distinct* indices of leaves in TXOT.

(ii) $\mathbf{x} = [x_1, x_2, \ldots, x_n] \in \mathbb{Z}_l^n$ is a vector of n scalars.

(iii) $\mathbf{P}' = [P'_1, P'_2, \ldots, P'_n] \in \mathbb{G}^n$ is a vector of n group elements such that $P'_j = x_j G$, i.e. x_j is the private key corresponding to P'_j for each $j = 1, 2, \ldots, n$.

(iv) $\mathbf{C}' = [C'_1, C'_2, \ldots, C'_n] \in \mathbb{G}^n$ and $\mathbf{H} = [H_1, H_2, \ldots, H_n] \in \mathbb{G}^n$ are vectors of n group elements.

(v) π_{mp} is a vector of n Merkle proofs where the jth proof equals

$$\mathsf{MerkleProof}\left(\text{root(TXOT)}, H_{pos}(P'_j \| C'_j \| H_j), i_j\right),$$

[4] Fewer R1CS constraints are desirable as they translate to smaller proofs and faster proof generation and verification times.

for each $j = 1, 2, \ldots, n$.

(vi) π_{nmp} is a vector of n non-membership proofs where the jth proof equals

$$\mathsf{NonMemberProof}\left(\mathsf{root}(\mathsf{KIT}), H_{pos}(x_j H_j)\right),$$

for each $j = 1, 2, \ldots, n$.

(vii) $r_{\text{res}} \in \mathbb{Z}_l$ is a scalar that satisfies

$$C_{\text{res}} = r_{\text{res}} G + \sum_{j=1}^{n} C'_j. \tag{2}$$

We claim that if an exchange knows a witness $\mathsf{wit}_{\text{RCG}}$ for an RCG protocol instance $\mathsf{inst}_{\text{RCG}}$, then the C_{res} is a commitment to an amount a_{res} that satisfies equation (1), assuming that the Poseidon hash function is collision resistant. We argue as follows:

- The jth Merkle proof in π_{mp} proves that the leaf at index i_j in the TXOT tree has value $H_{pos}(P'_j \| C'_j \| H_j)$. By construction, the leaf at index i_j in the TXOT tree has value $H_{pos}(P_{i_j} \| C_{i_j} \| H_{\text{p}}(P_{i_j}))$. From the collision resistance of the Poseidon hash function, it follows that $P'_j = P_{i_j}$, $C'_j = C_{i_j}$, and $H_j = H_{\text{p}}(P_{i_j})$.
  So the Merkle proofs in π_{mp} prove that the addresses in $\mathbf{P}'$ have appeared on the Monero blockchain and their corresponding Pedersen commitments are present in $\mathbf{C}'$. Furthermore, the exchange owns the addresses in $\mathbf{P}'$ since it knows $\mathbf{x}$ which contains the corresponding private keys. Thus $\mathcal{J}_{\text{res}} \subset \mathcal{J}_{\text{known}}$.
- Since x_j is the private key corresponding to $P'_j = P_{i_j}$ and $H_j = H_{\text{p}}(P_{i_j})$, the point $x_j H_j = x_j H_{\text{p}}(P_{i_j})$ is the key image of the one-time address P_{i_j}. So the non-membership proofs in π_{nmp} prove that the key images of the addresses in $\mathbf{P}'$ have not appeared on the Monero blockchain, i.e. they are unspent. Thus $\mathcal{J}_{\text{res}} \subset \mathcal{J}_{\text{unspent}}$.
- We have $\mathcal{J}_{\text{res}} \subset \mathcal{J}_{\text{known}} \bigcap \mathcal{J}_{\text{unspent}}$. Since $C'_j = C_{i_j}$, we can rewrite equation (2) as

$$C_{\text{res}} = r_{\text{res}} G + \sum_{j=1}^{n} C_{i_j} = r_{\text{res}} G + \sum_{i \in \mathcal{J}_{\text{res}}} C_i.$$

  Thus C_{res} is a commitment to an amount a_{res} that satisfies equation (1).

An exchange will give a zero-knowledge argument of knowledge of a witness for an instance $\mathsf{inst}_{\text{RCG}}$. In this instance, only C_{res} could possibly leak information about $\mathcal{J}_{\text{res}}$. The scalar r_{res} acts as a blinding factor to prevent an adversary from identifying the set $\mathcal{J}_{\text{res}}$ from C_{res}. As $C'_j = C_{i_j}$, if the $r_{\text{res}} G$ term were absent in equation (2), an adversary could attempt to identify $\mathcal{J}_{\text{res}}$ by finding a subset of commitments on the Monero blockchain that sum to C_{res}.

The motivation for including $H_{\text{p}}(P_i)$ in the leaves of the TXOT tree is that it reduces the number of R1CS constraints required to express the key image non-membership proofs. If it were not present in the leaves, then the key image computation of an address P_i would have to be expressed using R1CS constraints. This computation involves the Keccak hash function [9], which requires a large number of R1CS constraints. When $H_{\text{p}}(P_i)$ is included in the leaves, the Merkle proofs in π_{mp} ensure that $H_j = H_{\text{p}}(P_{i_j})$. So only the scalar multiplication $x_j H_j$ needs to be expressed using R1CS constraints to compute the key image of P_{i_j}.

7.3 The RCG Protocol as an IVC Scheme

To take advantage of the Nova recursive SNARK, we express the RCG protocol as an IVC scheme. Let F_{RCG} be the step function for an IVC scheme corresponding to the RCG protocol. The exchange will prove knowledge of a witness $\mathsf{wit}_{\text{RCG}}$ by proving the correct execution of n invocations of the step function F_{RCG}.

For $j = 1, 2, \ldots, n$, in the jth step the function F_{RCG} will check knowledge of the private key x_j, verify the jth Merkle proof in π_{mp}, verify the jth non-membership proof in π_{nmp}, and accumulate C'_j into C_{res}. A complete specification of the F_{RCG} is given in Section 7.4. We now introduce some notation that is used in the specification.

7.3.1 Double Spend Tree (DST). An important constraint on the witness $\mathsf{wit}_{\text{RCG}}$ is that the indices in $\mathcal{J}_{\text{res}}$ have to be distinct. Otherwise, an exchange can inflate its reserves by double counting an output. To ensure that outputs already considered in previous invocations of F_{RCG} are not considered again, we use an indexed Merkle tree called the **Double Spend Tree** DST. This DST is also used to prove non-collusion between exchanges, as explained in Section 8.

For $j = 1, 2, \ldots, n$, let DST_{j-1} and DST_j denote the state of the double spend tree before and after the jth invocation of F_{RCG}, respectively. The initial DST_0 corresponds to an empty indexed Merkle tree $\mathsf{IMT}_\emptyset$. Let $\mathsf{InsertProof}(\mathsf{root}_1, \mathsf{root}_2, v)$ denote the insertion proof that proves that inserting a leaf having value v into an indexed Merkle tree with root hash root_1 results in the indexed Merkle tree with root hash root_2.

Recall that x_j is the private key of the jth one-time address P_{i_j} used by the exchange. At step j, the function F_{RCG} does the following:

(i) Using $\mathsf{NonMemberProof}\left(\mathsf{root}(\mathsf{DST}_{j-1}), H_{pos}(x_j \| \mathsf{bh})\right)$, it verifies that the leaf $H_{pos}(x_j \| \mathsf{bh})$ is not present in DST_{j-1}.

(ii) Using $\mathsf{InsertProof}\left(\mathsf{root}(\mathsf{DST}_{j-1}), \mathsf{root}(\mathsf{DST}_j), H_{pos}(x_j \| \mathsf{bh})\right)$, it verifies that $\mathsf{root}(\mathsf{DST}_j)$ is the root hash of the indexed Merkle tree obtained by inserting the leaf $H_{pos}(x_j \| \mathsf{bh})$ into an indexed Merkle tree with root hash equal to $\mathsf{root}(\mathsf{DST}_{j-1})$.

The non-membership and insertion proofs are provided as private inputs to the step function F_{RCG}. In the jth step, the public input to F_{RCG} contains the root hash $\mathsf{root}(\mathsf{DST}_{j-1})$ and its public output contains the root hash $\mathsf{root}(\mathsf{DST}_j)$. The public inputs/outputs and private inputs to F_{RCG} are shown in Figure 2.

The non-membership proofs and insertion proofs ensure that an output which is used in step j cannot be used again in steps $j+1, j+2, \ldots, n$. We could have obtained the same guarantee by inserting any one of $H_{pos}(x_j)$, $H_{pos}(P_{i_j})$ or $H_{pos}(P_{i_j} \| \mathsf{bh})$ into the DST, instead of inserting $H_{pos}(x_j \| \mathsf{bh})$.

- The problem with using $H_{pos}(P_{i_j})$ or $H_{pos}(P_{i_j} \| \mathsf{bh})$ as leaves in the DST is that an adversary could try combinations of outputs to reconstruct the set $\mathcal{J}_{\text{res}}$ from the public root hash of the DST.
- The problem with using $H_{pos}(x_j)$ as leaves in the DST is that the public root hash of the DST would remain the same if the exchange uses the *same set of outputs* to generate its

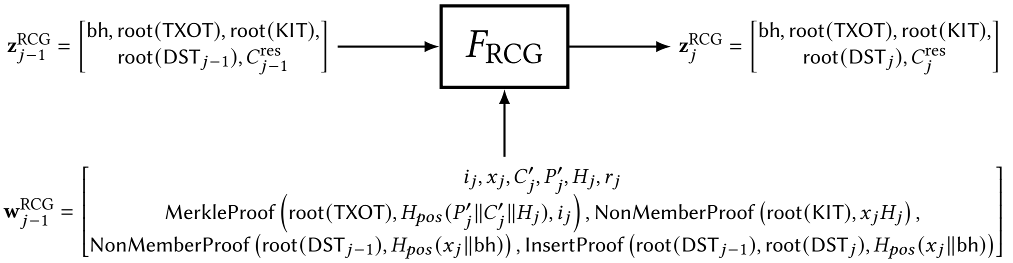
$\mathbf{z}_{j-1}^{\text{RCG}} = \begin{bmatrix} \text{bh}, \text{root}(\text{TXOT}), \text{root}(\text{KIT}), \\ \text{root}(\text{DST}_{j-1}), C_{j-1}^{\text{res}} \end{bmatrix}$
F_{RCG}
$\mathbf{z}_{j}^{\text{RCG}} = \begin{bmatrix} \text{bh}, \text{root}(\text{TXOT}), \text{root}(\text{KIT}), \\ \text{root}(\text{DST}_{j}), C_{j}^{\text{res}} \end{bmatrix}$
$\mathbf{w}_{j-1}^{\text{RCG}} = \begin{bmatrix} i_j, x_j, C'_j, P'_j, H_j, r_j \\ \text{MerkleProof}\left(\text{root}(\text{TXOT}), H_{pos}(P'_j \| C'_j \| H_j), i_j\right), \text{NonMemberProof}\left(\text{root}(\text{KIT}), x_j H_j\right), \\ \text{NonMemberProof}\left(\text{root}(\text{DST}_{j-1}), H_{pos}(x_j \| \text{bh})\right), \text{InsertProof}\left(\text{root}(\text{DST}_{j-1}), \text{root}(\text{DST}_j), H_{pos}(x_j \| \text{bh})\right) \end{bmatrix}$

Figure 2: Step function for MProve-Nova RCG protocol. In the jth step of the RCG protocol, $\mathbf{z}_{j-1}^{\text{RCG}}$ is the public input, $\mathbf{w}_{j-1}^{\text{RCG}}$ is the private input, and $\mathbf{z}_{j}^{\text{RCG}}$ is the public output.

reserves at *different* block heights. This can leak information about the asset strategy of the exchange.

7.3.2 Incremental Accumulation of the Reserves Amount. In the jth step, the public input of F_{RCG} will contain a Pedersen commitment C_{j-1}^{res} and its public output will contain a Pedersen commitment C_{j}^{res}. These commitments enable the incremental accumulation of the reserves amount across the n steps.

C_{j-1}^{res} is a Pedersen commitment to the reserves amount accumulated *before* step j. C_0^{res} is set to G, which commits to the zero amount with blinding factor 1. C_j^{res} is a Pedersen commitment to the reserves amount accumulated *after* step j. The final commitment C_n^{res} will be equal to the commitment C_{res} in the RCG protocol instance.

At the jth step, using a private scalar r_j the function F_{RCG} computes C_j^{res} as

$$C_j^{\text{res}} = C_{j-1}^{\text{res}} + C'_j + r_j G, \tag{3}$$

where C'_j is the jth group element in $\mathbf{C}'$. Like the scalar r_{res} in the RCG protocol witness, the role of r_j is to prevent an adversary from using the public values C_{j-1}^{res} and C_j^{res} to identify the output contributing to the reserves in the jth step.

7.4 Step Function Computation

We are now ready to specify the step function F_{RCG}. Let n be the number of outputs the exchange will use to contribute to its reserves. For $j = 1, 2, \ldots, n$, the **public input** $\mathbf{z}_{j-1}^{\text{RCG}}$ to F_{RCG} in step j is

$$\mathbf{z}_{j-1}^{\text{RCG}} = \left[\text{bh}, \text{root}(\text{TXOT}), \text{root}(\text{KIT}), \text{root}(\text{DST}_{j-1}), C_{j-1}^{\text{res}}\right].$$

The **public output** $\mathbf{z}_j$ of F_{RCG} in step j is

$$\mathbf{z}_{j}^{\text{RCG}} = \left[\text{bh}, \text{root}(\text{TXOT}), \text{root}(\text{KIT}), \text{root}(\text{DST}_{j}), C_{j}^{\text{res}}\right].$$

Not that $\mathbf{z}_{j-1}^{\text{RCG}}$ and $\mathbf{z}_{j}^{\text{RCG}}$ differ only in the last two components.

The **private input** $\mathbf{w}_{j-1}^{\text{RCG}}$ to F_{RCG} in step j is

$$\mathbf{w}_{j-1}^{\text{RCG}} = \begin{bmatrix} i_j, x_j, C'_j, P'_j, H_j, r_j \\ \text{MerkleProof}\left(\text{root}(\text{TXOT}), H_{pos}(P'_j \| C'_j \| H_j), i_j\right), \\ \text{NonMemberProof}\left(\text{root}(\text{KIT}), H_{pos}(x_j H_j)\right), \\ \text{NonMemberProof}\left(\text{root}(\text{DST}_{j-1}), H_{pos}(x_j \| \text{bh})\right), \\ \text{InsertProof}\left(\text{root}(\text{DST}_{j-1}), \text{root}(\text{DST}_j), H_{pos}(x_j \| \text{bh})\right) \end{bmatrix},$$

where $i_j, x_j, C'_j, P'_j, H_j$ are as defined in Section 7.2. The DST_{j-1}, DST_j, r_j terms were defined in Section 7.3.

For each $j = 1, 2, \ldots, n$, in step j function F_{RCG} performs the following computations:

(1) Using private input x_j, it computes $x_j G$ and checks that it is equal to the point P'_j. This proves knowledge of the private key corresponding to P'_j.
(2) Using private inputs P'_j, C'_j, H_j, it computes the leaf hash $H_{pos}\left(P'_j \| C'_j \| H_j\right)$.
(3) Using $\text{MerkleProof}\left(\text{root}(\text{TXOT}), H_{pos}(P'_j \| C'_j \| H_j), i_j\right)$, it verifies that the leaf hash computed above belongs to TXOT. This proves that (P'_j, C'_j) is an output on the Monero blockchain and that $H_j = H_{\text{p}}(P'_j)$ (as explained in Section 7.2).
(4) Using the private inputs x_j and H_j, it computes the point $x_j H_j$ and its hash $H_{pos}(x_j H_j)$.
(5) Using $\text{NonMemberProof}\left(\text{root}(\text{KIT}), H_{pos}(x_j H_j)\right)$, it proves the non-membership of $H_{pos}(x_j H_j)$ in KIT. This proves that P'_j is an unspent address.
(6) Using the private input x_j, it computes the hash $H_{pos}(x_j \| \text{bh})$.
(7) Using $\text{NonMemberProof}\left(\text{root}(\text{DST}_{j-1}), H_{pos}(x_j \| \text{bh})\right)$, it verifies the non-membership of leaf $H_{pos}(x_j \| \text{bh})$ in DST_{j-1}. This proves that the address P'_j has not been used in a previous step.
(8) Using $\text{InsertProof}\left(\text{root}(\text{DST}_{j-1}), \text{root}(\text{DST}_j), H_{pos}(x_j \| \text{bh})\right)$, it verifies that $\text{root}(\text{DST}_j)$ is the root hash of the indexed Merkle tree obtained by inserting the leaf $H_{pos}(x_j \| \text{bh})$ into an indexed Merkle tree with root hash equal to $\text{root}(\text{DST}_{j-1})$.
(9) Using the private input r_j, it computes the Pedersen commitment $C_j^{\text{res}} = C_{j-1}^{\text{res}} + C'_j + r_j G$.
(10) Using $\text{root}(\text{DST}_j), C_j^{\text{res}}$, it updates $\mathbf{z}_{j-1}^{\text{RCG}}$ and outputs $\mathbf{z}_{j}^{\text{RCG}}$.

7.5 Knowledge of IVC Witness Implies Knowledge of RCG Protocol Witness

In this section, we argue that knowledge of a witness for the IVC version of the RCG protocol implies the knowledge of a witness wit_{RCG} for an original RCG protocol instance inst_{RCG} (see Definition 7.2).

Definition 7.3. Let $\text{IMT}_\emptyset$ denote the empty indexed Merkle tree. An instance of the IVC version of the RCG protocol is the tuple

$\mathsf{inst}_{\text{RCG-IVC}} = (\mathbf{z}_0^{\text{RCG}}, \mathbf{z}_n^{\text{RCG}})$, where

$$\mathbf{z}_0^{\text{RCG}} = \left[\mathsf{bh}, \mathsf{root}(\mathsf{TXOT}), \mathsf{root}(\mathsf{KIT}), \mathsf{root}(\mathsf{IMT}_\emptyset), G\right],$$
$$\mathbf{z}_n^{\text{RCG}} = \left[\mathsf{bh}, \mathsf{root}(\mathsf{TXOT}), \mathsf{root}(\mathsf{KIT}), \mathsf{root}(\mathsf{DST}_n), C_n^{\text{res}}\right].$$

The point G in $\mathbf{z}_0^{\text{RCG}}$ corresponds to the commitment to the zero amount with blinding factor 1.

Definition 7.4. A witness for the instance $\mathsf{inst}_{\text{RCG-IVC}}$ is a vector of private inputs

$$\mathsf{wit}_{\text{RCG-IVC}} = \left[\mathbf{w}_0^{\text{RCG}}, \mathbf{w}_1^{\text{RCG}}, \ldots, \mathbf{w}_{n-1}^{\text{RCG}}\right]$$

such that for all $i = 0, 1, \ldots, n-1$ we have

$$\mathbf{z}_{i+1}^{\text{RCG}} = F_{\text{RCG}}(\mathbf{z}_i^{\text{RCG}}, \mathbf{w}_i^{\text{RCG}}).$$

Suppose an exchange knows a witness $\mathsf{wit}_{\text{RCG-IVC}}$ of length n. Then the values in the set $\mathcal{J}_{\text{res}}$ and vectors $\mathbf{x}, \mathbf{P}', \mathbf{C}', \mathbf{H}, \pi_{\text{mp}}, \pi_{\text{nmp}}$ required for $\mathsf{wit}_{\text{RCG}}$ are already present in $\mathsf{wit}_{\text{RCG-IVC}}$. As discussed in Section 7.3.1, the double spend tree guarantees that the indices in $\mathcal{J}_{\text{res}}$ are distinct. Finally, for $C_{\text{res}} = C_n^{\text{res}}$ the value of r_{res} in $\mathsf{wit}_{\text{RCG}}$ is given by $1 + \sum_{j=1}^{n} r_j$, since $C_0^{\text{res}} = G$ contributes the blinding factor 1.

7.6 Proof Generation and Verification

Prior to proof generation and verification, the exchange and the verifier use the structure of F_{RCG} to generate the Nova proving key pk_{RCG} and verification key vk_{RCG}, respectively. This has to be done only once.

7.6.1 Proof Generation. At a block height bh, the exchange first constructs the TXOT and KIT trees. At subsequent block heights, only the outputs and key images that appeared after block bh need to be added to these trees.

Suppose the exchange wants to use n outputs to generate the commitment C_{res} to its reserves. The exchange constructs the private inputs $\mathbf{w}_{j-1}^{\text{RCG}}, j = 1, 2, \ldots, n$, as defined in Section 7.4. The scalar r_j is chosen uniformly from $\mathbb{Z}_l$ for each j.

Starting from the public input $\mathbf{z}_0^{\text{RCG}}$ given in Definition 7.3, the exchange computes the remaining $\mathbf{z}_j^{\text{RCG}}$ values using $\mathbf{z}_{j+1}^{\text{RCG}} = F_{\text{RCG}}(\mathbf{z}_j^{\text{RCG}}, \mathbf{w}_j^{\text{RCG}})$ for $j = 0, 1, \ldots, n-1$. It then uses the Nova IVC scheme to generate an IVC proof Π_n attesting to the correct execution of the n steps (see Section 5.1). Finally, it generates a zkSNARK proof π_{RCG} proving knowledge of the IVC proof Π_n (see Section 5.2).

The tuple $(\mathsf{inst}_{\text{RCG-IVC}}, \pi_{\text{RCG}}) = ((\mathbf{z}_0^{\text{RCG}}, \mathbf{z}_n^{\text{RCG}}), \pi_{\text{RCG}})$ is shared with the zkSNARK verifier $\mathcal{V}_{\text{RCG}}$.

7.6.2 Proof Verification. The zkSNARK verifier $\mathcal{V}_{\text{RCG}}$ verifies the proof π_{RCG} as

$$0/1 \longleftarrow \mathcal{V}_{\text{RCG}}\left(\mathsf{vk}_{\text{RCG}}, \mathsf{inst}_{\text{RCG-IVC}}, \pi_{\text{RCG}}\right).$$

Note that the number of steps n used by the prover to obtain $\mathbf{z}_n^{\text{RCG}}$ is not revealed to the verifier. If the proof is accepted, the commitment C_n^{res} in $\mathbf{z}_n^{\text{RCG}}$ is a commitment to the exchange's reserves amount *assuming* that the root hashes root(TXOT) and root(KIT) are correct.

Note that the proof π_{RCG} does not verify that the root hashes root(TXOT) and root(KIT) actually correspond to the TXOT and KIT trees at block height bh. If an exchange publishes incorrect root hashes for these trees, it can inflate its reserves by using an output which is not on the Monero blockchain or by using a spent output. But incorrect root hashes can be detected as they are calculated from public data available on the Monero blockchain.

In theory, we could generate a Nova proof proving that a pair of root hashes correspond to the TXOT and KIT trees at block height bh. But the locations of the outputs and key images in Monero blocks can take many possible values, making it hard to express this computation using R1CS constraints.

8 Non-Collusion Protocol

In this section, we describe the non-collusion (NC) protocol that is used in MProve-Nova. It proves that a pair of exchanges that both used the RCG protocol to generate commitments to their reserves have not colluded, i.e. they did not use a common output to contribute to their respective reserves.

8.1 NC Protocol Instance and Witness

Suppose two exchanges Ex1 and Ex2, that have executed the RCG protocol at the same block height bh, want to prove that they have not colluded. Let n_1 and n_2 be the number of owned outputs of Ex1 and Ex2, respectively. Let $\mathsf{root}\left(\mathsf{DST}_{n_1}^{\mathsf{Ex1}}\right)$ and $\mathsf{root}\left(\mathsf{DST}_{n_2}^{\mathsf{Ex2}}\right)$ be the root hashes of the double spend trees output by the exchanges at the end of their respective RCG protocol executions.

Definition 8.1. An instance of the non-collusion protocol is given by the pair

$$\mathsf{inst}_{\text{NC}} = \left(\mathsf{root}\left(\mathsf{DST}_{n_1}^{\mathsf{Ex1}}\right), \mathsf{root}\left(\mathsf{DST}_{n_2}^{\mathsf{Ex2}}\right)\right).$$

In the NC protocol, one of the exchanges generates the proof using information from the other exchange. Suppose Ex1 generates the proof using information from Ex2. Then we can define the NC protocol witness as follows.

Definition 8.2. A witness for an NC protocol instance $\mathsf{inst}_{\text{NC}}$ is a pair $\mathsf{wit}_{\text{NC}} = (\mathbf{v}, \pi_{\text{nmp}})$, where

(i) the root hash of the indexed Merkle tree constructed using the vector of leaves $\mathbf{v} = [v_1, v_2, \ldots, v_{n_2}] \in \mathbb{F}_s^{n_2}$ is equal to $\mathsf{root}\left(\mathsf{DST}_{n_2}^{\mathsf{Ex2}}\right)$, and

(ii) π_{nmp} is a vector of n_2 non-membership proofs where the jth proof equals

$$\mathsf{NonMemberProof}\left(\mathsf{root}\left(\mathsf{DST}_{n_1}^{\mathsf{Ex1}}\right), v_j\right),$$

for each $j = 1, 2, \ldots, n_2$.

Let $\mathbf{v}^{(1)} = \left[v_1^{(1)}, v_2^{(1)}, \ldots, v_{n_1}^{(1)}\right]$ be the vector of leaves in $\mathsf{DST}_{n_1}^{\mathsf{Ex1}}$ and let $\mathbf{v}^{(2)} = \left[v_1^{(2)}, v_2^{(2)}, \ldots, v_{n_2}^{(2)}\right]$ be the vector of leaves in $\mathsf{DST}_{n_2}^{\mathsf{Ex2}}$. To allow Ex1 to prove knowledge of the witness wit_{NC}, Ex2 shares the leaves $\mathbf{v}^{(2)}$ of its double spend tree with Ex1. Note that these leaves are not revealed during the execution of the RCG protocol execution.

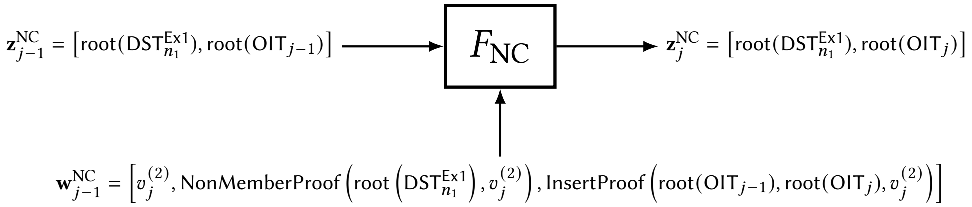
$\mathbf{z}_{j-1}^{\mathrm{NC}} = \left[\mathsf{root}(\mathsf{DST}_{n_1}^{\mathsf{Ex1}}), \mathsf{root}(\mathsf{OIT}_{j-1})\right]$
F_{NC}
$\mathbf{z}_{j}^{\mathrm{NC}} = \left[\mathsf{root}(\mathsf{DST}_{n_1}^{\mathsf{Ex1}}), \mathsf{root}(\mathsf{OIT}_{j})\right]$
$\mathbf{w}_{j-1}^{\mathrm{NC}} = \left[v_j^{(2)}, \mathsf{NonMemberProof}\left(\mathsf{root}\left(\mathsf{DST}_{n_1}^{\mathsf{Ex1}}\right), v_j^{(2)}\right), \mathsf{InsertProof}\left(\mathsf{root}(\mathsf{OIT}_{j-1}), \mathsf{root}(\mathsf{OIT}_{j}), v_j^{(2)}\right)\right]$

Figure 3: Step Function for MProve-Nova Non-Collusion protocol. In the jth step of the NC protocol, $\mathbf{z}_{j-1}^{\mathrm{NC}}$ is the public input, $\mathbf{w}_{j-1}^{\mathrm{NC}}$ is the private input, and $\mathbf{z}_{j}^{\mathrm{NC}}$ is the public output.

The roles of Ex1 and Ex2 can be interchanged, but then Ex1 will have to share $\mathbf{v}^{(1)}$ with Ex2. In our protocol description, we assume that Ex1 generates the non-collusion proof using $\mathbf{v}^{(2)}$ from Ex2.

Recall that $v_j^{(1)} = H_{pos}\left(x_j^{(1)} \| \mathsf{bh}\right)$ for $j = 1, 2, \ldots, n_1$ and $v_j^{(2)} = H_{pos}\left(x_j^{(2)} \| \mathsf{bh}\right)$ for $j = 1, 2, \ldots, n_2$, where the $x_j^{(1)}$s and $x_j^{(2)}$s are the private keys of unspent outputs owned by Ex1 and Ex2, respectively. Knowledge of a witness $\mathsf{wit}_{\mathrm{NC}}$ proves that $\mathbf{v}^{(1)}$ and $\mathbf{v}^{(2)}$ have no elements in common. Assuming that the Poseidon hash function is collision resistant, the sets of private keys used by the two exchanges have no elements in common. This implies that they have not colluded.

8.2 Step Function Computation

We express the NC protocol as an IVC scheme. Let F_{NC} be the corresponding step function. Ex1 will prove knowledge of a witness $\mathsf{wit}_{\mathrm{NC}}$ by proving the correct execution of n_2 invocations of the step function F_{NC}.

The NC protocol requires an indexed Merkle tree called the **Output Inclusion Tree** OIT. It is initially empty and is later populated with the leaves $\mathbf{v}^{(2)}$ of Ex2's double spend tree $\mathsf{DST}_{n_2}^{\mathsf{Ex2}}$. Let OIT_{j-1} and OIT_j denote the states of the output inclusion tree before and after the jth step, respectively.

For $j = 1, 2, \ldots, n_2$, the **public input** $\mathbf{z}_{j-1}^{\mathrm{NC}}$ to F_{NC} in step j is

$$\mathbf{z}_{j-1}^{\mathrm{NC}} = \left[\mathsf{root}(\mathsf{DST}_{n_1}^{\mathsf{Ex1}}), \mathsf{root}(\mathsf{OIT}_{j-1})\right].$$

The **public output** $\mathbf{z}_{j}^{\mathrm{NC}}$ of F_{NC} in step j is

$$\mathbf{z}_{j}^{\mathrm{NC}} = \left[\mathsf{root}(\mathsf{DST}_{n_1}^{\mathsf{Ex1}}), \mathsf{root}(\mathsf{OIT}_{j})\right].$$

Note that $\mathbf{z}_{j-1}^{\mathrm{NC}}$ and $\mathbf{z}_{j}^{\mathrm{NC}}$ differ only in the last component.

The **private input** $\mathbf{w}_{j-1}^{\mathrm{NC}}$ to F_{NC} in step j is

$$\mathbf{w}_{j-1}^{\mathrm{NC}} = \begin{bmatrix} v_j^{(2)}, \\ \mathsf{NonMemberProof}\left(\mathsf{root}\left(\mathsf{DST}_{n_1}^{\mathsf{Ex1}}\right), v_j^{(2)}\right), \\ \mathsf{InsertProof}\left(\mathsf{root}(\mathsf{OIT}_{j-1}), \mathsf{root}(\mathsf{OIT}_{j}), v_j^{(2)}\right) \end{bmatrix},$$

where $v_j^{(2)}$ is the jth leaf in Ex2's double spend tree $\mathsf{DST}_{n_2}^{\mathsf{Ex2}}$.

For each $j = 1, 2, \ldots, n_2$, in step j function F_{NC} performs the following computations:

(1) Using $\mathsf{NonMemberProof}\left(\mathsf{root}\left(\mathsf{DST}_{n_1}^{\mathsf{Ex1}}\right), v_j^{(2)}\right)$, it verifies that the leaf $v_j^{(2)}$ is not present in $\mathsf{DST}_{n_1}^{\mathsf{Ex1}}$.

(2) Using $\mathsf{InsertProof}\left(\mathsf{root}(\mathsf{OIT}_{j-1}), \mathsf{root}(\mathsf{OIT}_{j}), v_j^{(2)}\right)$, it verifies that $\mathsf{root}(\mathsf{OIT}_j)$ is the root hash of the indexed Merkle tree obtained by inserting the leaf $v_j^{(2)}$ into an indexed Merkle tree with root hash equal to $\mathsf{root}(\mathsf{OIT}_{j-1})$.

(3) Using $\mathsf{root}(\mathsf{OIT}_j)$, it updates $\mathbf{z}_{j-1}^{\mathrm{NC}}$ and outputs $\mathbf{z}_{j}^{\mathrm{NC}}$.

After step n_2, the root hash of OIT_{n_2} is checked to be equal to the root hash of $\mathsf{DST}_{n_2}^{\mathsf{Ex2}}$. The equality of these root hashes ensures that every leaf in $\mathsf{DST}_{n_2}^{\mathsf{Ex2}}$ was considered in a step of the NC protocol, and consequently checked for non-membership in $\mathsf{DST}_{n_1}^{\mathsf{Ex1}}$.

8.3 Knowledge of IVC Witness Implies Knowledge of NC Protocol Witness

In this section, we argue that knowledge of a witness of the IVC version of the NC protocol implies the knowledge of a witness $\mathsf{wit}_{\mathrm{NC}}$ for the original NC instance.

Definition 8.3. Let $\mathsf{IMT}_{\emptyset}$ denote the empty indexed Merkle tree. An instance of the IVC version of the NC protocol is the tuple $\mathsf{inst}_{\text{NC-IVC}} = (\mathbf{z}_0^{\mathrm{NC}}, \mathbf{z}_{n_2}^{\mathrm{NC}})$, where

$$\mathbf{z}_0^{\mathrm{NC}} = \left[\mathsf{root}(\mathsf{DST}_{n_1}^{\mathsf{Ex1}}), \mathsf{root}(\mathsf{IMT}_{\emptyset})\right],$$
$$\mathbf{z}_{n_2}^{\mathrm{NC}} = \left[\mathsf{root}(\mathsf{DST}_{n_1}^{\mathsf{Ex1}}), \mathsf{root}(\mathsf{DST}_{n_2}^{\mathsf{Ex2}})\right].$$

Note that the root hash of the OIT in $\mathbf{z}_{n_2}^{\mathrm{NC}}$ is equal to the root hash of $\mathsf{DST}_{n_2}^{\mathsf{Ex2}}$.

Definition 8.4. A witness for the instance $\mathsf{inst}_{\text{NC-IVC}}$ is a vector of private inputs

$$\mathsf{wit}_{\text{NC-IVC}} = \left[\mathbf{w}_0^{\mathrm{NC}}, \mathbf{w}_1^{\mathrm{NC}}, \ldots, \mathbf{w}_{n-1}^{\mathrm{NC}}\right]$$

such that for all $i = 0, 1, \ldots, n-1$ we have

$$\mathbf{z}_{i+1}^{\mathrm{NC}} = F_{\mathrm{NC}}(\mathbf{z}_i^{\mathrm{NC}}, \mathbf{w}_i^{\mathrm{NC}}).$$

Suppose an exchange knows a witness $\mathsf{wit}_{\text{NC-IVC}}$ of length n. We assume that the Poseidon hash function is collision resistant. Since the final OIT root hash $\mathsf{root}(\mathsf{OIT}_n) = \mathsf{root}\left(\mathsf{DST}_{n_2}^{\mathsf{Ex2}}\right)$, every leaf in $\mathsf{DST}_{n_2}^{\mathsf{Ex2}}$ must have been inserted into OIT_n in one of the n steps. Thus n must be equal to n_2. Furthermore, the non-membership of every leaf of $\mathsf{DST}_{n_2}^{\mathsf{Ex2}}$ in $\mathsf{DST}_{n_1}^{\mathsf{Ex1}}$ must have been checked in one of the n_2 steps. Thus the exchange must know n_2 leaves which form the tree $\mathsf{DST}_{n_2}^{\mathsf{Ex2}}$ and n_2 non-membership proofs as defined in Definition 8.2.

8.4 Proof Generation and Verification

Prior to proof generation and verification, the exchange and the verifier use the structure of F_{NC} to generate the Nova proving key $\mathsf{pk}_{\mathrm{NC}}$ and verification key $\mathsf{vk}_{\mathrm{NC}}$, respectively.

8.4.1 Proof Generation. To generate an NC proof for an instance $\mathsf{inst}_{\mathrm{NC}}$ (see Definition 8.1), the pair of exchanges $\mathsf{Ex1}$ and $\mathsf{Ex2}$ need to collaborate. First, $\mathsf{Ex2}$ sends the leaves $\mathbf{v}^{(2)}$ corresponding to its double spend tree $\mathsf{DST}^{\mathsf{Ex2}}_{n_2}$ to $\mathsf{Ex1}$. Using $\mathbf{v}^{(2)}$, the $\mathsf{Ex1}$ constructs the private inputs $\mathbf{w}^{\mathrm{NC}}_{j-1}, j = 1, 2, \ldots, n_2$, as defined in Section 8.2.

Starting from the public input $\mathbf{z}^{\mathrm{NC}}_0$ given in Definition 8.3, $\mathsf{Ex1}$ computes the remaining $\mathbf{z}^{\mathrm{NC}}_j$ values using $\mathbf{z}^{\mathrm{NC}}_{j+1} = F_{\mathrm{NC}}(\mathbf{z}^{\mathrm{NC}}_j, \mathbf{w}^{\mathrm{NC}}_j)$ for $j = 0, 1, \ldots, n_2 - 1$. It then uses the Nova IVC scheme to generate an IVC proof attesting to the correct execution of the n_2 steps (see Section 5.1). Finally, it generates a zkSNARK proof π_{NC} proving knowledge of the IVC proof (see Section 5.2).

The tuple $(\mathsf{inst}_{\text{NC-IVC}}, \pi_{\mathrm{NC}}) = ((\mathbf{z}^{\mathrm{NC}}_0, \mathbf{z}^{\mathrm{NC}}_{n_2}), \pi_{\mathrm{NC}})$ is shared with the zkSNARK verifier $\mathcal{V}_{\mathrm{NC}}$.

We make two observations about NC proof generation.

(i) As an NC proof is for a pair of exchanges, to prove that a set of N exchanges have not colluded we need to generate $\binom{N}{2}$ NC proofs. These proofs can be generated in parallel as there is no dependence between them.

(ii) $\mathsf{Ex1}$ learns the leaf values $v^{(2)}_1, v^{(2)}_2, \ldots, v^{(2)}_{n_2}$ of $\mathsf{Ex2}$'s double spend tree $\mathsf{DST}^{\mathsf{Ex2}}_{n_2}$. Recall that $v^{(2)}_j = H_{pos}(x^{(2)}_j \| \mathsf{bh})$ where H_{pos} is the Poseidon hash function and $x^{(2)}_j$ is the private key of the jth output used by $\mathsf{Ex2}$ to generate its reserves. If we model the Poseidon hash function as a random oracle, the $v^{(2)}_j$s do not leak any information about the private keys. However, the *number of outputs* n_2 used by $\mathsf{Ex2}$ is revealed to $\mathsf{Ex1}$.

8.4.2 Proof Verification. The zkSNARK verifier $\mathcal{V}_{\mathrm{NC}}$ verifies the proof π_{NC} as

$$0/1 \leftarrow \mathcal{V}_{\mathrm{NC}}\left(\mathsf{vk}_{\mathrm{NC}}, \mathsf{inst}_{\text{NC-IVC}}, \pi_{\mathrm{NC}}\right).$$

Before verifying the proof, the verifier checks that the root hashes $\mathrm{root}(\mathsf{DST}^{\mathsf{Ex1}}_{n_1})$, $\mathrm{root}(\mathsf{DST}^{\mathsf{Ex2}}_{n_2})$ in $\mathsf{inst}_{\text{NC-IVC}}$ appeared in two valid RCG IVC protocol instances containing the same block height bh.

Note that the number of steps n_2 used by $\mathsf{Ex1}$ to obtain $\mathbf{z}^{\mathrm{NC}}_{n_2}$ is not revealed to the verifier. So the value of n_2 is only leaked to $\mathsf{Ex1}$ and not to NC proof verifiers (who could be customers). If the proof is accepted, the verifier is convinced that the two exchanges did not collude while generating their respective RCG protocol proofs at the same block height.

9 Security Analysis

In this section, we analyze the security of MProve-Nova. Let $\lambda \in \mathbb{N}$ be a security parameter. We model computationally bounded entities as probabilistic polynomial-time (PPT) algorithms with running times that are polynomial in λ.

Our security analysis uses the asymptotic approach. So we consider MProve-Nova as instantiated on a sequence of Monero-like systems $\{\mathcal{M}_\lambda \mid \lambda \in \mathbb{N}\}$, where the discrete logarithm problem in $\mathbb{G}$, the decisional Diffie-Hellman problem in $\mathbb{G}$, the problems of finding collisions or preimages of hash functions $H_{\mathrm{s}}, H_{\mathrm{p}}, H_{\mathrm{K}}, H_{pos}$, all become harder with increasing λ. The generation of $\mathcal{M}_\lambda$ for a given value of λ is described in Appendix G.

9.1 Inflation Resistance

We want to prove that MProve-Nova is an inflation-resistant PoR protocol. This entails proving that a valid RCG protocol proof π_{RCG} implies that the C_{res} in the corresponding RCG protocol instance $\mathsf{inst}_{\mathrm{RCG}}$ (see Definition 7.1) is a commitment to an amount of unspent coins actually owned by the exchange, except with a negligible probability. In other words, a valid RCG protocol proof must prevent an exchange from inflating the amount a_{res} committed by C_{res}.

From our discussion in Section 7.2, it is enough to prove that a valid π_{RCG} implies that the prover knows an RCG protocol witness $\mathsf{wit}_{\mathrm{RCG}}$ (see Definition 7.2), except with a negligible probability. Furthermore, since the knowledge of a witness $\mathsf{wit}_{\text{RCG-IVC}}$ for the IVC version of the RCG protocol implies knowledge of a witness $\mathsf{wit}_{\mathrm{RCG}}$ for the original RCG protocol (see Section 7.5), it is enough to prove that a valid π_{RCG} implies that the prover knows a witness $\mathsf{wit}_{\text{RCG-IVC}}$, except with a negligible probability. In other words, it is enough to prove that the RCG protocol satisfies *knowledge-soundness* (see the Definitions E.2, F.3 of IVC and zkSNARK knowledge-soundness for context).

Let $\mathcal{R}_{\text{RCG-IVC}}$ be the relation consisting of the instance-witness pairs $(\mathsf{inst}_{\text{RCG-IVC}}, \mathsf{wit}_{\text{RCG-IVC}})$ as given in Definitions 7.3, 7.4. From Section 7.6.2, recall that $\mathcal{V}_{\mathrm{RCG}}$ is the zkSNARK verifier used in the RCG protocol. Let $\mathcal{G}_{RCG}$ denote the PPT algorithm that generates the Nova public parameters (of both IVC scheme and zkSNARK) for the RCG protocol. Let $\mathcal{K}_{\mathrm{RCG}}$ be the deterministic algorithm that generates the Nova prover and verifier keys.

The below theorem says that if a PPT adversary can generate a valid zkSNARK proof π_{RCG} for some instance $\mathsf{inst}_{\text{RCG-IVC}}$, then a valid witness $\mathsf{wit}_{\text{RCG-IVC}}$ can be extracted from it, except with a negligible probability. Thus the C^{res}_n in $\mathsf{inst}_{\text{RCG-IVC}}$, which is equal to the C_{res} in $\mathsf{inst}_{\mathrm{RCG}}$ (see Section 7.3.2), is a Pedersen commitment to an amount of unspent coins owned by the exchange (see discussion in Section 7.2).

Theorem 9.1. *For any PPT adversary Ex, there is an expected polynomial-time extractor $\mathcal{E}_{RCG}$ such that for all randomness ρ, we have*

$$\Pr\left[\begin{array}{c|l} \mathcal{V}_{RCG}\begin{pmatrix} vk_{RCG}, \\ inst_{RCG\text{-}IVC}, \\ \pi_{RCG} \end{pmatrix} = 1, & pp \leftarrow \mathcal{G}_{RCG}(1^\lambda), \\ & \begin{pmatrix} inst_{RCG\text{-}IVC}, \\ \pi_{RCG} \end{pmatrix} \leftarrow Ex(pp; \rho), \\ \begin{pmatrix} inst_{RCG\text{-}IVC}, \\ wit_{RCG\text{-}IVC} \end{pmatrix} \notin \mathcal{R}_{RCG\text{-}IVC} & \begin{pmatrix} pk_{RCG}, \\ vk_{RCG} \end{pmatrix} \leftarrow \mathcal{K}_{RCG}(pp, F_{RCG}), \\ & wit_{RCG\text{-}IVC} \leftarrow \mathcal{E}_{RCG}\left(pp, \rho\right) \end{array}\right]$$

$$\leq negl(\lambda).$$

Proof. The proof is given in Appendix H. □

9.2 Collusion Resistance

We want to prove that MProve-Nova is a collusion-resistant PoR protocol. This entails proving that a valid NC protocol proof π_{NC}

implies that the two exchanges whose roots appear in the NC protocol instance $\mathsf{inst}_{\mathrm{NC}}$ (see Definition 8.1) did not use a common output while generating their reserves commitments, except with a negligible probability.

From our discussion in Section 8.1, it is enough to prove that a valid π_{NC} implies that the prover knows an NC protocol witness $\mathsf{wit}_{\mathrm{NC}}$ (see Definition 8.2), except with a negligible probability. Furthermore, since the knowledge of a witness $\mathsf{wit}_{\mathrm{NC\text{-}IVC}}$ for the IVC version of the NC protocol implies knowledge of a witness $\mathsf{wit}_{\mathrm{NC}}$ for the original NC protocol (see Section 8.3), it is enough to prove that a valid π_{NC} implies that the prover knows a witness $\mathsf{wit}_{\mathrm{NC\text{-}IVC}}$, except with a negligible probability. In other words, it is enough to prove that the NC protocol satisfies *knowledge-soundness.*

Let $\mathcal{R}_{\mathrm{NC\text{-}IVC}}$ be the relation consisting of the instance-witness pairs $(\mathsf{inst}_{\mathrm{NC\text{-}IVC}}, \mathsf{wit}_{\mathrm{NC\text{-}IVC}})$ as given in Definitions 8.3, 8.4. From Section 8.4.2, recall that $\mathcal{V}_{\mathrm{NC}}$ is the zkSNARK verifier used in the NC protocol. Let $\mathcal{G}_{NC}$ denote the PPT algorithm that generates the Nova public parameters (of both IVC scheme and zkSNARK) for the NC protocol. Let $\mathcal{K}_{\mathrm{NC}}$ be the deterministic algorithm that generates the Nova prover and verifier keys.

Theorem 9.2. *For any PPT adversary Ex, there is an expected polynomial-time extractor $\mathcal{E}_{NC}$ such that for all randomness ρ, we have*

$$\Pr\left[\begin{array}{c|l} \mathcal{V}_{NC}\begin{pmatrix} vk_{NC}, \\ inst_{NC\text{-}IVC}, \\ \pi_{NC}\end{pmatrix} = 1, & pp \leftarrow \mathcal{G}_{NC}(1^{\lambda}), \\ & \begin{pmatrix} inst_{NC\text{-}IVC}, \\ \pi_{NC}\end{pmatrix} \leftarrow Ex(pp;\rho), \\ \begin{pmatrix} inst_{NC\text{-}IVC}, \\ wit_{NC\text{-}IVC}\end{pmatrix} \notin \mathcal{R}_{NC\text{-}IVC} & \begin{pmatrix} pk_{NC}, \\ vk_{NC}\end{pmatrix} \leftarrow \mathcal{K}_{NC}(pp, F_{NC}), \\ & wit_{NC\text{-}IVC} \leftarrow \mathcal{E}_{NC}\,(pp,\rho) \end{array}\right]$$
$$\le negl(\lambda).$$

Proof. The proof is given in Appendix I. □

The above theorem says that if a PPT adversary can generate a valid zkSNARK proof π_{NC} for some instance $\mathsf{inst}_{\mathrm{NC\text{-}IVC}}$, then a valid witness $\mathsf{wit}_{\mathrm{NC\text{-}IVC}}$ can be extracted from it, except with a negligible probability. Thus, as long as the Poseidon hash function is collision resistant, the two exchanges whose DST roots appear in $\mathsf{inst}_{\mathrm{NC\text{-}IVC}}$ did not share any outputs while generating their respective reserves commitments (see discussion in Section 8.1).

9.3 Privacy

As discussed in Appendices B and C, MProve and MProve+ negatively impact the privacy of both Monero exchanges and regular Monero users. In this section, we show that MProve-Nova proofs only reveal the number of outputs used by an exchange in the NC proof generation. No other information about an exchange's outputs or their key images is revealed.

Recall from Sections 7.6.2 and 8.4.2 that the zkSNARK verifiers do not learn the number of outputs used by the exchanges. However, the NC proof generation requires $\mathsf{Ex2}$ to reveal its double spend tree leaves to $\mathsf{Ex1}$. If we model the Poseidon hash function H_{pos} as a random oracle, the leaves themselves do not reveal any information about the outputs used by $\mathsf{Ex2}$. However, the number of leaves equals the number of outputs used by $\mathsf{Ex2}$. While $\mathsf{Ex2}$ could request $\mathsf{Ex1}$ to keep the number of outputs a secret, we consider the worst-case scenario where $\mathsf{Ex1}$ itself uses this information to either identify $\mathsf{Ex2}$'s outputs or violate the untraceability, unlinkability, and amount confidentiality properties of Monero.

To the best of our knowledge, we are not aware of any attack which can use the number of outputs used by the exchange to affect the privacy of Monero exchanges or users in the real world. However, we cannot prove that no such attack exists because the number of outputs can affect privacy in extreme cases.

For example, consider the hypothetical scenario where at some block height bh all the N_{bh} RingCT outputs that have appeared on the Monero blockchain[5] are owned only by two exchanges $\mathsf{Ex1}$ and $\mathsf{Ex2}$. Suppose they own n_1 and n_2 outputs respectively, where $n_1 + n_2 = N_{\mathsf{bh}}$. During the NC protocol proof generation, if $\mathsf{Ex2}$ reveals the n_2 leaves of its double spend tree, then $\mathsf{Ex1}$ immediately knows that all the outputs it does not own belong to $\mathsf{Ex2}$. This hypothetical scenario is unlikely in practice due to output ownership by other Monero users and exchanges.

We claim that whatever a PPT adversary can infer from the Monero blockchain after observing a polynomial number of MProve-Nova proofs can also be inferred by a PPT adversary who only knows the number of outputs owned by the exchanges. To prove our claim, we need an algorithm that generates an instance of the Monero blockchain.

Recall that $\{\mathcal{M}_\lambda \mid \lambda \in \mathbb{N}\}$ is a sequence of Monero-like systems. Let $\mathcal{B}$ be a PPT algorithm which given $\mathcal{M}_\lambda$ and a block height bh, generates an instance B_{bh} of the Monero blockchain up to block height bh.

$$\mathsf{B}_{\mathsf{bh}} \leftarrow \mathcal{B}(\mathcal{M}_\lambda, \mathsf{bh}).$$

The instance B_{bh} will contain bh Monero blocks with each block containing some valid transactions.

We want to model an adversary that observes a polynomial number of MProve-Nova proofs on the blockchain instance B_{bh} and then attempts to violate the privacy of Monero users or the privacy of an exchange (by identifying its outputs). MProve-Nova proofs consist of RCG protocol proofs and NC protocol proofs. To simplify our analysis, we consider a stronger adversary $\mathcal{A}$ that observes the NC protocol witnesses instead of NC protocol proofs. Specifically, for every RCG protocol proof $\mathcal{A}$ is given the leaves $\mathbf{v}$ of the DST whose root appears in the RCG protocol instance. Given the DST leaves, $\mathcal{A}$ can itself generate the NC protocol proofs between any pair of exchanges (if their RCG protocol proofs are at the same block height and they have not colluded). Considering this stronger adversary simplifies our analysis because we don't need to specify which pairs of exchanges generated NC protocol proofs and which exchange revealed its DST leaves in a particular NC protocol proof. Any privacy property that holds against the adversary $\mathcal{A}$ will also hold against an adversary which observes NC protocol proofs instead of the DST leaves.

For Monero blockchain instance B_{bh}, suppose $\mathcal{A}$ observes a polynomial $x(\lambda)$ number of RCG protocol proofs. These proofs can be generated by different exchanges at various block heights where each height is at most bh. Let $h_1, h_2, \ldots, h_{x(\lambda)}$ be the heights at which these proofs were generated. These heights are not necessarily distinct. Let TXOT_{h_j} and KIT_{h_j} be the transaction outputs and

[5] As of May 2024, N_{bh} is more than 100 million.

key image trees at height h_j, respectively. Let n_j be the number of outputs used to generate the jth RCG protocol proof.

The jth RCG protocol proof consists of an instance-proof pair $\left(\mathsf{inst}^{(j)}_{\text{RCG-IVC}}, \pi^{(j)}_{\text{RCG}}\right)$ where according to Definition 7.3 the instance is given by $\mathsf{inst}^{(j)}_{\text{RCG-IVC}} = \left(\mathbf{z}^{(j)}_0, \mathbf{z}^{(j)}_{n_j}\right)$ where

$$\mathbf{z}^{(j)}_0 = \left[h_j, \mathsf{root}(\mathsf{TXOT}_{h_j}), \mathsf{root}(\mathsf{KIT}_{h_j}), \mathsf{root}(\mathsf{IMT}_\emptyset), G\right],$$
$$\mathbf{z}^{(j)}_{n_j} = \left[h_j, \mathsf{root}(\mathsf{TXOT}_{h_j}), \mathsf{root}(\mathsf{KIT}_{h_j}), \mathsf{root}(\mathsf{DST}_j), C^{\text{res}}_j\right].$$

Here DST_j and C^{res}_j denote the double spend tree and reserves commitment generated in the jth RCG protocol instance. And, $\pi^{(j)}_{\text{RCG}}$ denotes the jth zkSNARK proof that will be verified by the zkSNARK verifier $\mathcal{V}_{\text{RCG}}$ as described in Section 7.6.2.

For notational convenience, we will use RT_j to denote the jth instance-proof pair $\left(\mathsf{inst}^{(j)}_{\text{RCG-IVC}}, \pi^{(j)}_{\text{RCG}}\right)$, where RT abbreviates *RCG transcript*. Let $\mathbf{v}^{(j)}$ denote the DST leaves vector corresponding to the tree DST_j. As discussed earlier, for each RT_j observed by the adversary $\mathcal{A}$ the vector $\mathbf{v}^{(j)}$ will be revealed to it. Hence the adversary $\mathcal{A}$ observes the vector

$$\left[\left(\mathsf{RT}_1, \mathbf{v}^{(1)}\right), \left(\mathsf{RT}_2, \mathbf{v}^{(2)}\right), \ldots, \left(\mathsf{RT}_{x(\lambda)}, \mathbf{v}^{(x(\lambda))}\right)\right].$$

We have following theorem which states that whatever the adversary $\mathcal{A}$ can infer from the MProve-Nova proofs can also be inferred by an adversary $\mathcal{A}'$ which only knows the values of the reserves commitments, the DST leaves, and the block heights at which the proofs were generated. The symbol $\approx$ denotes computationally indistinguishable distributions (see Definition F.4).

Theorem 9.3. *For every PPT adversary $\mathcal{A}$, there exists a PPT adversary $\mathcal{A}'$ such that, for a blockchain instance B_{bh} we have*

$$\mathcal{A}\left(B_{bh}, (RT_1, \mathbf{v}^{(1)}), \ldots, (RT_{x(\lambda)}, \mathbf{v}^{(x(\lambda))})\right) \approx$$
$$\mathcal{A}'\left(B_{bh}, C^{res}_1, \ldots, C^{res}_{x(\lambda)}, \mathbf{v}^{(1)}, \ldots, \mathbf{v}^{(x(\lambda))}, h_1, \ldots, h_{x(\lambda)}\right)$$

where $x(\lambda)$ is a polynomial in the security parameter λ.

Proof. The proof is given in Appendix J. □

Corollary 9.4. *In the random oracle model, MProve-Nova proofs generated by non-colluding exchanges only reveal the number of outputs owned by an exchange to PPT adversaries.*

Proof. In Theorem 9.3, we showed that whatever can be inferred by a PPT adversary from the RCG protocol proofs and DST leaves can be inferred by a PPT adversary that only knows the reserves commitments, DST leaves, and the block heights at which the proofs were generated.

First, note that the Pedersen commitments C^{res}_j are uniformly distributed in $\mathbb{G}$ and independent of each other and the $\mathbf{v}^{(j)}$ vectors. So they don't reveal any information even to a computationally unbounded adversary, let alone a PPT adversary.

Recall that DST leaves vector $\mathbf{v}^{(j)}$ has components of the form $H_{pos}(x_i \| h_j)$ where H_{pos} is the Poseidon hash function, x_i is a private key, and h_j is a block height. A valid RCG protocol proof ensures that the x_is in a given $\mathbf{v}^{(j)}$ are distinct. If we model H_{pos} as a random oracle, then the components of $\mathbf{v}^{(j)}$ are independent and uniformly distributed in the field $\mathbb{F}_s$.

Suppose $\mathbf{v}^{(j)}$ and $\mathbf{v}^{(j')}$ correspond to DST leaves vectors at the same block height, i.e. $h_j = h_{j'}$. Under the assumption that the exchanges do not collude, the x_i values in $\mathbf{v}^{(j)}$ and $\mathbf{v}^{(j')}$ cannot overlap. In the random oracle model, the vectors $\mathbf{v}^{(j)}$ and $\mathbf{v}^{(j')}$ will then be independent.

If $\mathbf{v}^{(j)}$ and $\mathbf{v}^{(j')}$ correspond to DST leaves vectors at different block heights, i.e. $h_j \neq h_{j'}$. Then even if the x_i values in $\mathbf{v}^{(j)}$ and $\mathbf{v}^{(j')}$ overlap (which can happen if an exchange uses an output at both block heights), the vectors $\mathbf{v}^{(j)}$ and $\mathbf{v}^{(j')}$ will still be independent in the random oracle model.

Hence each $\mathbf{v}^{(j)}$ only reveals the number of leaves in the DST which is equal to the number of outputs owned by an exchange at height h_j. This proves the corollary. □

10 Implementation and Performance

The reference implementation of Nova [19] was not zero-knowledge [36] when we began this work. Angel *et al.* [2, 3] had implemented a zero-knowledge version of Nova by using hiding commitments and zero-knowledge sumcheck. Their code was based on an older version of the Nova implementation. We ported the relevant commits to the latest version of the Nova implementation. We used our modified implementation of Nova to implement both the subprotocols of MProve-Nova in Rust (source available on GitHub [47]). The implementation uses Pasta curves [61], a 2-cycle of elliptic curves, and the Poseidon hash function [23, 32].

To use Nova, the step function F must be expressed as R1CS constraints using the bellpepper library [33]. We implemented the following component gadgets in bellpepper (sources available on GitHub [48, 53, 54]).

(1) *Regular and indexed Merkle trees* [4, 49]: While implementations of regular Merkle trees already existed [62], our implementation of indexed Merkle trees is new.
(2) bellpepper-emulated: A gadget for non-native finite field arithmetic inspired by the emulated [18] package (written in Go) from the gnark zkSNARK library [10].
(3) bellpepper-ed25519: A gadget for ed25519 elliptic curve operations using bellperson-emulated.

Table 1 shows comparison between MProve-Nova RCG protocol, MProve+, and MProve for proof generation/verification. The simulations were run on a 64 core 2.30GHz Intel Xeon Gold 6314U CPU with access to 125GiB RAM. The open source implementations of MProve [5] and MProve+ [6] were used to perform the simulations.

For MProve and MProve+ the simulations were run for different sizes of the anonymity set $\mathcal{P}_{\text{anon}}$ up to 45,000 outputs, whereas for MProve-Nova RCG protocol the anonymity set is the set of all outputs on the Monero blockchain. Thus the comparison is unfair towards MProve-Nova PoR since for an anonymity set of all outputs, MProve and MProve+ would be impractical (see discussion in Sections B.4 and C.4). However, despite this, the MProve-Nova RCG protocol gives practical results and performs better in terms of verification times and proof sizes.

For the MProve-Nova RCG protocol, the proving time is linear in the number of exchange-owned outputs n, while the proof verification times and proof sizes are constant. The proving time per

Table 1: Performance comparison of protocols. n is the number of outputs owned by the exchange. PT denotes proving time, VT denotes verification time and PS denotes proof size with units in parentheses. Values with a * are estimated values due to simulation running out of memory.

n	*MProve-Nova RCG*			*MProve+*				*MProve*			
	PT (Hrs)	VT (s)	PS (KB)	$\lvert\mathcal{P}_{\text{anon}}\rvert$	PT (Hrs)	VT (s)	PS (KB)	$\lvert\mathcal{P}_{\text{anon}}\rvert$	PT (s)	VT (s)	PS (MB)
500	0.34	4.3	28.02	10,000	0.29	112.1	82.43	10,000	7.3	3.8	8.32
1,000	0.68	4.3	28.02	15,000	0.62	236.9	162.50	15,000	11.7	5.7	12.48
3,000	2.03	4.3	28.02	20,000	3.51*	1337.7*	400.90*	20,000	16.8	7.6	16.64
5,000	3.40	4.3	28.02	25,000	6.06*	2292.5*	723.23*	25,000	23.2	9.5	20.80
7,000	4.78	4.3	28.02	30,000	8.75*	3318.5*	1043.49*	30,000	30.4	11.4	24.96
10,000	6.94	4.3	28.02	35,000	22.49*	8820.1*	1523.88*	35,000	38.7	13.3	29.12
15,000	10.51	4.3	28.02	40,000	35.03*	13586.0*	2402.42*	40,000	47.3	15.2	33.28
20,000	14.00	4.3	28.02	45,000	47.98*	19624.1*	3205.06*	45,000	57.4	17.1	37.44

Table 2: Performance of MProve-Nova NC protocol. n_2 denotes the number of leaves in the double spend tree $\mathsf{DST}^{\mathsf{Ex2}}$ of exchange Ex2

n_2	PT (mins)	VT (s)	PS (KB)
500	2.39	0.2	23.70
1,000	4.75	0.2	23.70
3,000	14.14	0.2	23.70
5,000	23.50	0.2	23.70
7,000	33.08	0.2	23.70
10,000	47.11	0.2	23.70
15,000	70.62	0.2	23.70
20,000	93.88	0.2	23.70

10,000 owned outputs is about 7 hours. The verification time and proof size are constant at about 4.3 s and 28 KB, respectively.

The MProve-Nova RCG protocol has smaller proof sizes and faster verification times as compared to both MProve+ and MProve. MProve has faster proving times because we chose $|\mathcal{P}_{\text{anon}}| \leq 45,000$. If $|\mathcal{P}_{\text{anon}}|$ is increased to 100 million, MProve proofs would require 35 hours. While this seems reasonable, proof sizes would increase to 80 GB.

Table 2 shows the performance of the MProve-Nova NC protocol. In all cases, the proof size is 24 KB and proof verification times are about 200 ms. The proving time is linear in n_2, the number of outputs owned by Ex2, taking about 47 minutes per 10,000 outputs.

11 Conclusion

We described MProve-Nova, the first Monero PoR protocol that reveals no information about the exchange-owned outputs or their key images in the random oracle model, except their number. It is also the first Monero PoR protocol to achieve proof sizes and verification times that are independent of the number of outputs on the Monero blockchain. We compared MProve-Nova with MProve+ and MProve to show that our protocol has practical proving times and proof sizes. The proving times can be further reduced using non-uniform IVC schemes like SuperNova [26].

The MProve-Nova NC protocol reveals the number of outputs owned by the exchange which shares its double spend tree leaves with the other exchange. The construction of a non-collusion protocol using multi-party computation techniques in which the exchanges do not reveal any information to each other and generate a publicly-verifiable non-collusion proof is a possible direction for future work.

The MProve-Nova protocol assumes that the transactions output tree and key image tree roots have been constructed correctly from the Monero blockchain. While incorrect values for these tree roots can be detected by anyone with a copy of the Monero blockchain, it would require them to recompute the roots. Generating a SNARK proof for the validity of these tree roots is a challenging direction for future work.

Proving ownership of an output in Monero only requires a proof of knowledge of the corresponding private key. In Bitcoin, an output is specified by a challenge script and proving ownership of an output requires a proof of knowledge of a corresponding response script. The challenge script can take many possible forms making it difficult to prove knowledge of an output inside an R1CS circuit. This is the main obstacle that prevents us from applying our protocol to Bitcoin or similar UTXO-based cryptocurrencies.

Acknowledgments

We acknowledge the support of Trust Lab, IIT Bombay, for providing access to computational resources necessary for running the simulations. We thank Abhiram Kothapalli and Srinath Setty for their help in understanding the zero-knowledge property of Nova. We also express our gratitude to Manoj Prabhakaran, Sruthi Sekar, Chaya Ganesh and Chethan Kamath for insightful discussions.

References

[1] Zachary Amsden and et al. 2020. The Libra Blockchain. https://diem-developers-components.netlify.app/papers/the-diem-blockchain/2020-05-26.pdf

[2] Sebastian Angel. 2023. Nova Implementation with Zero-knowledge. https://github.com/sga001/Nova

[3] Sebastian Angel, Eleftherios Ioannidis, Elizabeth Margolin, Srinath Setty, and Jess Woods. 2023. Reef: Fast Succinct Non-Interactive Zero-Knowledge Regex Proofs. Cryptology ePrint Archive, Paper 2023/1886. https://eprint.iacr.org/2023/1886 https://eprint.iacr.org/2023/1886.

[4] Aztec. 2023. Indexed Merkle Tree. https://docs.aztec.network/aztec/concepts/storage/trees/indexed_merkle_tree

[5] Suyash Bagad. 2020. Implementation of MProve. https://github.com/suyash67/MProve-Ristretto

[6] Suyash Bagad. 2021. Implementation of MProve+. https://github.com/suyash67/MProvePlus-Ristretto

[7] Foteini Baldimtsi, Panagiotis Chatzigiannis, S. Gordon, Phi Le, and Daniel McVicker. 2022. gOTzilla: Efficient Disjunctive Zero-Knowledge Proofs from MPC in the Head, with Application to Proofs of Assets in Cryptocurrencies. *Proceedings on Privacy Enhancing Technologies* (2022), 229–249. https://doi.org/10.56553/popets-2022-0107

[8] Eli Ben-Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. 2014. Zerocash: Decentralized Anonymous Payments from Bitcoin. In *2014 IEEE Symposium on Security and Privacy (SP)*. IEEE, 459–474. https://doi.org/10.1109/SP.2014.36

[9] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. 2014. The making of KECCAK. *Cryptologia* 38, 1 (2014), 26–60. https://doi.org/10.1080/01611194.2013.856818

[10] Gautam Botrel, Thomas Piellard, Youssef El Housni, Ivo Kubjas, and Arya Tabaie. 2022. *ConsenSys/gnark: v0.7.0*. https://doi.org/10.5281/zenodo.5819104

[11] B. Bünz, J. Bootle, D. Boneh, A. Poelstra, P. Wuille, and G. Maxwell. 2018. Bulletproofs: Short Proofs for Confidential Transactions and More. In *2018 IEEE Symposium on Security and Privacy (IEEE S&P)*. 315–334. https://doi.org/10.1109/SP.2018.00020

[12] ChainSec. 2023. The Complete List of Crypto Exchange Hacks - ChainSec — chainsec.io. https://chainsec.io/exchange-hacks/

[13] Konstantinos Chalkias, Panagiotis Chatzigiannis, and Yan Ji. 2022. Broken Proofs of Solvency in Blockchain Custodial Wallets and Exchanges. Cryptology ePrint Archive, Paper 2022/043. https://eprint.iacr.org/2022/043

[14] Konstantinos Chalkias, Kevin Lewi, Payman Mohassel, and Valeria Nikolaenko. 2020. Distributed Auditing Proofs of Liabilities. Cryptology ePrint Archive, Paper 2020/468. https://eprint.iacr.org/2020/468

[15] Panagiotis Chatzigiannis, Foteini Baldimtsi, and Konstantinos Chalkias. 2021. SoK: Auditability and Accountability in Distributed Payment Systems. Cryptology ePrint Archive, Paper 2021/239. https://eprint.iacr.org/2021/239

[16] Panagiotis Chatzigiannis and Konstantinos Chalkias. 2021. Proof of Assets in the Diem Blockchain. In *Applied Cryptography and Network Security Workshops: ACNS 2021 Satellite Workshops, AIBlock, AIHWS, AIoTS, CIMSS, Cloud S&P, SCI, SecMT, and SiMLA, Kamakura, Japan, June 21–24, 2021, Proceedings* (Kamakura, Japan). Springer-Verlag, Berlin, Heidelberg, 27–41. https://doi.org/10.1007/978-3-030-81645-2_3

[17] Electric Coin Co. 2024. *Zcash Protocol Specification*. Technical Report. Electric Coin Co. https://zips.z.cash/protocol/protocol.pdf Overwinter+Sapling+Blossom+Heartwood+Canopy.

[18] Consensys. 2020. Implementation of gnark emulated package. https://github.com/Consensys/gnark/tree/master/std/math/emulated

[19] Nova Contributors. 2021. Nova Implementation. https://github.com/microsoft/Nova

[20] Gaby G. Dagher, Benedikt Bünz, Joseph Bonneau, Jeremy Clark, and Dan Boneh. 2015. Provisions: Privacy-Preserving Proofs of Solvency for Bitcoin Exchanges. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security* (Denver, Colorado, USA) *(CCS '15)*. Association for Computing Machinery, New York, NY, USA, 720–731. https://doi.org/10.1145/2810103.2813674

[21] Arijit Dutta, Suyash Bagad, and Saravanan Vijayakumaran. 2021. MProve+: Privacy Enhancing Proof of Reserves Protocol for Monero. *IEEE Transactions on Information Forensics and Security* 16 (2021), 3900–3915. https://doi.org/10.1109/TIFS.2021.3088035

[22] Arijit Dutta and Saravanan Vijayakumaran. 2019. MProve: A Proof of Reserves Protocol for Monero Exchanges. In *2019 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*. 330–339. https://doi.org/10.1109/EuroSPW.2019.00043

[23] Lorenzo Grassi, Dmitry Khovratovich, Christian Rechberger, Arnab Roy, and Markus Schofnegger. 2021. Poseidon: A New Hash Function for Zero-Knowledge Proof Systems. In *30th USENIX Security Symposium (USENIX Security 21)*. USENIX Association, 519–535. https://www.usenix.org/conference/usenixsecurity21/presentation/grassi

[24] Yan Ji and Konstantinos Chalkias. 2021. Generalized Proof of Liabilities. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security* (Virtual Event, Republic of Korea) *(CCS '21)*. Association for Computing Machinery, New York, NY, USA, 3465–3486. https://doi.org/10.1145/3460120.3484802

[25] Koe, Kurt M. Alonso, and Sarang Noether. 2020. Zero to Monero: Second Edition. https://www.getmonero.org/library/Zero-to-Monero-2-0-0.pdf

[26] Abhiram Kothapalli and Srinath Setty. 2022. SuperNova: Proving universal machine executions without universal circuits. Cryptology ePrint Archive, Paper 2022/1758. https://eprint.iacr.org/2022/1758

[27] Abhiram Kothapalli and Srinath Setty. 2023. HyperNova: Recursive arguments for customizable constraint systems. Cryptology ePrint Archive, Paper 2023/573. https://eprint.iacr.org/2023/573 https://eprint.iacr.org/2023/573.

[28] Abhiram Kothapalli and Srinath Setty. 2024. personal communication.

[29] Abhiram Kothapalli, Srinath Setty, and Ioanna Tzialla. 2021. Nova: Recursive Zero-Knowledge Arguments from Folding Schemes. Cryptology ePrint Archive, Paper 2021/370. https://eprint.iacr.org/2021/370.

[30] Abhiram Kothapalli, Srinath Setty, and Ioanna Tzialla. 2022. Nova: Recursive Zero-Knowledge Arguments from Folding Schemes. In *Advances in Cryptology – CRYPTO 2022: 42nd Annual International Cryptology Conference, CRYPTO 2022, Santa Barbara, CA, USA, August 15–18, 2022, Proceedings, Part IV* (Santa Barbara, CA, USA). Springer-Verlag, Berlin, Heidelberg, 359–388. https://doi.org/10.1007/978-3-031-15985-5_13

[31] Amrit Kumar, Clément Fischer, Shruti Tople, and Prateek Saxena. 2017. A Traceability Analysis of Monero's Blockchain. In *Computer Security – ESORICS 2017*, Simon N. Foley, Dieter Gollmann, and Einar Snekkenes (Eds.). Springer International Publishing, Cham, 153–173. https://doi.org/10.1007/978-3-319-66399-9_9

[32] Lurk Lab. 2020. neptune : Implementation of the Poseidon hash function. https://github.com/lurk-lab/neptune

[33] Lurk Lab. 2023. bellpepper : Rust Library for R1CS circuits. https://github.com/lurk-lab/bellpepper

[34] Russell W. F. Lai, Viktoria Ronge, Tim Ruffing, Dominique Schröder, Sri Aravinda Krishnan Thyagarajan, and Jiafan Wang. 2019. Omniring: Scaling Private Payments Without Trusted Setup. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security* (London, United Kingdom) *(CCS '19)*. Association for Computing Machinery, New York, NY, USA, 31–48. https://doi.org/10.1145/3319535.3345655

[35] Joseph K. Liu, Victor K. Wei, and Duncan S. Wong. 2004. Linkable Spontaneous Anonymous Group Signature for Ad Hoc Groups. In *Information Security and Privacy*, Huaxiong Wang, Josef Pieprzyk, and Vijay Varadharajan (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 325–335. https://doi.org/10.1007/978-3-540-27800-9_28

[36] Microsoft. 2023. Zero-knowledge implementation gap in Nova. https://github.com/microsoft/Nova/issues/174

[37] Monero. 2024. The Monero Project. https://www.getmonero.org/

[38] MoneroSchedule 2020. Monero Scheduled Software Upgrades. https://github.com/monero-project/monero/#scheduled-software-upgrades Last Accessed: August 13, 2023.

[39] Malte Möser, Kyle Soska, Ethan Heilman, Kevin Lee, Henry Heffan, Shashvat Srivastava, Kyle Hogan, Jason Hennessey, Andrew Miller, Arvind Narayanan, and Nicolas Christin. 2018. An Empirical Analysis of Traceability in the Monero Blockchain. *Proceedings on Privacy Enhancing Technologies* (2018), 143–163. https://doi.org/10.1515/popets-2018-0025

[40] Wilson Nguyen, Dan Boneh, and Srinath Setty. 2023. Revisiting the Nova Proof System on a Cycle of Curves. Cryptology ePrint Archive, Paper 2023/969. https://eprint.iacr.org/2023/969

[41] Shen Noether and Adam Mackenzie. 2016. Ring Confidential Transactions. *Ledger* 1 (12 2016), 1–18. https://doi.org/10.5195/LEDGER.2016.34

[42] Torben Pryds Pedersen. 1992. Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing. In *Advances in Cryptology — CRYPTO '91*. Springer, 129–140. https://doi.org/10.1007/3-540-46766-1_9

[43] Privacy and Scaling Explorations (PSE). 2024. Semaphore: Privacy-preserving signaling protocol. https://semaphore.pse.dev/. Accessed: 2024-12-12.

[44] Elements Project. 2018. Proof-of-Reserves tool for Bitcoin. https://github.com/ElementsProject/reserves

[45] Steven Roose. 2019. Standardizing Bitcoin Proof of Reserves. https://blog.blockstream.com/en-standardizing-bitcoin-proof-of-reserves/

[46] Stoffu Noether. 2018. Reserve Proof Pull Request. https://github.com/monero-project/monero/pull/3027

[47] Varun Thakore. 2024. Implementation of MProve-Nova. https://github.com/varunthakore/mprove-nova

[48] Varun Thakore. 2024. Vanilla and Indexed Merkle Trees. https://github.com/varunthakore/merkle-trees

[49] Ioanna Tzialla, Abhiram Kothapalli, Bryan Parno, and Srinath Setty. 2022. Transparency Dictionaries with Succinct Proofs of Correct Operation. *ISOC Conference on Network and Distributed System Security (NDSS)* (2022). https://doi.org/10.14722/ndss.2022.23143

[50] Paul Valiant. 2008. Incrementally Verifiable Computation or Proofs of Knowledge Imply Time/Space Efficiency. In *Proceedings of the 5th Conference on Theory of Cryptography* (New York, USA) *(TCC'08)*. Springer-Verlag, Berlin, Heidelberg, 1–18. https://doi.org/10.1007/978-3-540-78524-8_1

[51] Nicolas van Saberhagen. 2013. CryptoNote v 2.0. https://bytecoin.org/old/whitepaper.pdf

[52] Saravanan Vijayakumaran. 2023. Analysis of CryptoNote Transaction Graphs Using the Dulmage-Mendelsohn Decomposition. In *5th Conference on Advances in Financial Technologies (AFT 2023)*, Vol. 282. 28:1–28:22. https://doi.org/10.4230/LIPIcs.AFT.2023.28

[53] Saravanan Vijayakumaran. 2024. bellpepper-ed25519. https://github.com/lurk-lab/bellpepper-gadgets/tree/main/crates/ed25519

[54] Saravanan Vijayakumaran. 2024. bellpepper-emulated. https://github.com/lurk-lab/bellpepper-gadgets/tree/main/crates/emulated

[55] Dimaz Ankaa Wijaya, Joseph Liu, Ron Steinfeld, and Dongxi Liu. 2017. Monero transaction in block 1468439 spending 5 RingCT outputs. https://xmrchain.net/tx/8d4a0c7eccf92542eb5e1f09e72cc0d934b180b768bc95388d33051db83194bb

[56] Dimaz Ankaa Wijaya, Joseph Liu, Ron Steinfeld, and Dongxi Liu. 2018. Monero Ring Attack: Recreating Zero Mixin Transaction Effect. In *2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/ 12th IEEE International Conference O n Big Data Science And Engineering (TrustCom/BigDataSE)*. 1196–1201. https://doi.org/10.1109/TrustCom/BigDataSE.2018.00165

[57] Wikipedia. 2023. FTX — Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/wiki/FTX

[58] Wikipedia. 2023. Mt. Gox — Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/wiki/Mt._Gox

[59] Wikipedia contributors. 2024. Tornado Cash. https://en.wikipedia.org/wiki/Tornado_Cash. Accessed: 2024-12-12.

[60] Zuoxia Yu, Man Ho Au, Jiangshan Yu, Rupeng Yang, Qiuliang Xu, and Wang Fat Lau. 2019. New Empirical Traceability Analysis of CryptoNote-Style Blockchains. In *Financial Cryptography and Data Security*. 133–149. https://doi.org/10.1007/978-3-030-32101-7_9

[61] Zcash. 2020. Pasta curves. https://github.com/zcash/pasta

[62] zkcrypto. 2015. bellman : Rust Library for R1CS circuits. https://github.com/zkcrypto/bellman

A Overview of Monero

Monero [37] is the most popular instantiation of the CryptoNote protocol [51], with additional privacy and efficiency improvements. In Monero transactions, receiver identities are hidden using *one-time addresses*, sender identities are obfuscated using *linkable ring signatures*, and the number of coins being transferred is hidden using *Pedersen commitments* [42]. These techniques together achieve three design properties of Monero, namely *unlinkability*, *untraceability* and *amount confidentiality*.

A.1 One-Time Addresses and Unlinkability

Monero public keys are points in the prime order subgroup of the twisted Edwards elliptic curve ed25519 [25]. Let $\mathbb{G}$ denote this subgroup whose order is a 253-bit prime l. Monero private keys are integers in the set $\mathbb{Z}_l = \{0, 1, 2, ..., l-1\}$. For the base point $G \in \mathbb{G}$, the public key $P \in \mathbb{G}$ corresponding to a private key $x \in \mathbb{Z}_l$ is denoted by $P = xG$. We will use additive notation for scalar multiplication throughout this paper.

Suppose Alice wants to send some Monero coins to Bob.

(1) Bob shares a public key pair $(B_{vk}, B_{sk}) \in \mathbb{G}^2$ with Alice. The subscripts vk and sk are abbreviations of *view key* and *spend key*. Let $(b_{vk}, b_{sk}) \in \mathbb{Z}_l^2$ denote the corresponding private key pair.

(2) She signs a transaction transferring coins she owns to Bob. This transaction will contain a random point R and a *one-time address* P that will be controlled by Bob.

(3) Alice chooses a random scalar $r \in \mathbb{Z}_l$ and computes the random point R as rG.

(4) Alice creates the one-time address P as

$$P = H_s(rB_{vk} \| o_{index})G + B_{sk}$$

where $\|$ denotes concatenation, $H_s : \{0,1\}^* \mapsto \mathbb{Z}_l$ is a scalar-valued cryptographic hash function, and o_{index} is the index of the new *output* (defined in Section 2) in the transaction.[6] Note that the private key corresponding to P is known *only* to Bob as it equals

$$x = H_s(b_{vk}R \| o_{index}) + b_{sk},$$

where $B_{sk} = b_{sk}G$.

(5) Alice broadcasts the transaction containing (P, R), which will be eventually included in a Monero block by miners.

(6) Bob identifies transactions transferring coins to him as follows:
   (i) For every new Monero block, Bob reads the point pairs (P, R) in all the transactions.
   (ii) He computes the point $P' = H_s(b_{vk}R \| o_{index})G + B_{sk}$.
   (iii) If $P' = P$, Bob concludes that the transaction is sending coins to him.

(7) Bob adds P to the list of one-time addresses owned by him.

Note that Bob will always be able to identify transactions meant for him as $rB_{vk} = rb_{vk}G = b_{vk}R$. This is nothing but a Diffie-Hellman shared secret between public keys R and B_{vk}.

One-time addresses generated using Bob's public key pair cannot be linked to his key pair as long as the decisional Diffie-Hellman (DDH) problem remains hard in ed25519. This is called the *unlinkability* property of Monero i.e. given a one-time address P, a probabilistic polynomial time (PPT) adversary can identify the corresponding public key pair (B_{vk}, B_{sk}) with a probability which is only negligibly better than random guessing. In this way, Monero hides the receiver's identity in a transaction.

A.2 Linkable Ring Signatures and Untraceability

Monero uses linkable ring signatures [35, 41] to obfuscate sender identities, while preventing double spending. Given a list of public keys, a ring signature allows a signer to prove that he knows the private key of one public key from the list *without* revealing which one. A linkable ring signature allows an observer to link multiple ring signatures generated using the same private key.

Suppose Bob wants to spend the coins tied to a one-time address P he owns, i.e. he knows $x \in \mathbb{Z}_l$ such that $P = xG$. This one-time address is already present on the Monero blockchain. He proceeds as follows:

(1) For a protocol-specified ring size n, Bob randomly samples $n-1$ one-time addresses $P_1, P_2, \ldots, P_{n-1}$ (all distinct from P) from the blockchain. These are called *decoy* addresses.

(2) Bob signs the spending transaction using a linkable ring signature on the set of one-time addresses

$$\mathcal{R} = \{P_1, P_2, \ldots, P_{n-1}, P\}.$$

This set is sorted in chronological order (oldest address first) to prevent the ordering of the keys in $\mathcal{R}$ from leaking the identity of P. The set $\mathcal{R}$ is called the *transaction ring*.

(3) Bob includes the linkable ring signature in the transaction he broadcasts to the Monero P2P network.

Hiding the identity of the spending key opens up the possibility of double spending. To prevent this, the linkable ring signature contains an element of $\mathbb{G}$ called the *key image*, defined as $I = xH_p(P)$ where $H_p : \mathbb{G} \mapsto \mathbb{G}$ is a point-valued cryptographic hash function.

Two linkable ring signatures spending from the same one-time address will have identical key images. The Monero blockchain maintains the set $\mathcal{I}$ of key images that have appeared in past transactions. If the coins tied to a one-time address P have already been

[6] The output index is included to allow the creation of distinct one-time addresses from the same public key pair in the same transaction.

spent, then its key image I will already be in $\mathcal{I}$. Monero block miners will reject transactions whose linkable ring signatures have key images from $\mathcal{I}$.

At the same time, revealing the key image of a one-time address does not leak information about the latter as long as the DDH problem remains hard in ed25519. To see this, let $H_p(P) = yG$ for some unknown $y \in \mathbb{Z}_l$. Then $I = xH_p(P) = xyG$ is the Diffie-Hellman function of $P = xG$ and $H_p(P)$. If the DDH problem is assumed to be hard in ed25519, then given P and $H_p(P)$ a polynomial-time observer cannot distinguish between I and a uniformly chosen point from $\mathbb{G}$, except with a negligible probability.

Linkable ring signatures achieve the *untraceability* property of Monero while preventing double spending i.e. given a transaction ring $\mathcal{R}$, a PPT adversary can correctly identify the one-time address P in $\mathcal{R}$ that is actually being spent with a probability which is only negligibly better than random guessing.

A.3 Pedersen Commitments and Amount Confidentiality

In the original CryptoNote protocol specification, the number of coins tied to a one-time address was public. To create a ring signature spending from an address, the spender could only sample from other addresses containing the same amount.

To improve privacy, Monero introduced the use of Pedersen commitments [42] to hide the number of coins tied to a one-time address. The Pedersen commitment to an amount $a \in \{0, 1, 2, ..., 2^{64} - 1\}$ is given by

$$C(a, y) = aH + yG,$$

where $y \in \mathbb{Z}_l$ is a randomly chosen *blinding factor* and $H \in \mathbb{G}$ is a curve point whose discrete logarithm with respect to the base point G is unknown. Such commitments are perfectly hiding and computationally binding. Pedersen commitments achieve the *amount confidentiality* property of Monero by hiding the amount in a transaction.

For a transaction which transfers coins to be valid, the source address must have more coins than the sum of the transferred amount and the transaction fees. When the number of coins associated with addresses are hidden in Pedersen commitments, checking this condition is non-trivial.

Pedersen commitments are homomorphic in the following sense. If C_1 and C_2 are Pedersen commitments to amounts a_1, a_2 respectively, then $C_1 + C_2$ is a Pedersen commitment to the amount $a_1 + a_2$. The homomorphic property of Pedersen commitments is used in conjunction with *range proofs* to check the sum of input amounts in a transaction exceed the sum of the output amounts. A range proof proves that the amount committed to by a Pedersen commitment is in a given range like $\{0, 1, 2, ..., 2^{64} - 1\}$.

When Alice wants to transfer some of her coins to Bob, she creates a Pedersen commitment $C(a, y)$ in addition to the one-time address P whose private key is known to Bob. Bob needs to know a and y to verify the transaction and spend from P in the future.

To communicate a and y to Bob, Alice includes

$$a' = a \oplus H_K(H_K(rB_{vk}))$$
$$y' = y \oplus H_K(rB_{vk})$$

in the transaction, where $\oplus$ is bitwise XOR and H_K is the Keccak hash function. As the point R is contained in the transaction, Bob can use his private view key b_{vk} to recover a and y from a' and y' using the Diffie-Hellman shared secret $rB_{vk} = b_{vk}R$ as

$$a = a' \oplus H_K(H_K(b_{vk}R)),$$
$$y = y' \oplus H_K(b_{vk}R).$$

A consequence of this design is that Alice knows the opening of the commitment $C(a, y)$ associated with the one-time address P, even though she does not know the private key corresponding to P. On the other hand, knowledge of the private key of a one-time address P implies knowledge of an opening to the Pedersen commitment C associated with P.

B MProve

In this appendix, we describe the MProve [22] PoR protocol. Recall that Monero uses a cyclic elliptic curve group $\mathbb{G}$ whose order l is a 253-bit prime and base point is G. Let $\mathbb{Z}_l = \{0, 1, \ldots, l - 1\}$ denote the set of scalars (private keys).

Let $\mathcal{P}_{\text{all}}$ be the set of *all* one-time addresses corresponding to RingCT outputs that have appeared on the Monero blockchain. An exchange knows the private keys corresponding to a subset $\mathcal{P}_{\text{known}}$ of $\mathcal{P}_{\text{all}}$ which will be used to generate the proof of reserves. The exchange chooses a subset $\mathcal{P}_{\text{anon}}$ of $\mathcal{P}_{\text{all}}$ such that

$$\mathcal{P}_{\text{known}} \subset \mathcal{P}_{\text{anon}} \subset \mathcal{P}_{\text{all}}.$$

$\mathcal{P}_{\text{anon}}$ will be the *anonymity set* in which the exchange's addresses will be hidden.

Let $\mathcal{P}_{\text{anon}} = \{P_1, P_2, \ldots, P_N\}$. Let C_i be the Pedersen commitment corresponding to P_i. An MProve instance is given by the tuple $\mathsf{inst} = (\mathbf{P}, \mathbf{C}, C_{\text{assets}})$ where $\mathbf{P} = [P_1, P_2, \ldots, P_N] \in \mathbb{G}^N$, $\mathbf{C} = [C_1, C_2, \ldots, C_N] \in \mathbb{G}^N$, and $C_{\text{assets}} \in \mathbb{G}$. The point C_{assets} will be a Pedersen commitment to the exchange's reserves amount.

An MProve proof is given by a tuple $\pi = (\mathbf{C}', \mathbf{\Gamma}, \mathbf{\Sigma})$ where $\mathbf{C}' \in \mathbb{G}^N$ and $\mathbf{\Gamma}, \mathbf{\Sigma}$ are vectors of N signatures described below.

B.1 Proof Generation

Let $x_i \in \mathbb{Z}_l$ be the private key corresponding to P_i. The exchange knows x_i for each $P_i \in \mathcal{P}_{\text{known}}$. The exchange generates the MProve instance inst and proof π as follows.

(1) The exchange constructs $\mathbf{P}$ by arranging the elements of $\mathcal{P}_{\text{anon}}$ as a vector. It constructs $\mathbf{C}$ by arranging the corresponding Pedersen commitments as a vector.

(2) For each $P_i \in \mathcal{P}_{\text{anon}}$, the exchange randomly chooses z_i from $\mathbb{Z}_l$ and generates C_i' as

$$C_i' = \begin{cases} z_iG & \text{if } P_i \in \mathcal{P}_{\text{known}}, \\ z_iG + C_i & \text{if } P_i \notin \mathcal{P}_{\text{known}}. \end{cases}$$

It sets $\mathbf{C}' = \begin{bmatrix} C_1' & C_2' & \cdots & C_N' \end{bmatrix}$.

(3) The exchange calculates C_{assets} using the following equation.

$$C_{\text{assets}} = \sum_{i=1}^{N} (C_i - C_i')$$

(4) For each $i = 1, 2, \ldots, N$, z_i is the discrete logarithm of either C_i' or $C_i' - C_i$. The exchange uses z_i to generate a ring

signature γ_i verifiable by the pair of public keys $(C_i', C_i' - C_i)$. Each γ_i belongs to $\mathbb{Z}_l^3$ (see the MProve paper [22] for the details regarding the generation of γ_i). The exchange sets $\mathbf{\Gamma} = \begin{bmatrix} \gamma_1 & \gamma_2 & \cdots & \gamma_N \end{bmatrix}$.

(5) For each $i = 1, 2, \ldots, N$, the exchange knows the private key x_i of P_i if $P_i \in \mathcal{P}_{\text{known}}$ or the discrete logarithm of $C_i' - C_i$. It generates a linkable ring signature σ_i verifiable by the pair of public keys $(P_i, C_i' - C_i)$. Each σ_i belongs to $\mathbb{G} \times \mathbb{Z}_l^3$ and will contain the key image I_i of either P_i or $C_i' - C_i$. The exchange sets $\Sigma = \begin{bmatrix} \sigma_1 & \sigma_2 & \cdots & \sigma_N \end{bmatrix}$.

An MProve proof does not leak the members of $\mathcal{P}_{\text{known}}$ because of the following reasons. The ring signature γ_i does not reveal if it was generated using the discrete logarithm of C_i' or $C_i' - C_i$. As long as the decisional Diffie-Hellman (DDH) problem is hard in $\mathbb{G}$, a polynomial-time adversary who observes the linkable ring signature σ_i and the key image I_i cannot detect if it was generated using the discrete logarithm of P_i or $C_i' - C_i$ (see Section A.2). Hence the MProve proof does not reveal if $P_i \in \mathcal{P}_{\text{known}}$ or not. We will see later in this appendix that revealing the key image of P_i is a major drawback of MProve.

B.2 Proof Verification

Let $\mathcal{I}$ be the set of all key images which have appeared on the Monero blockchain. Given the instance $\mathsf{inst} = (\mathbf{P}, \mathbf{C}, C_{\text{assets}})$ and proof $\pi = (\mathbf{C}', \mathbf{\Gamma}, \Sigma)$, the MProve verifier checks the following:

(1) For each $i = 1, 2, \ldots, N$, the verifier checks that the P_i from $\mathbf{P}$ is an address on the Monero blockchain. If not, it rejects the proof.

(2) For each $i = 1, 2, \ldots, N$, the verifier reads the Pedersen commitment C_i corresponding to P_i from the Monero blockchain. It checks that

$$C_{\text{assets}} = \sum_{i=1}^{N} (C_i - C_i')$$

If the equation does not hold, it rejects the proof.

(3) For each $i = 1, 2, \ldots, N$, it checks that γ_i from $\mathbf{\Gamma}$ is a correct ring signature for the public keys $(C_i', C_i' - C_i)$. If signature verification fails, it rejects the proof.

(4) For each $i = 1, 2, \ldots, N$, it checks that σ_i from Σ is a correct linkable ring signature for the public keys $(P_i, C_i' - C_i)$. If signature verification fails, it rejects the proof.

(5) For each $i = 1, 2, \ldots, N$, it checks that the key image I_i revealed in σ_i does not belong to $\mathcal{I}$, i.e. it has not appeared on the Monero blockchain. Otherwise, it rejects the proof for using a spent output.

(6) For each $i = 1, 2, \ldots, N$, it checks that the key image I_i revealed in σ_i has not appeared in an MProve proof published by another exchange. If a key image repeats in the MProve proofs generated by two exchanges, then collusion is declared and the proof is rejected.

B.3 Implications of a Correct MProve Proof

The soundness of an MProve proof relies on the assumption that no polynomial-time adversary can forge a ring signature or a linkable ring signature on ed25519, except with a negligible probability. If an MProve proof passes all the verifier's checks, then either the following theorem holds or the exchange (prover) has managed to forge one of the signatures in $\mathbf{\Gamma}$ or Σ.

Theorem B.1 (MProve). *Let* $(\mathbf{P}, \mathbf{C}, C_{assets})$, $(\mathbf{C}', \mathbf{\Gamma}, \Sigma)$ *be an MProve instance-proof pair that passes all the verifier's checks. Let* $\mathcal{P}_{anon} = \{P_1, \ldots, P_N\}$. *Let the Pedersen commitment corresponding to* P_i *be* $C_i = a_i H + y_i G$, *with amount* a_i *and blinding factor* y_i. *Suppose that none of the signatures in* $\mathbf{\Gamma}$ *or* Σ *have been forged. Then there exists a subset of the anonymity set indices* $\mathcal{J} \subset \{1, 2, \ldots, N\}$ *such that the following conditions hold.*

(i) C_{assets} *is a Pedersen commitment to the amount*

$$\sum_{i \in \mathcal{J}} a_i.$$

(ii) If $i \in \mathcal{J}$ *and* $a_i \neq 0$, *then* $P_i \in \mathcal{P}_{known}$ *and the key image* I_i *revealed by the linkable ring signature* σ_i *is equal to the key image of* P_i. *In other words, the exchange knows the private key corresponding to* P_i *and will generate* σ_i *using this private key. Furthermore, this* P_i *is an unspent address.*

Proof. The outline of the proof is as follows. If the exchange does not know the private key corresponding to P_i, i.e. $P_i \notin P_{\text{known}}$, then it will be forced to generate the linkable ring signature σ_i using the discrete logarithm of $C_i' - C_i$. This means the $C_i - C_i'$ is a commitment to the zero amount. Hence it will not contribute an amount term to C_{assets}. Such indices i will not be included in the set $\mathcal{J}$.

Suppose the exchange knows the private key corresponding to P_i, i.e. $P_i \in P_{\text{known}}$. To generate the ring signature γ_i, the exchange needs the discrete logarithm of either C_i' or $C_i' - C_i$ with respect to the base point G.

- If the discrete logarithm of C_i' is used to generate γ_i, then C_i' is a commitment to the zero amount as it is of the form $z_i G$. If $a_i \neq 0$, then the discrete logarithm of $C_i' - C_i = z_i G - y_i G - a_i H$ with respect to G is not known, as the discrete logarithm of H with respect to G is unknown. So the exchange will be forced to generate the linkable ring signature σ_i using the private key corresponding to P_i, revealing its key image I_i. Furthermore, the $C_i - C_i'$ term in C_{assets} will contribute an amount equal to a_i. The indices i corresponding to this case will form the set $\mathcal{J}$.
- If the discrete logarithm of $C_i' - C_i$ is used to generate γ_i, then $C_i' - C_i$ is a commitment to the zero amount. Then irrespective of which discrete logarithm the exchange uses to generate the linkable ring signature σ_i, the $C_i - C_i'$ term in C_{assets} will contribute the zero amount. The indices i corresponding to this case will not be included in the set $\mathcal{J}$.

So all indices i with $P_i \in \mathcal{P}_{\text{known}}$ will not be included in $\mathcal{J}$. Only those indices i where $C_i - C_i'$ is a commitment to a non-zero amount will be included in $\mathcal{J}$. For such indices, we are ensured that $P_i \in \mathcal{P}_{\text{known}}$. As the verifier checks that the key image I_i revealed in σ_i has not appeared on the Monero blockchain, the address P_i is unspent for each $i \in \mathcal{J}$.

□

As $P_i \in \mathcal{P}_{\text{known}}$ for each $i \in \mathcal{J}$ with $a_i \neq 0$, all the amounts that contribute to $\sum_{i \in \mathcal{J}} a_i$ correspond to addresses owned by the exchange. This means that the exchange cannot inflate its reserves beyond what it actually owns.

B.4 Drawbacks

The main drawback of MProve is that the linkable ring signature σ_i reveals the key image I_i of P_i for $i \in \mathcal{J}$. This key image is used to prove that the corresponding P_i is unspent and to detect collusion between exchanges. When such a P_i is actually spent by the exchange in a Monero transaction, the key image I_i will reappear. An observer who has seen the MProve proof can immediately identify P_i as the spending key in the transaction ring. This violates the untraceability property of Monero and also identifies P_i as an address owned by the exchange in the previously observed MProve proof.

As proof sizes in MProve are proportional to $|\mathcal{P}_{\text{anon}}|$, it is not feasible for the exchange to set $\mathcal{P}_{\text{anon}} = \mathcal{P}_{\text{all}}$. As of May 2024, the number of RingCT addresses is more than 100 million. If $|\mathcal{P}_{\text{anon}}|$ is 100 million, the MProve proof generation time will be about 35 hours and proof size will exceed 80 GB. If $\mathcal{P}_{\text{anon}} \neq \mathcal{P}_{\text{all}}$, then an MProve proof reveals that the exchange-owned addresses belong to a strict subset of the set of all addresses.

C MProve+

In this appendix, we describe the MProve+ [21] PoR protocol. It relies on the Bulletproofs [11] protocol, a zero-knowledge argument of knowledge which only requires hardness of the discrete logarithm problem in an underlying cyclic group.

Given a vector of group elements $\mathbf{G} = \left[G_1, G_2, \ldots, G_N\right]$ and a vector of scalars $\mathbf{x} = \left[x_1, x_2, \ldots, x_N\right]$ of same length, let $\mathbf{x} \cdot \mathbf{G}$ denote the multi-scalar multiplication operation given by

$$\mathbf{x} \cdot \mathbf{G} = \sum_{i=1}^{N} x_i G_i.$$

As in the case of MProve, an exchange chooses an anonymity set $\mathcal{P}_{\text{anon}}$ such that

$$\mathcal{P}_{\text{known}} \subset \mathcal{P}_{\text{anon}} \subset \mathcal{P}_{\text{all}},$$

where $\mathcal{P}_{\text{all}}$ is the set of all one-time addresses corresponding to RingCT outputs that have appeared on the Monero blockchain and the exchange knows the private keys corresponding to the subset $\mathcal{P}_{\text{known}}$.

Let $\mathcal{P}_{\text{anon}} = \{P_1, P_2, \ldots, P_N\}$. Let C_i be the Pedersen commitment corresponding to P_i. Recall that $H_{\text{p}} : \mathbb{G} \mapsto \mathbb{G}$ is the point-valued cryptographic hash function used in the calculation of key images in Monero.

For an integer s satisfying $1 \leq s \leq N$, an MProve+ instance is a tuple $\mathsf{inst} = (\mathbf{P}, \mathbf{H}, \mathbf{C}, \mathbf{I}, C_{\text{assets}}) \in \mathbb{G}^{3N+s+1}$ where $\mathbf{P} = [P_1, P_2, \ldots, P_N] \in \mathbb{G}^N$, $\mathbf{H} = \left[H_{\text{p}}(P_1), H_{\text{p}}(P_2), \ldots, H_{\text{p}}(P_N)\right]$, $\mathbf{C} = [C_1, C_2, \ldots, C_N] \in \mathbb{G}^N$, $\mathbf{I} = [I_1, I_2, \ldots, I_s] \in \mathbb{G}^s$ and $C_{\text{assets}} \in \mathbb{G}$. The point C_{assets} will be a Pedersen commitment to the exchange's reserves amount. An MProve+ proof is a Bulletproofs proof π that proves that the MProve+ instance satisfies certain desirable properties (as explained in the following subsection).

C.1 Proof Generation

The exchange generates the MProve+ instance and proof as follows.

(1) The exchange constructs $\mathbf{P}$ by arranging the elements of $\mathcal{P}_{\text{anon}}$ as a vector. It constructs $\mathbf{C}$ by arranging the corresponding Pedersen commitments as a vector. It hashes the elements of $\mathbf{P}$ using H_{p} to construct vector $\mathbf{H}$.

(2) Let $s = |\mathcal{P}_{\text{known}}|$ and $\mathcal{P}_{\text{known}} = \{P_{i_1}, P_{i_2}, \ldots, P_{i_s}\}$. For each $P_{i_j} \in \mathcal{P}_{\text{known}}$, the exchange uses the corresponding private key x_j to calculate the key image $I_j = x_j H_{\text{p}}(P_{i_j})$. It arranges these key images into the vector $\mathbf{I} = [I_1, I_2, \ldots, I_s]$.

(3) The exchange chooses a scalar r uniformly from $\mathbb{Z}_l$ and sets $C_{\text{assets}} = rG + \sum_{j=1}^{s} C_{i_j}$ where C_{i_j} is the Pedersen commitment corresponding to P_{i_j}.

(4) The exchange generates a Bulletproofs proof π, which is a zero-knowledge argument of knowledge of a witness $\mathsf{wit} = (\mathbf{x}, \mathbf{e}_1, \mathbf{e}_2, \ldots, \mathbf{e}_s, \mathbf{a}, \mathbf{r}, a_{\text{res}}, r_{res}) \in \mathbb{Z}_l^{(N+3)s+2}$ that satisfies the following conditions.

- All the elements of wit are either scalars from $\mathbb{Z}_l$ or vectors whose components belong to $\mathbb{Z}_l$.
- Each $\mathbf{e}_j$ is a *unit vector* of length N, i.e. all its components except one are 0. The non-zero component has the value 1.
- $\mathbf{x}$ is a vector of length s. Let x_j denote the jth component of $\mathbf{x}$. Then for each $j \in \{1, 2, \ldots, s\}$, the following condition holds:

  $$\mathbf{e}_j \cdot \mathbf{P} = x_j G.$$

  In other words, the vector $\mathbf{x}$ contains the private keys of s addresses in $\mathcal{P}_{\text{anon}}$.
- $\mathbf{a}$ and $\mathbf{r}$ are both vectors of length s. Let a_j and r_j denote the jth components of $\mathbf{a}$ and $\mathbf{r}$, respectively. For each $j \in \{1, 2, \ldots, s\}$, the following condition holds:

  $$\mathbf{e}_j \cdot \mathbf{C} = r_j G + a_j H.$$

  In other words, the vectors $\mathbf{a}, \mathbf{r}$ contain the amounts and blinding factors of s commitments in $\mathbf{C}$. Furthermore, the locations of these commitments in $\mathbf{C}$ match with the locations of the addresses in $\mathbf{P}$ whose private keys are contained in $\mathbf{x}$.
- For each $j \in \{1, 2, \ldots, s\}$, the following condition holds:

  $$\mathbf{e}_j \cdot \mathbf{H} = \left(x_j^{-1}\right) I_j.$$

  Note that $\mathbf{e}_j \cdot \mathbf{P} = P_{i_j}$ and $\mathbf{e}_j \cdot \mathbf{H} = H_{\text{p}}(P_{i_j})$ for some $i_j \in \{1, 2, \ldots, N\}$ as $\mathbf{e}_j$ is a unit vector. As $\mathbf{e}_j \cdot \mathbf{P} = x_j G$, it follows that x_j is the private key corresponding to P_{i_j}. From $\mathbf{e}_j \cdot \mathbf{H} = \left(x_j^{-1}\right) I_j$, it follows that $I_j = x_j H_{\text{p}}(P_{i_j})$. In other words, I_j is the key image of an address P_{i_j} in $\mathcal{P}_{\text{anon}}$, whose private key is the jth component of $\mathbf{x}$.
- C_{assets} is equal to $r_{\text{res}} G + a_{\text{res}} H$ and $a_{\text{res}} = \sum_{j=1}^{s} a_j$. In other words, C_{assets} is a Pedersen commitment to the amount a_{res}. Furthermore, this amount is the sum of the amounts in the commitments corresponding to addresses in $\mathbf{P}$ whose private keys are contained in $\mathbf{x}$. The value of r_{res} will be $r + \sum_{j=1}^{s} r_j$.

C.2 Proof Verification

Let $\mathcal{I}$ be the set of all key images which have appeared on the Monero blockchain. Given the instance $\mathsf{inst} = (\mathbf{P}, \mathbf{H}, \mathbf{C}, \mathbf{I}, C_{\text{assets}})$ and Bulletproof proof π, the MProve+ verifier checks the following:

(1) For each $i = 1, 2, \ldots, N$, the verifier checks that the P_i from $\mathbf{P}$ is an address on the Monero blockchain. If not, it rejects the MProve+ proof.
(2) For each $i = 1, 2, \ldots, N$, the verifier checks that the ith components of $\mathbf{H}$ and $\mathbf{C}$ contain $H_\text{p}(P_i)$ and the Pedersen commitment C_i corresponding to P_i, respectively. If not, it rejects the proof.
(3) It checks that all the key images in $\mathbf{I}$ are distinct. If not, it rejects the proof for trying to use an output more than once. This check ensures that the unit vectors $\mathbf{e}_1, \mathbf{e}_2, \ldots, \mathbf{e}_s$ are all distinct.
(4) It uses the Bulletproofs verifier to check that π passes verification. If not, it rejects the proof.
(5) It checks that none of the key images in $\mathbf{I}$ appear in the set of key images $\mathcal{I}$. Otherwise, it rejects the proof for using a spent output.
(6) It checks that none of the key images $\mathbf{I}$ have appeared in an MProve+ proof published by another exchange. If a key image repeats in the MProve+ proofs generated by two exchanges, then collusion is declared and the proof is rejected.

C.3 Implications of a Correct MProve+ Proof

The soundness of an MProve+ proof relies on the assumption that no polynomial-time adversary can forge a Bulletproofs proof π in a group where the discrete logarithm problem is hard, except with a negligible probability. Since the Bulletproofs protocol is zero-knowledge in the random oracle model, the proof π does not reveal any information about the witness wit.

If the Bulletproofs proof π passes verification, then except with a negligible probability the exchange knows a witness wit which satisfies the properties mentioned in Section C.1. This implies the following.

(i) The exchange knows the private keys of s addresses in $\mathcal{P}_{\text{anon}}$. In other words, it owns s addresses in the anonymity set.
(ii) The identities of the s addresses owned by the exchange are not revealed, as they are part of the witness wit.
(iii) C_{assets} is a commitment to the sum of the amounts in the commitments corresponding to the s addresses owned by the exchange.
(iv) The vector $\mathbf{I}$ contains the key images of the s addresses owned by the exchange.

As long as the DDH problem is hard in $\mathbb{G}$, a polynomial-time adversary cannot use $\mathbf{I}$ to identify the s addresses owned by the exchange.

A correct MProve+ proof also requires that the key images in $\mathbf{I}$ are distinct and have not appeared in the set of all key images $\mathcal{I}$ or in another exchange's MProve+ proof. This ensures that the s addresses used to generate the PoR are used exactly once each, correspond to unspent outputs, and that two exchanges are not sharing addresses.

C.4 Drawbacks

Like MProve, the MProve+ protocol also requires the prover to reveal the key images of the addresses it uses to generate the PoR. While MProve establishes a direct relationship between a specific address P_i and its key image I_i, MProve+ only reveals that the key images in $\mathbf{I}$ correspond to some s addresses in $\mathcal{P}_{\text{anon}}$.

Suppose an address $P_i \in P_{\text{known}}$ is spent by the exchange after it was used in an MProve+ proof. The spending transaction will have a transaction ring $\mathcal{R}$ of addresses containing P_i and the key image I_i of P_i. An observer who has seen the MProve+ proof can identify that the transaction is being performed by the exchange. Since the address corresponding to I_i has to belong to both $\mathcal{R}$ and $\mathcal{P}_{\text{anon}}$, the observer can mark addresses which do not belong to $\mathcal{R} \cap \mathcal{P}_{\text{anon}}$ as decoy addresses in the transaction ring. If $|\mathcal{R} \cap \mathcal{P}_{\text{anon}}| < |\mathcal{R}|$, the untraceability property of Monero is violated by the MProve+ protocol, as the probability of correctly identifying the address being spent increases from $\frac{1}{|\mathcal{R}|}$ to $\frac{1}{|\mathcal{R} \cap \mathcal{P}_{\text{anon}}|}$. In the extreme case of $|\mathcal{R} \cap \mathcal{P}_{\text{anon}}| = 1$, the address P_i is identified as the address being spent in the transaction ring $\mathcal{R}$.

An exchange could mitigate the above issue by choosing the transaction rings $\mathcal{R}$ spending from addresses used in MProve+ proofs to always be subsets of $\mathcal{P}_{\text{anon}}$. But the fact that the exchange is the party performing the transaction is still revealed. This can negatively impact the exchange's privacy in some scenarios. For example, a flurry of transactions from an exchange could mean that the exchange is trying to sell its Monero reserves. This can cause buyers to offer lower prices.

While the Bulletproofs proof π has a size which is logarithmic in $|\mathcal{P}_{\text{anon}}| \times |\mathcal{P}_{\text{known}}|$, the MProve+ proof sizes increases linearly with $|\mathcal{P}_{\text{known}}|$ as one key image is revealed per exchange-owned address. Furthermore, the proof generation and verification times increase linearly [21] with $\mathcal{P}_{\text{anon}}$ making it infeasible to set $\mathcal{P}_{\text{anon}} = \mathcal{P}_{\text{all}}$. If $\mathcal{P}_{\text{anon}}$ contains 100 million addresses, then the proof verification time can take more than 10,000 hours and the proof generation time will be an order of magnitude larger. This implies that set $\mathcal{P}_{\text{anon}}$ can only be chosen to be a small subset of $\mathcal{P}_{\text{all}}$. So an MProve+ proof will reveal that the exchange-owned addresses belong to this small subset.

D Rank-1 Constraint Systems

Let $\mathbb{F}$ be a finite field. A rank-1 constraint system (R1CS) instance is a tuple $(\mathbb{F}, A, B, C, io, m, n)$ where

- A, B, C are $m \times m$ matrices with entries from the field $\mathbb{F}$ with at most $n = \Omega(m)$ non-zero entries.
- io is a vector with entries from $\mathbb{F}$ representing the public input and output of the instance, whose length satisfies $|io| + 1 \leq m$.

Definition D.1. An R1CS instance $(\mathbb{F}, A, B, C, io, m, n)$ is said to be *satisfiable* if there exists a witness $w \in \mathbb{F}^{m-|io|-1}$ such that

$$Az \circ Bz = Cz,$$

where $z = \begin{bmatrix} io & 1 & w \end{bmatrix}^T$ and $\circ$ is the Hadamard product operation.

The equation in the above definition encodes m R1CS constraints in the field $\mathbb{F}$. Each constraint is a quadratic expression in the entries of z.

Let $a_{i,j}, b_{i,j}, c_{i,j}$ denote the entries of the A, B, C matrices, respectively. Let z_j denote the jth entry of z. Then for $i = 1, 2, \dots, m$, the ith R1CS constraint is given by

$$\left(\sum_{j=1}^{m} a_{i,j} z_j\right)\left(\sum_{j=1}^{m} b_{i,j} z_j\right) = \sum_{j=1}^{m} c_{i,j} z_j.$$

A *relaxed* R1CS instance is a tuple $(\mathbb{F}, A, B, C, E, s, io, m, n)$ which has two more components when compared to an R1CS instance, a vector $E \in \mathbb{F}^m$ and a scalar $s \in \mathbb{F}$.

Definition D.2. A relaxed R1CS instance $(\mathbb{F}, A, B, C, E, s, io, m, n)$ is said to be *satisfiable* if there exists a witness $w \in \mathbb{F}^{m-|io|-1}$ such that

$$Az \circ Bz = sCz + E,$$

where $z = \begin{bmatrix} io & s & w \end{bmatrix}^T$ and $\circ$ is the Hadamard product operation.

Let Commit be an algorithm that given some public parameters pp generates commitments to vectors with entries from $\mathbb{F}$. Let $\overline{E} = \mathsf{Commit}(\mathsf{pp}, E)$ and $\overline{w} = \mathsf{Commit}(\mathsf{pp}, w)$ for $E \in \mathbb{F}^m$ and $w \in \mathbb{F}^{m-|io|-1}$.

A *committed relaxed* R1CS instance is a tuple

$$\left(\mathbb{F}, A, B, C, \overline{E}, \overline{w}, s, io, m, n\right)$$

which replaces E in a relaxed R1CS instance with the components $\overline{E}$ and $\overline{w}$.

Definition D.3. A committed relaxed R1CS instance given by $\left(\mathbb{F}, A, B, C, \overline{E}, \overline{w}, s, io, m, n\right)$ is said to be *satisfiable* if there exists an extended witness $(E, w) \in \mathbb{F}^m \times \mathbb{F}^{m-|io|-1}$ such that

$$\begin{aligned} \overline{E} &= \mathsf{Commit}(\mathsf{pp}, E) \\ \overline{w} &= \mathsf{Commit}(\mathsf{pp}, w) \\ Az \circ Bz &= sCz + E, \end{aligned}$$

where $z = \begin{bmatrix} io & s & w \end{bmatrix}^T$ and $\circ$ is the Hadamard product operation.

E IVC Scheme Definitions

Let F be a polynomial-time computable function. Starting from an input z_0, F is used to compute output z_n using n invocations of the form

$$z_{i+1} = F(z_i, w_i),$$

for $i = 0, 1, \dots, n-1$, where w_i is an auxiliary input to the ith step.

An IVC scheme allows a prover to generate a proof Π_{i+1} for the statement $z_{i+1} = F(z_i, w_i)$ given a proof Π_i for the statement $z_i = F(z_{i-1}, w_{i-1})$. It is defined by a tuple of PPT algorithms $(\mathcal{G}, \mathcal{P}, \mathcal{V}, \mathcal{K})$ where $\mathcal{G}$ is the public parameters generator, $\mathcal{P}$ is the IVC prover, $\mathcal{V}$ is the IVC verifier, and $\mathcal{K}$ generates the prover and verifier keys.

Let λ be a security parameter. Let PPT and EPT denote probabilistic polynomial-time and expected polynomial-time, respectively. The completeness and knowledge-soundness of IVC scheme are defined as follows [30].

Definition E.1 (Perfect Completeness). An IVC scheme $(\mathcal{G}, \mathcal{P}, \mathcal{V}, \mathcal{K})$ satisfies *perfect completeness* if for any PPT adversary $\mathcal{A}$, we have

$$\Pr\left[\mathcal{V}\begin{pmatrix}\mathsf{vk}, z_0, \\ z_{i+1}, \Pi_{i+1}\end{pmatrix} = 1 \;\middle|\; \begin{array}{l} \mathsf{pp} \leftarrow \mathcal{G}(1^\lambda), \\ F, (z_0, z_{i+1}, z_i, w_i, \Pi_i) \leftarrow \mathcal{A}(\mathsf{pp}), \\ (\mathsf{pk}, \mathsf{vk}) \leftarrow \mathcal{K}(\mathsf{pp}, F), \\ z_{i+1} = F(z_i, w_i), \\ \mathcal{V}\,(\mathsf{vk}, z_0, z_i, \Pi_i) = 1, \\ \Pi_{i+1} \leftarrow \mathcal{P}\,(\mathsf{pk}, z_0, z_{i+1}; z_i, w_i, \Pi_i) \end{array}\right] = 1,$$

where F is a polynomial-time computable function, pp are public parameters, pk is a prover key, and vk is a verifier key.

In other words, if the verifier of an IVC scheme that satisfies perfect completeness accepts an IVC proof Π_i for initial input z_0 and final output z_i, then the prover can generate an IVC proof Π_{i+1} for initial input z_0 and final output z_{i+1} which will always be accepted by the verifier. Here $z_{i+1} = F(z_i, w_i)$ for some auxiliary input w_i.

Definition E.2 (Knowledge-Soundness). An IVC scheme $(\mathcal{G}, \mathcal{P}, \mathcal{V}, \mathcal{K})$ satisfies *knowledge-soundness* if for any constant $n \in \mathbb{N}$ and EPT adversary $\mathcal{P}^*$, there exists an EPT extractor $\mathcal{E}$ such that for any input randomness ρ

$$\Pr\left[\begin{array}{c} z_n \neq z, \\ \mathcal{V}\begin{pmatrix}\mathsf{vk}, z_0, \\ z, \Pi\end{pmatrix} = 1 \end{array} \;\middle|\; \begin{array}{l} \mathsf{pp} \leftarrow \mathcal{G}(1^\lambda), \\ F, (z_0, z, \Pi) \leftarrow \mathcal{P}^*(\mathsf{pp}; \rho), \\ (\mathsf{pk}, \mathsf{vk}) \leftarrow \mathcal{K}(\mathsf{pp}, F), \\ (w_0, \dots, w_{n-1}) \leftarrow \mathcal{E}(\mathsf{pp}, z_0, z; \rho), \\ z_i \leftarrow F(z_{i-1}, w_{i-1}), \forall i \in \{1, 2, \dots, n\} \end{array}\right]$$
$$\leq \mathsf{negl}(\lambda).$$

In other words, if the verifier of an IVC scheme that satisfies knowledge-soundness accepts an IVC proof Π for initial input z_0 and final output z, then the prover knows auxiliary inputs $(w_0, w_1, \dots, w_{n-1})$ such that $z = z_n$ except with a negligible probability where z_n is the end result of the sequence of computations $z_i = F(z_{i-1}, w_{i-1})$ for $i = 1, 2, \dots, n$.

The Nova IVC scheme satisfies perfect completeness and knowledge-soundness [30].

F zkSNARK Definitions

In this section, we present the definition of a zero-knowledge succinct non-interactive argument of knowledge (zkSNARK) using notation from the Nova and HyperNova papers [27, 30]. Let $\lambda \in \mathbb{N}$ denote a security parameter. Let PPT and EPT denote probabilistic polynomial-time and expected polynomial-time, respectively.

Definition F.1 (Non-interactive Argument of Knowledge). Let $\mathcal{R}$ be a relation over public parameters pp, structure s, instance u, and witness w tuples. A non-interactive argument of knowledge for $\mathcal{R}$ consists of PPT algorithms $(\mathcal{G}, \mathcal{P}, \mathcal{V})$ and deterministic $\mathcal{K}$, denoting the generator, the prover, the verifier, and the encoder, respectively, with the following interface:

- $\mathsf{pp} \leftarrow \mathcal{G}(1^\lambda)$: On input λ, $\mathcal{G}$ samples public parameters pp.
- $(\mathsf{pk}, \mathsf{vk}) \leftarrow \mathcal{K}(\mathsf{pp}, \mathsf{s})$: On input s, representing common structure among instances, $\mathcal{K}$ outputs prover key pk and verifier key vk.

- $\pi \leftarrow \mathcal{P}\,(\mathsf{pk}, u, w)$: On input instance u and witness w, $\mathcal{P}$ outputs a proof π proving that $(\mathsf{pp}, \mathsf{s}, u, w) \in \mathcal{R}$.
- $1/0 \leftarrow \mathcal{V}\,(\mathsf{vk}, u, \pi)$: On input instance u and proof π, $\mathcal{V}$ verifies proof π for instance u. It outputs 1 if the proof verification succeeds and 0 otherwise.

Definition F.2 (Perfect Completeness). A non-interactive argument of knowledge $(\mathcal{G}, \mathcal{P}, \mathcal{V}, \mathcal{K})$ for relation $\mathcal{R}$ satisfies perfect completeness if for any PPT adversary $\mathcal{A}$ we have

$$\Pr\left[\mathcal{V}\left(\mathsf{vk}, u, \pi\right) = 1 \;\middle|\; \begin{array}{l} \mathsf{pp} \leftarrow \mathcal{G}(1^{\lambda}),\\ (\mathsf{s}, (u, w)) \leftarrow \mathcal{A}(\mathsf{pp}),\\ (\mathsf{pp}, \mathsf{s}, u, w) \in \mathcal{R},\\ (\mathsf{pk}, \mathsf{vk}) \leftarrow \mathcal{K}(\mathsf{pp}, \mathsf{s}),\\ \pi \leftarrow \mathcal{P}\,(\mathsf{pk}, u, w) \end{array}\right] = 1.$$

In other words, if a non-interactive argument of knowledge for a relation $\mathcal{R}$ satisfies perfect completeness, then for every instance u in $\mathcal{R}$ the prover $\mathcal{P}$ can use the witness w to generate a proof π that will always be accepted by the verifier $\mathcal{V}$.

Definition F.3 (Knowledge-Soundness). A non-interactive argument of knowledge $(\mathcal{G}, \mathcal{P}, \mathcal{V}, \mathcal{K})$ for relation $\mathcal{R}$ satisfies knowledge-soundness if for all EPT adversaries $\mathcal{A}$ there exists an EPT extractor $\mathcal{E}$ such that for all randomness ρ we have

$$\Pr\left[\begin{array}{c} \mathcal{V}\left(\mathsf{vk}, u, \pi\right) = 1,\\ (\mathsf{pp}, \mathsf{s}, u, w) \notin \mathcal{R} \end{array} \;\middle|\; \begin{array}{l} \mathsf{pp} \leftarrow \mathcal{G}(1^{\lambda}),\\ (\mathsf{s}, u, \pi) \leftarrow \mathcal{A}(\mathsf{pp}; \rho),\\ (\mathsf{pk}, \mathsf{vk}) \leftarrow \mathcal{K}(\mathsf{pp}, \mathsf{s}),\\ w \leftarrow \mathcal{E}\,(\mathsf{pp}, \rho) \end{array}\right] \le \mathsf{negl}(\lambda).$$

In other words, if a non-interactive argument of knowledge for a relation $\mathcal{R}$ satisfies knowledge-soundness, then if an adversary $\mathcal{A}$ can generate a valid proof π for an instance u then it must know a witness w such that (u, w) is a valid instance-witness pair in $\mathcal{R}$.

To define the notion of zero-knowledge, we need to first define computational indistinguishability.

Definition F.4 (Computational Indistinguishability). Let X_λ and Y_λ be two sequences of distributions ranging over $\{0,1\}^{p(\lambda)}$ for a polynomial p. We say that X_λ and Y_λ are computationally indistinguishable, denoted by $X_\lambda \approx Y_\lambda$, if for any PPT adversary $\mathcal{A}$ we have

$$\left| \Pr_{x \leftarrow X_\lambda}\left[\mathcal{A}(x) = 1\right] - \Pr_{y \leftarrow Y_\lambda}\left[\mathcal{A}(y) = 1\right] \right| \le \mathsf{negl}(\lambda).$$

We adapt the definition of zero-knowledge for an interactive argument of knowledge given in the HyperNova paper [27, Definition 26] to the non-interactive setting to obtain the following definition. The Nova zkSNARK satisfies this definition of zero-knowledge [28].

Definition F.5 (Zero-Knowledge). A non-interactive argument of knowledge $(\mathcal{G}, \mathcal{P}, \mathcal{V}, \mathcal{K})$ for relation $\mathcal{R}$ satisfies zero-knowledge if there exists an EPT simulator $\mathcal{S}$ such that for any PPT adversary $\mathcal{A}$ we have

$$\left\{ (\mathsf{pp}, \mathsf{s}, u, \pi, \mathsf{st}) \;\middle|\; \begin{array}{l} \mathsf{pp} \leftarrow \mathcal{G}(1^{\lambda}),\\ (\mathsf{s}, (u, w), \mathsf{st}) \leftarrow \mathcal{A}(\mathsf{pp}),\\ (\mathsf{pp}, \mathsf{s}, u, w) \in \mathcal{R},\\ (\mathsf{pk}, \mathsf{vk}) \leftarrow \mathcal{K}(\mathsf{pp}, \mathsf{s}),\\ \pi \leftarrow \mathcal{P}\,(\mathsf{pk}, u, w) \end{array} \right\}$$

$$\approx \left\{ (\mathsf{pp}, \mathsf{s}, u, \pi, \mathsf{st}) \;\middle|\; \begin{array}{l} \mathsf{pp} \leftarrow \mathcal{G}(1^{\lambda}),\\ (\mathsf{s}, (u, w), \mathsf{st}) \leftarrow \mathcal{A}(\mathsf{pp}),\\ (\mathsf{pp}, \mathsf{s}, u, w) \in \mathcal{R},\\ \pi \leftarrow \mathcal{S}\,(\mathsf{pp}, \mathsf{s}, u, \mathsf{st}) \end{array} \right\}.$$

Here st is any auxiliary input available to the verifier.

In other words, if a non-interactive argument of knowledge for a relation $\mathcal{R}$ satisfies zero-knowledge, then for any PPT adversary $\mathcal{A}$ that generates an instance-witness pair (u, w) in $\mathcal{R}$ and auxiliary information st there exists an EPT simulator $\mathcal{S}$ that can generate a simulated proof π^{sim} using only the public parameters pp, structure s, instance u and auxiliary information st such that the joint distributions of $(\mathsf{pp}, \mathsf{s}, u, \pi^{\mathrm{act}}, \mathsf{st})$ and $(\mathsf{pp}, \mathsf{s}, u, \pi^{\mathrm{sim}}, \mathsf{st})$ are computationally indistinguishable, where π^{act} is the actual proof generated by an honest prover.

Definition F.6 (Succinctness). A non-interactive argument of knowledge $(\mathcal{G}, \mathcal{P}, \mathcal{V}, \mathcal{K})$ for relation $\mathcal{R}$ is succinct if the size of the proof π and verifier running time are at most polylogarithmic in the size of the structure s and witness w.

G Generating a Sequence of Monero-like Systems

Monero uses the ed25519 curve, which means that a real-world deployment of MProve-Nova will use this curve. But our analysis of MProve-Nova's security takes an asymptotic approach where our claims are expressed as functions of a security parameter $\lambda \in \mathbb{N}$. For the purpose of the security analysis, we will assume that MProve-Nova is being used in a sequence of Monero-like systems $\{\mathcal{M}_\lambda \mid \lambda \in \mathbb{N}\}$, where the discrete logarithm problem in $\mathbb{G}$, the decisional Diffie-Hellman problem in $\mathbb{G}$, the problems of finding collisions or preimages of hash functions $H_{\mathrm{s}}, H_{\mathrm{p}}, H_{\mathrm{K}}, H_{pos}$, all become harder with increasing λ. In this appendix, we describe how $\mathcal{M}_\lambda$ can be instantiated for a given λ.

Let GroupGen be a PPT algorithm that on input 1^λ generates a triple $(\mathbb{G}, l, G)$ where $\mathbb{G}$ is an elliptic curve of prime order l with generator G and the bit length of l satisfies $|l| = \lambda$.

Definition G.1. The *discrete logarithm problem is hard* relative to GroupGen if for all PPT adversaries $\mathcal{A}$ there exists a negligible function negl such that

$$\Pr\left[G' = xG \;\middle|\; \begin{array}{l} (\mathbb{G}, l, G) \leftarrow \mathsf{GroupGen}(1^{\lambda}),\\ G' \text{ is chosen uniformly from } \mathbb{G},\\ x \leftarrow \mathcal{A}(\mathbb{G}, l, G, G'), x \in \mathbb{Z}_l. \end{array}\right] \le \mathsf{negl}(\lambda).$$

Definition G.2. The *decisional Diffie-Hellman (DDH) problem is hard* relative to GroupGen if for all PPT adversaries $\mathcal{A}$ there exists a negligible function negl such that

$$\left| \Pr\left[\mathcal{A}\left(\mathbb{G}, l, G, xG, yG, zG\right) = 1\right] - \Pr\left[\mathcal{A}\left(\mathbb{G}, l, G, xG, yG, xyG\right) = 1\right] \right| \leq \mathsf{negl}(\lambda),$$

where $(\mathbb{G}, l, G) \leftarrow \mathsf{GroupGen}(1^\lambda)$ and x, y, z are chosen uniformly and independently from $\mathbb{Z}_l$.

We assume that the existence of an algorithm GroupGen which satisfies the two definitions above. To instantiate $\mathcal{M}_\lambda$, we run the GroupGen algorithm first to obtain the elliptic curve group $\mathbb{G}$.

Let $\mathbb{E}_1$ be a prime order elliptic curve whose points have coordinates in a prime field $\mathbb{F}_q$. Let the p be the prime which equals $|\mathbb{E}_1|$. Then $\mathbb{F}_p$ is called the *scalar field* of $\mathbb{E}_1$ and $\mathbb{F}_q$ is called the *base field* of $\mathbb{E}_1$. If there exists another elliptic curve $\mathbb{E}_2$ with scalar field $\mathbb{F}_q$ and base field $\mathbb{F}_p$, then $\mathbb{E}_1$ and $\mathbb{E}_2$ are said to form a *curve cycle*. The implementation of Nova requires a cycle of elliptic curves.

Let CurveCycleGen be a PPT algorithm that on input 1^λ generates a tuple $(\mathbb{E}_1, \mathbb{E}_2, G_1, G_2, p, q)$ where $\mathbb{E}_1$ and $\mathbb{E}_2$ form a curve cycle and have orders p and q respectively. Furthermore, G_1 and G_2 are generators of $\mathbb{E}_1$ and $\mathbb{E}_2$ respectively. The CurveCycleGen algorithm can interpreted as two instances of GroupGen, since it generates two prime order groups. We assume that the discrete logarithm problem is hard relative to both instances of GroupGen embedded in CurveCycleGen. For a given value of λ, we run the CurveCycleGen algorithm to obtain a pair of elliptic curves $\mathbb{E}_1, \mathbb{E}_2$.

A Monero-like system needs a hash function $H_\mathrm{s} : \{0,1\}^* \mapsto \mathbb{Z}_l$ to generate one-time addresses, a hash function $H_\mathrm{p} : \mathbb{G} \mapsto \mathbb{G}$ to generate key images, and a hash function $H_\mathrm{K} : \{0,1\}^* \mapsto \{0,1\}^{|l|}$ to blind the scalars of an output Pedersen commitment. Additionally, the MProve-Nova protocol requires a hash function $H_{pos} : \{0,1\}^* \mapsto \mathbb{F}_p$, where F_p is the scalar field of the elliptic curve $\mathbb{E}_1$. All four of these hash functions need to be collision resistant.

We use the notion of keyed hash functions to define the collision resistance of these hash functions. Let HashGen be a PPT algorithm that on input 1^λ generates a pair of PPT algorithms (Gen, H). The algorithm Gen takes 1^λ as input and generates a key $t \in \{0,1\}^{f(\lambda)}$ where f is a polynomial. The algorithm H takes as inputs a key t and a bitstring $x \in \{0,1\}^*$. It outputs a string $H^t(x) \in \{0,1\}^{l(\lambda)}$ where l is a polynomial. We require $H^t(\cdot)$ to be collision resistant even if an adversary knows t.

Definition G.3. A hash function $\Pi = (\mathsf{Gen}, H)$ is *collision resistant* if for all PPT adversaries $\mathcal{A}$ there is a negligible function negl such that

$$\Pr\left[H^t(x) = H^t(x') \;\middle|\; \begin{array}{l} t \leftarrow \mathsf{Gen}(1^\lambda), \\ x, x' \leftarrow \mathcal{A}(t), x \neq x'. \end{array} \right] \leq \mathsf{negl}(\lambda).$$

We assume that the existence of an algorithm HashGen that outputs hash functions satisfying the above definition. To complete the instantiation of $\mathcal{M}_\lambda$, the algorithm HashGen is invoked four times with input 1^λ to obtain keyed versions of the hash functions $H_\mathrm{s}, H_\mathrm{p}, H_\mathrm{K}, H_{pos}$. For each of these hash functions, the corresponding key generation algorithm Gen is run with input 1^λ and the resulting keys t are made public to obtain the unkeyed versions of the hash functions.

H Proof of Theorem 9.1

The existence of the extractor $\mathcal{E}_\mathrm{RCG}$ will follow from the knowledge-soundness of the zkSNARK and IVC scheme used in Nova (see Definitions E.2 and F.3).

Suppose a PPT adversary Ex uses randomness ρ to generate a zkSNARK proof π_RCG for an instance $\mathsf{inst}_\mathrm{RCG\text{-}IVC}$ that is verified to be correct by the zkSNARK verifier. Recall from Section 7.6.1 that the proof π_RCG proves knowledge of an IVC proof Π_n.

Since the zkSNARK used in Nova satisfies knowledge-soundness, by Definition F.3, there exists an EPT extractor $\mathcal{E}_\mathrm{SNARK}$ which outputs a valid IVC proof Π_n for $\mathsf{inst}_\mathrm{RCG\text{-}IVC}$ using the randomness ρ used by Ex, except with a negligible probability $\mathsf{negl}_1(\lambda)$. So the probability that the output Π_n of $\mathcal{E}_\mathrm{SNARK}$ is a valid IVC proof is at least $1 - \mathsf{negl}_1(\lambda)$.

Since the IVC scheme in Nova satisfies knowledge-soundness, setting $\mathcal{P}^* = \mathcal{E}_\mathrm{SNARK}$ in Definition E.2, there exists an EPT extractor $\mathcal{E}_\mathrm{IVC}$ which outputs a valid witness $\mathsf{wit}_\mathrm{RCG\text{-}IV}$ for $\mathsf{inst}_\mathrm{RCG\text{-}IVC}$ using the randomness ρ used by Ex, except with a negligible probability $\mathsf{negl}_2(\lambda)$. So conditioned on the event that Π_n is a valid IVC proof, the probability that the output $\mathsf{wit}_\mathrm{RCG\text{-}IV}$ of $\mathcal{E}_\mathrm{IVC}$ is a valid witness for the instance $\mathsf{inst}_\mathrm{RCG\text{-}IVC}$ is at least $1 - \mathsf{negl}_2(\lambda)$.

We construct the required extractor $\mathcal{E}_\mathrm{RCG}$ by composing $\mathcal{E}_\mathrm{SNARK}$ with $\mathcal{E}_\mathrm{IVC}$. Using the randomness ρ, $\mathcal{E}_\mathrm{RCG}$ first runs the extractor $\mathcal{E}_\mathrm{SNARK}$ to get Π_n. Then it runs the extractor $\mathcal{E}_\mathrm{IVC}$ again with input ρ to get the witness $\mathsf{wit}_\mathrm{RCG\text{-}IV}$.

The probability that $\mathsf{wit}_\mathrm{RCG\text{-}IV}$ is a valid witness is at least

$$\left(1 - \mathsf{negl}_1(\lambda)\right)\left(1 - \mathsf{negl}_2(\lambda)\right) = 1 - \mathsf{negl}_3(\lambda),$$

for a negligible function $\mathsf{negl}_3(\lambda)$. Thus the extractor $\mathcal{E}_\mathrm{RCG}$ succeeds in generating a valid witness, except with the negligible probability $\mathsf{negl}_3(\lambda)$. Since both $\mathcal{E}_\mathrm{SNARK}$ and $\mathcal{E}_\mathrm{IVC}$ have expected polynomial running times, $\mathcal{E}_\mathrm{RCG}$ has an expected polynomial running time.

I Proof of Theorem 9.2

The proof of this theorem is identical to the proof of Theorem 9.1. Suppose a PPT adversary Ex uses randomness ρ to generate a zkSNARK proof π_NC for an instance $\mathsf{inst}_\mathrm{NC\text{-}IVC}$ that is verified to be correct by the zkSNARK verifier. By the knowledge-soundness of the zkSNARK, there exists an EPT extractor $\mathcal{E}_\mathrm{SNARK}$ which outputs a valid IVC proof Π_n for $\mathsf{inst}_\mathrm{NC\text{-}IVC}$ using the randomness ρ used by Ex, except with a negligible probability $\mathsf{negl}_1(\lambda)$.

Since the IVC scheme in Nova satisfies knowledge-soundness, there exists an EPT extractor $\mathcal{E}_\mathrm{IVC}$ which outputs a valid witness $\mathsf{wit}_\mathrm{NC\text{-}IV}$ for $\mathsf{inst}_\mathrm{NC\text{-}IVC}$ using the randomness ρ used by Ex, except with a negligible probability $\mathsf{negl}_2(\lambda)$.

As in Appendix H, the required extractor $\mathcal{E}_\mathrm{NC}$ is obtained by composing $\mathcal{E}_\mathrm{SNARK}$ with $\mathcal{E}_\mathrm{IVC}$. It has an expected polynomial running time. It will generate a valid witness $\mathsf{wit}_\mathrm{NC\text{-}IV}$, except with a negligible probability $\mathsf{negl}_3(\lambda)$.

J Proof of Theorem 9.3

The outline of the proof is as follows. For every adversary $\mathcal{A}$, we will define an adversary $\mathcal{A}'$ which will run $\mathcal{A}$ as a subroutine. $\mathcal{A}'$ will generate simulated transcripts RT_j^{sim}, which are computationally indistinguishable from the actual RCG protocol transcripts RT_j.

The final output of $\mathcal{A}'$ will be the output of its subroutine $\mathcal{A}$ whose inputs are the simulated protocol transcripts.

We will argue using a hybrid argument that the vector of simulated protocol transcripts is computationally indistinguishable from a vector of actual RCG transcripts. Since $\mathcal{A}$ is a PPT algorithm, its output when fed actual protocol transcripts is computationally indistinguishable from its output when it is fed simulated protocol transcripts. This will prove the theorem.

Let RT denote a RCG protocol transcript generated by an exchange at a block height h using a blockchain instance B_{bh}. Then $\mathsf{RT} = (\mathsf{inst}_{\text{RCG-IVC}}, \pi_{\text{RCG}}) = ((\mathbf{z}_0, \mathbf{z}_n), \pi_{\text{RCG}})$ where

$$\mathbf{z}_0 = \left[h, \mathsf{root}(\mathsf{TXOT}_h), \mathsf{root}(\mathsf{KIT}_h), \mathsf{root}(\mathsf{IMT}_\emptyset), G\right],$$
$$\mathbf{z}_n = \left[h, \mathsf{root}(\mathsf{TXOT}_h), \mathsf{root}(\mathsf{KIT}_h), \mathsf{root}(\mathsf{DST}), C^{\text{res}}\right].$$

Let $\mathbf{v}$ be the DST leaves vector corresponding to the tree DST. The following lemma proves the existence of an EPT simulator $\mathcal{S}_{\text{RT}}$ which uses $\mathsf{B}_{\mathsf{bh}}, C^{\text{res}}, h$, and $\mathbf{v}$ to output a simulated transcript $\mathsf{RT}^{\mathsf{sim}}$ such that the joint distributions of $(\mathsf{RT}, \mathbf{v}, \mathsf{st}')$ and $(\mathsf{RT}^{\mathsf{sim}}, \mathbf{v}, \mathsf{st}')$ are computationally indistinguishable for any auxiliary information st' that can be generated by a polynomial-time adversary.

Lemma J.1. *Let RT denote a RCG protocol transcript generated by a honest prover at block height h using a blockchain instance B_{bh}. Let C^{res} be the reserves commitment in RT and let* $\mathbf{v}$ *be the DST leaves vector of the double spend tree generated in RT. Let pp_{RCG} be the public parameters of the RCG protocol. For any PPT adversary $\mathcal{A}$ that generates auxiliary information $st' \leftarrow \mathcal{A}(pp_{RCG})$, there exists a PPT simulator $\mathcal{S}_{RT}$ which generates a simulated protocol transcript $RT^{sim} \leftarrow \mathcal{S}_{RT}(B_{bh}, C^{res}, h, \mathbf{v})$ such that*

$$(RT, \mathbf{v}, st') \approx (RT^{sim}, \mathbf{v}, st').$$

Proof. The simulator $\mathcal{S}_{\text{RT}}$ proceeds as follows:

1. Using the blockchain instance B_{bh} and the height h, it constructs the trees TXOT_h and KIT_h.
2. Using the leaves $\mathbf{v}$, it constructs the double spend tree DST.
3. Using $h, \mathsf{root}(\mathsf{TXOT}_h), \mathsf{root}(\mathsf{KIT}_h), \mathsf{root}(\mathsf{DST}), C^{\text{res}}$, it constructs the instance $\mathsf{inst}_{\text{RCG-IVC}} = (\mathbf{z}_0, \mathbf{z}_n)$.
4. For the zkSNARK used in Nova, there exists a PPT simulator $\mathcal{S}_{\text{RCG}}$ (see Appendix D of the Nova preprint [29]) which takes as input the instance $\mathsf{inst}_{\text{RCG-IVC}}$ and generates a simulated zkSNARK proof
$$\pi_{\text{sim}} \leftarrow \mathcal{S}_{\text{RCG}}\left(\mathsf{pp}_{\text{RCG}}, \mathsf{s}_{\text{RCG}}, \mathsf{inst}_{\text{RCG-IVC}}\right).$$
such that
$$\left(\mathsf{pp}_{\text{RCG}}, \mathsf{s}_{\text{RCG}}, \mathsf{inst}_{\text{RCG-IVC}}, \pi_{\text{RCG}}, \mathsf{st}\right) \approx \left(\mathsf{pp}_{\text{RCG}}, \mathsf{s}_{\text{RCG}}, \mathsf{inst}_{\text{RCG-IVC}}, \pi_{\text{sim}}, \mathsf{st}\right)$$
for any polynomially computable auxiliary information st. While zero-knowledge as defined in Definition F.5 only requires the simulator $\mathcal{S}_{\text{RCG}}$ to run in *expected* polynomial-time, $\mathcal{S}_{\text{RCG}}$ in fact has a polynomial running time. The simulator $\mathcal{S}_{\text{RT}}$ runs $\mathcal{S}_{\text{RCG}}$ as a subroutine to obtain the simulated proof π_{sim}.
5. The simulator $\mathcal{S}_{\text{RT}}$ outputs the simulated RCG protocol transcript as $\mathsf{RT}^{\mathsf{sim}} = (\mathsf{inst}_{\text{RCG-IVC}}, \pi_{\text{sim}})$.

Since each of the above steps can be executed in polynomial time, $\mathcal{S}_{\text{RT}}$ has a polynomial running time.

For the case when the auxiliary information $\mathsf{st} = (\mathbf{v}, \mathsf{st}')$, we have

$$\begin{aligned}
&\left(\mathsf{pp}_{\text{RCG}}, \mathsf{s}_{\text{RCG}}, \mathsf{inst}_{\text{RCG-IVC}}, \pi_{\text{RCG}}, \mathbf{v}, \mathsf{st}'\right) \\
&\qquad\approx \left(\mathsf{pp}_{\text{RCG}}, \mathsf{s}_{\text{RCG}}, \mathsf{inst}_{\text{RCG-IVC}}, \pi_{\text{sim}}, \mathbf{v}, \mathsf{st}'\right) \\
\implies &\left(\mathsf{inst}_{\text{RCG-IVC}}, \pi_{\text{RCG}}, \mathbf{v}, \mathsf{st}'\right) \approx \left(\mathsf{inst}_{\text{RCG-IVC}}, \pi_{\text{sim}}, \mathbf{v}, \mathsf{st}'\right) \\
\implies &\left(\mathsf{RT}, \mathbf{v}, \mathsf{st}'\right) \approx \left(\mathsf{RT}^{\mathsf{sim}}, \mathbf{v}, \mathsf{st}'\right).
\end{aligned}$$

This completes the proof of the lemma. □

For $j = 1, 2, \ldots, x(\lambda)$, let RT_j be the jth RCG protocol generated by an honest prover. Let $\mathbf{v}^{(j)}$ be the leaves vector of the DST generated in RT_j. Let $\mathsf{RT}_j^{\mathsf{sim}}$ be the jth simulated proof corresponding to RT_j generated by the simulator $\mathcal{S}_{\text{RT}}$ described in Lemma J.1. Note that $\mathcal{S}_{\text{RT}}$ is independent of the auxiliary information st'; it only depends on the values of $\mathsf{B}_{\mathsf{bh}}, C^{\text{res}}, h, \mathbf{v}$. We will use a hybrid argument to show that the vector of actual protocol transcript and DST leaves pairs is computationally indistinguishable from a vector of simulated protocol transcript and DST leaves pairs. This is stated precisely in the following lemma.

Lemma J.2. *Let RT_j, RT_j^{sim},* $\mathbf{v}^{(j)}$ *be as defined above. Then we have*

$$\left[B_{bh}, (RT_1, \mathbf{v}^{(1)}), (RT_2, \mathbf{v}^{(2)}), \ldots, (RT_{x(\lambda)}, \mathbf{v}^{(x(\lambda))})\right] \approx \left[B_{bh}, (RT_1^{sim}, \mathbf{v}^{(1)}), (RT_2^{sim}, \mathbf{v}^{(2)}), \ldots, (RT_{x(\lambda)}^{sim}, \mathbf{v}^{(x(\lambda))})\right].$$

Proof. For every $l \in \{0, 1, \ldots, x(\lambda)\}$, we define a hybrid random variable T_l as a vector containing l simulated RCG protocol transcripts followed by $x(\lambda) - l$ actual RCG protocol transcripts,

$$T_l = \begin{bmatrix} \mathsf{B}_{\mathsf{bh}}, (\mathsf{RT}_1^{\mathsf{sim}}, \mathbf{v}^{(1)}), \ldots, (\mathsf{RT}_l^{\mathsf{sim}}, \mathbf{v}^{(l)}), \\ (\mathsf{RT}_{l+1}, \mathbf{v}^{(l+1)}), \ldots, (\mathsf{RT}_{x(\lambda)}, \mathbf{v}^{(x(\lambda))}) \end{bmatrix}.$$

For $l = 0$, we have

$$T_0 = \left[\mathsf{B}_{\mathsf{bh}}, (\mathsf{RT}_1, \mathbf{v}^{(1)}), (\mathsf{RT}_2, \mathbf{v}^{(2)}), \ldots, (\mathsf{RT}_{x(\lambda)}, \mathbf{v}^{(x(\lambda))})\right],$$

and for $l = x(\lambda)$, we have

$$T_{x(\lambda)} = \left[\mathsf{B}_{\mathsf{bh}}, (\mathsf{RT}_1^{\mathsf{sim}}, \mathbf{v}^{(1)}), (\mathsf{RT}_2^{\mathsf{sim}}, \mathbf{v}^{(2)}), \ldots, (\mathsf{RT}_{x(\lambda)}^{\mathsf{sim}}, \mathbf{v}^{(x(\lambda))})\right].$$

To prove the lemma, we need to prove that $T_0 \approx T_{x(\lambda)}$. We will prove this by showing that

$$T_0 \approx T_1 \approx T_2 \approx \cdots \approx T_{x(\lambda)-1} \approx T_{x(\lambda)}.$$

To see why $T_0 \approx T_1$, consider a PPT adversary $\mathcal{A}_1$ in Lemma J.1 that generates the auxiliary information

$$\mathsf{st}_1' = \left[\mathsf{B}_{\mathsf{bh}}, (\mathsf{RT}_2, \mathbf{v}^{(2)}), \ldots, (\mathsf{RT}_{x(\lambda)}, \mathbf{v}^{(x(\lambda))})\right].$$

The adversary $\mathcal{A}_1$ could be a coalition of exchanges which generates $x(\lambda) - 1$ actual RCG protocol transcripts with indices $2, 3, \ldots, x(\lambda)$ and reveals the corresponding DST leaves vectors. By Lemma J.1, there exists a PPT simulator $\mathcal{S}_{\text{RT}}$ that generates a simulated protocol transcript

$$\mathsf{RT}_1^{\mathsf{sim}} \leftarrow \mathcal{S}_{\text{RT}}\left(\mathsf{B}_{\mathsf{bh}}, C_1^{\text{res}}, h_1, \mathbf{v}^{(1)}\right)$$

such that

$$(\mathsf{RT}_1, \mathbf{v}^{(1)}, \mathsf{st}'_1) \approx (\mathsf{RT}_1^{\mathsf{sim}}, \mathbf{v}^{(1)}, \mathsf{st}'_1) \implies T_0 \approx T_1.$$

To prove that $T_1 \approx T_2$, consider a PPT adversary $\mathcal{A}_2$ in Lemma J.1 that generates the auxiliary information

$$\mathsf{st}'_2 = \begin{bmatrix} \mathsf{B}_{\mathsf{bh}}, (\mathsf{RT}_1^{\mathsf{sim}}, \mathbf{v}^{(1)}), (\mathsf{RT}_3, \mathbf{v}^{(3)}), (\mathsf{RT}_4, \mathbf{v}^{(4)}), \\ \dots, (\mathsf{RT}_{x(\lambda)}, \mathbf{v}^{(x(\lambda))}) \end{bmatrix}.$$

The adversary $\mathcal{A}_2$ could be a coalition of exchanges which generates $x(\lambda) - 1$ actual RCG protocol transcripts with indices $1, 3, \dots, x(\lambda)$ and replaces RT_1 with $\mathsf{RT}_1^{\mathsf{sim}}$ by using the simulator $\mathcal{S}_{\mathrm{RT}}$. As before, this coalition reveals the DST leaves vectors corresponding to the transcripts. By Lemma J.1, there exists a PPT simulator $\mathcal{S}_{\mathrm{RT}}$ that generates a simulated protocol transcript

$$\mathsf{RT}_2^{\mathsf{sim}} \leftarrow \mathcal{S}_{\mathrm{RT}}\left(\mathsf{B}_{\mathsf{bh}}, C_2^{\mathrm{res}}, h_2, \mathbf{v}^{(2)}\right)$$

such that

$$\begin{aligned} &(\mathsf{RT}_2, \mathbf{v}^{(2)}, \mathsf{st}'_2) \approx (\mathsf{RT}_2^{\mathsf{sim}}, \mathbf{v}^{(2)}, \mathsf{st}'_2) \\ \implies &\begin{bmatrix} \mathsf{B}_{\mathsf{bh}}, (\mathsf{RT}_1^{\mathsf{sim}}, \mathbf{v}^{(1)}), (\mathsf{RT}_2, \mathbf{v}^{(2)}), (\mathsf{RT}_3, \mathbf{v}^{(3)}), \\ \dots, (\mathsf{RT}_{x(\lambda)}, \mathbf{v}^{(x(\lambda))}) \end{bmatrix} \\ &\approx \\ &\begin{bmatrix} \mathsf{B}_{\mathsf{bh}}, (\mathsf{RT}_1^{\mathsf{sim}}, \mathbf{v}^{(1)}), (\mathsf{RT}_2^{\mathsf{sim}}, \mathbf{v}^{(2)}), (\mathsf{RT}_3, \mathbf{v}^{(3)}), \\ \dots, (\mathsf{RT}_{x(\lambda)}, \mathbf{v}^{(x(\lambda))}) \end{bmatrix} \\ \implies &T_1 \approx T_2. \end{aligned}$$

To prove that $T_2 \approx T_3$, we will consider the auxiliary information

$$\mathsf{st}'_3 = \begin{bmatrix} \mathsf{B}_{\mathsf{bh}}, (\mathsf{RT}_1^{\mathsf{sim}}, \mathbf{v}^{(1)}), (\mathsf{RT}_2^{\mathsf{sim}}, \mathbf{v}^{(2)}), (\mathsf{RT}_4, \mathbf{v}^{(4)}), \\ (\mathsf{RT}_5, \mathbf{v}^{(5)}), \dots, (\mathsf{RT}_{x(\lambda)}, \mathbf{v}^{(x(\lambda))}) \end{bmatrix}.$$

By Lemma J.1, there exists a PPT simulator $\mathcal{S}_{\mathrm{RT}}$ that generates a simulated protocol transcript

$$\mathsf{RT}_3^{\mathsf{sim}} \leftarrow \mathcal{S}_{\mathrm{RT}}\left(\mathsf{B}_{\mathsf{bh}}, C_3^{\mathrm{res}}, h_3, \mathbf{v}^{(3)}\right)$$

such that

$$\begin{aligned} &(\mathsf{RT}_3, \mathbf{v}^{(3)}, \mathsf{st}'_3) \approx (\mathsf{RT}_3^{\mathsf{sim}}, \mathbf{v}^{(3)}, \mathsf{st}'_3) \\ \implies &\begin{bmatrix} \mathsf{B}_{\mathsf{bh}}, (\mathsf{RT}_1^{\mathsf{sim}}, \mathbf{v}^{(1)}), (\mathsf{RT}_2^{\mathsf{sim}}, \mathbf{v}^{(2)}), (\mathsf{RT}_3, \mathbf{v}^{(3)}), \\ (\mathsf{RT}_4, \mathbf{v}^{(4)}), \dots, (\mathsf{RT}_{x(\lambda)}, \mathbf{v}^{(x(\lambda))}) \end{bmatrix} \\ &\approx \\ &\begin{bmatrix} \mathsf{B}_{\mathsf{bh}}, (\mathsf{RT}_1^{\mathsf{sim}}, \mathbf{v}^{(1)}), (\mathsf{RT}_2^{\mathsf{sim}}, \mathbf{v}^{(2)}), (\mathsf{RT}_3^{\mathsf{sim}}, \mathbf{v}^{(3)}), \\ (\mathsf{RT}_4, \mathbf{v}^{(4)}), \dots, (\mathsf{RT}_{x(\lambda)}, \mathbf{v}^{(x(\lambda))}) \end{bmatrix} \\ \implies &T_2 \approx T_3. \end{aligned}$$

In general, for $l = 1, 2, \dots, x(\lambda)$ the lth auxiliary information will have the form

$$\mathsf{st}'_l = \begin{bmatrix} \mathsf{B}_{\mathsf{bh}}, (\mathsf{RT}_1^{\mathsf{sim}}, \mathbf{v}^{(1)}), (\mathsf{RT}_2^{\mathsf{sim}}, \mathbf{v}^{(2)}), \dots, (\mathsf{RT}_{l-1}^{\mathsf{sim}}, \mathbf{v}^{(l-1)}), \\ (\mathsf{RT}_{l+1}, \mathbf{v}^{(l+1)}), (\mathsf{RT}_{l+2}, \mathbf{v}^{(l+2)}), \dots, (\mathsf{RT}_{x(\lambda)}, \mathbf{v}^{(x(\lambda))}) \end{bmatrix}.$$

And the simulator will generate the lth simulated protocol transcript as

$$\mathsf{RT}_l^{\mathsf{sim}} \leftarrow \mathcal{S}_{\mathrm{RT}}\left(\mathsf{B}_{\mathsf{bh}}, C_l^{\mathrm{res}}, h_l, \mathbf{v}^{(l)}\right)$$

such that

$$(\mathsf{RT}_l, \mathbf{v}^{(l)}, \mathsf{st}'_l) \approx (\mathsf{RT}_l^{\mathsf{sim}}, \mathbf{v}^{(l)}, \mathsf{st}'_l) \implies T_{l-1} \approx T_l.$$

Then for any PPT distinguisher $\mathcal{D}$, there exists a negligible function negl such that

$$\left| \Pr\left[\mathcal{D}\left(T_{l-1}\right) = 1\right] - \Pr\left[\mathcal{D}\left(T_l\right) = 1\right] \right| \leq \mathsf{negl}(\lambda).$$

To complete the proof, consider any PPT distinguisher $\mathcal{D}$. Then

$$\begin{aligned} &\left| \Pr\left[\mathcal{D}\left(T_0\right) = 1\right] - \Pr\left[\mathcal{D}\left(T_{x(\lambda)}\right) = 1\right] \right| \\ =&\Big| \Pr\left[\mathcal{D}\left(T_0\right) = 1\right] - \Pr\left[\mathcal{D}\left(T_1\right) = 1\right] \\ &+ \Pr\left[\mathcal{D}\left(T_1\right) = 1\right] - \Pr\left[\mathcal{D}\left(T_2\right) = 1\right] \\ &+ \Pr\left[\mathcal{D}\left(T_2\right) = 1\right] - \Pr\left[\mathcal{D}\left(T_3\right) = 1\right] \\ &\quad\vdots \\ &+ \Pr\left[\mathcal{D}\left(T_{x(\lambda)-2}\right) = 1\right] - \Pr\left[\mathcal{D}\left(T_{x(\lambda)-1}\right) = 1\right] \\ &+ \Pr\left[\mathcal{D}\left(T_{x(\lambda)-1}\right) = 1\right] - \Pr\left[\mathcal{D}\left(T_{x(\lambda)}\right) = 1\right] \Big| \\ \leq &\sum_{l=1}^{x(\lambda)} \left| \Pr\left[\mathcal{D}\left(T_{l-1}\right) = 1\right] - \Pr\left[\mathcal{D}\left(T_l\right) = 1\right] \right| \\ \leq &\ x(\lambda) \cdot \mathsf{negl}(\lambda) \end{aligned}$$

for some negligible function negl. Since $x(\lambda)$ is a polynomial in λ, $x(\lambda) \cdot \mathsf{negl}(\lambda)$ is also negligible. This implies that $T_0 \approx T_{x(\lambda)}$. □

To complete the proof of Theorem 9.3, we need to show that for every PPT adversary $\mathcal{A}$ there is a PPT adversary $\mathcal{A}'$ such that

$$\begin{aligned} &\mathcal{A}\left(\mathsf{B}_{\mathsf{bh}}, (\mathsf{RT}_1, \mathbf{v}^{(1)}), (\mathsf{RT}_2, \mathbf{v}^{(2)}), \dots, (\mathsf{RT}_{x(\lambda)}, \mathbf{v}^{(x(\lambda))})\right) \approx \\ &\mathcal{A}'\left(\mathsf{B}_{\mathsf{bh}}, C_1^{\mathrm{res}}, \dots, C_{x(\lambda)}^{\mathrm{res}}, \mathbf{v}^{(1)}, \dots, \mathbf{v}^{(x(\lambda))}, h_1, \dots, h_{x(\lambda)}\right). \end{aligned}$$

For any PPT adversary $\mathcal{A}$, the strategy of the corresponding adversary $\mathcal{A}'$ will be as follows:

(1) For $i = 1, 2, \dots, x(\lambda)$, the adversary $\mathcal{A}'$ uses the simulator $\mathcal{S}_{\mathrm{RT}}$ of Lemma J.1 with $\mathsf{B}_{\mathsf{bh}}, C_i^{\mathrm{res}}, \mathbf{v}^{(i)}, h_i$ as inputs to generate the ith simulated RCG protocol transcript

$$\mathsf{RT}_i^{\mathsf{sim}} \leftarrow \mathcal{S}_{\mathrm{RT}}\left(\mathsf{B}_{\mathsf{bh}}, C_i^{\mathrm{res}}, h_i, \mathbf{v}^{(i)}\right).$$

(2) $\mathcal{A}'$ will run the adversary $\mathcal{A}$ as a subroutine with actual RCG protocol transcripts replaced with simulated protocol transcripts as

$$\mathcal{A}\left(\mathsf{B}_{\mathsf{bh}}, (\mathsf{RT}_1^{\mathsf{sim}}, \mathbf{v}^{(1)}), (\mathsf{RT}_2^{\mathsf{sim}}, \mathbf{v}^{(2)}), \dots, (\mathsf{RT}_{x(\lambda)}^{\mathsf{sim}}, \mathbf{v}^{(x(\lambda))})\right).$$

The final output of $\mathcal{A}'$ will be the output of its subroutine $\mathcal{A}$.

By Lemma J.2, we know that

$$\begin{aligned} &\left[\mathsf{B}_{\mathsf{bh}}, (\mathsf{RT}_1, \mathbf{v}^{(1)}), \dots, (\mathsf{RT}_{x(\lambda)}, \mathbf{v}^{(x(\lambda))})\right] \approx \\ &\left[\mathsf{B}_{\mathsf{bh}}, (\mathsf{RT}_1^{\mathsf{sim}}, \mathbf{v}^{(1)}), \dots, (\mathsf{RT}_{x(\lambda)}^{\mathsf{sim}}, \mathbf{v}^{(x(\lambda))})\right] \end{aligned}$$

Since $\mathcal{A}$ is a PPT algorithm, we have

$$\begin{aligned} &\mathcal{A}\left(\mathsf{B}_{\mathsf{bh}}, (\mathsf{RT}_1, \mathbf{v}^{(1)}), \dots, (\mathsf{RT}_{x(\lambda)}, \mathbf{v}^{(x(\lambda))})\right) \approx \\ &\mathcal{A}\left(\mathsf{B}_{\mathsf{bh}}, (\mathsf{RT}_1^{\mathsf{sim}}, \mathbf{v}^{(1)}), \dots, (\mathsf{RT}_{x(\lambda)}^{\mathsf{sim}}, \mathbf{v}^{(x(\lambda))})\right) \end{aligned}$$

Since the random variable on the right hand side is

$$\mathcal{A}'\left(\mathsf{B}_{\mathsf{bh}}, C_1^{\text{res}}, \ldots, C_{x(\lambda)}^{\text{res}}, \mathbf{v}^{(1)}, \ldots, \mathbf{v}^{(x(\lambda))}, h_1, \ldots, h_{x(\lambda)}\right),$$

we have proved the claim of Theorem 9.3. □

K List of Symbols

A list of symbols used to describe the MProve-Nova protocol is given in Table 3 for easy reference.

Table 3: List of symbols used in MProve-Nova description

Symbol	Description
$\mathbb{G}$	Prime order subgroup of the ed25519 elliptic curve whose order is a 253-bit prime l
G	The base point of the group $\mathbb{G}$
H	An element of group $\mathbb{G}$ whose discrete logarithm wrt G is unknown. Used to generate Pedersen commitments
$\mathbb{Z}_l$	The set $\{0, 1, 2, \ldots, l-1\}$ of scalars from which Monero private keys and Pedersen commitment blinding factors are chosen
H_s	$H_\mathrm{s} : \{0,1\}^* \mapsto \mathbb{Z}_l$ is a scalar-valued cryptographic hash function that is used in Monero one-time address generation
H_p	$H_\mathrm{p} : \mathbb{G} \mapsto \mathbb{G}$ is a point-valued cryptographic hash function that is used in Monero key image generation
P	An element in $\mathbb{G}$ representing a one-time address. Usually appears with subscripts
C	An element in $\mathbb{G}$ representing a Pedersen commitment of the form $aH + yG$ for $a, y \in \mathbb{Z}_l$ where a is the amount and y is a blinding factor. To explicitly specify a and y, the commitment $C = aH + yG$ is sometimes written as $C(a, y)$.
(P, C)	An element of $\mathbb{G}^2$ that denotes a Monero RingCT output
C_res	A Pedersen commitment to an exchange's reserves amount
$\mathbb{F}_s$	Prime field used to express R1CS constraints
H_{pos}	$H_{pos} : \{0,1\}^* \mapsto \mathbb{F}_s$ is the Poseidon hash function
bh	Block height
N_bh	Number of RingCT outputs that have appeared on the Monero blockchain up to block height bh
$\mathcal{T}_\mathsf{bh}$	Vector of all outputs $[(P_1, C_1), (P_2, C_2), \ldots, (P_{N_\mathsf{bh}}, C_{N_\mathsf{bh}})]$ that have appeared on the Monero blockchain up to block height bh
$\mathcal{I}_\mathsf{bh}$	Set of all key images that have appeared on the Monero blockchain up to block height bh
$\mathcal{J}_\mathrm{known}$	A subset of $\{1, 2, \ldots, N_\mathsf{bh}\}$ containing the indices of outputs which are owned by an exchange
$\mathcal{J}_\mathrm{unspent}$	A subset of $\{1, 2, \ldots, N_\mathsf{bh}\}$ containing the indices of unspent outputs
TXOT	Transaction outputs tree, a regular Merkle tree with leaves of the form $H_{pos}\left(P \| C \| H_\mathrm{p}(P)\right)$ where $(P, C) \in \mathcal{T}_\mathsf{bh}$
KIT	Key images tree, an indexed Merkle tree with leaves of the form $H_{pos}\,(I)$ where $I \in \mathcal{I}_\mathsf{bh}$
DST	Double spend tree, an indexed Merkle tree with leaves of the form $H_{pos}\,(x \| \mathsf{bh})$ where $x \in \mathbb{Z}_l$. DST_{j-1} and DST_j denote the double spend trees before and after the jth step of the RCG protocol
OIT	Output inclusion tree, an indexed Merkle tree with leaves of the form $H_{pos}\,(x \| \mathsf{bh})$ where $x \in \mathbb{Z}_l$. OIT_{j-1} and OIT_j denote the output inclusion trees before and after the jth step of the NC protocol
$\mathsf{inst}_\mathrm{RCG}$	Instance of the RCG protocol (see Definition 7.1)
$\mathsf{wit}_\mathrm{RCG}$	Witness of the RCG protocol (see Definition 7.2)
$\mathsf{inst}_\text{RCG-IVC}$	Instance of the IVC version of the RCG protocol (see Definition 7.3)
$\mathsf{wit}_\text{RCG-IVC}$	Witness of the IVC version of the RCG protocol (see Definition 7.4)
pk_RCG	Nova proving key for the RCG protocol
vk_RCG	Nova verification key for the RCG protocol
π_RCG	Nova zkSNARK proof for the RCG protocol
$\mathcal{V}_\mathrm{RCG}$	Nova zkSNARK verifier for the RCG protocol
$\mathsf{DST}_{n_1}^{\mathsf{Ex1}}$	Double spend tree of Exchange 1 containing n_1 leaves
$v_j^{(1)}$	The jth leaf of $\mathsf{DST}_{n_1}^{\mathsf{Ex1}}$ for $j = 1, \cdots, n_1$
$\mathsf{DST}_{n_2}^{\mathsf{Ex2}}$	Double spend tree of Exchange 2 containing n_2 leaves
$v_j^{(2)}$	The jth leaf of $\mathsf{DST}_{n_2}^{\mathsf{Ex2}}$ for $j = 1, \cdots, n_2$
$\mathsf{inst}_\mathrm{NC}$	Instance of the NC protocol (see Definition 8.1)
wit_NC	Witness of the NC protocol (see Definition 8.2)
$\mathsf{inst}_\text{NC-IVC}$	Instance of the IVC version of the NC protocol (see Definition 8.3)
$\mathsf{wit}_\text{NC-IVC}$	Witness of the IVC version of the NC protocol (see Definition 8.4)
pk_NC	Nova proving key for the NC protocol
vk_NC	Nova verification key for the NC protocol
π_NC	Nova zkSNARK proof for the NC protocol
$\mathcal{V}_\mathrm{NC}$	Nova zkSNARK verifier for the NC protocol
$\mathsf{IMT}_\emptyset$	An empty indexed Merkle
λ	A security parameter taking values in $\mathbb{N}$
$\mathcal{M}_\lambda$	A sequence of Monero-like systems where the discrete logarithm problem in $\mathbb{G}$, the decisional Diffie-Hellman problem in $\mathbb{G}$, the problems of finding collisions or preimages of hash functions $H_\mathrm{s}, H_\mathrm{p}, H_{pos}$, all become harder with increasing λ
B_bh	An instance of the Monero blockchain up to block height bh, generated using the specifications in $\mathcal{M}_\lambda$
RT_j	The jth RCG protocol transcript containing the jth instance-proof pair $\left(\mathsf{inst}_\text{RCG-IVC}^{(j)}, \pi_\mathrm{RCG}^{(j)}\right)$
$\mathsf{v}^{(j)}$	The vector of leaves of the double spend tree corresponding to RT_j