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

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

ABSTRACT

We present MProve-Nova, a proof of reserves (PoR) protocol for Monero that leverages the Nova recursive SNARK to achieve two firsts (without requiring any trusted setup). It is the first privacy-preserving Monero PoR protocol that does not reveal any information about the exchange-owned outputs or their key images. It is 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.

In our implementation of MProve-Nova, we observed proof sizes of 27 KB and verification times of 4.5 seconds. Proving times increase linearly with the number of outputs owned by the exchange ($\approx$ 0.7 hour per 1,000 outputs) but remain independent of the number of outputs on the Monero blockchain. We also describe the design and implementation of a Nova-based non-collusion protocol that takes the MProve-Nova proofs from a pair of exchanges as input and proves that the sets of outputs they used to generate their respective proofs of reserves are non-overlapping.

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 [50] and the collapse of FTX [49]. The total funds lost due to exchange hacks alone is estimated to be at least \$2.4 billion as of April 2023 [13]. 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 [35], a privacy-focused cryptocurrency based on the CryptoNote protocol [47]. Specifically, 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 coin addresses (*outputs*) owned by the exchange.

Without this 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*[1] have not appeared in any past transaction.

Privacy is especially important in Monero PoR protocols because Monero transactions contain 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, a proof of solvency is useless if no one verifies it. So it is desirable to have PoR/PoL 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.

Paper organization. In the next section, we give a brief overview of Monero. In Section 3, we describe the challenges involved in designing Monero PoR protocols. Our contributions are listed in Section 4. Section 5 describes related work. In Section 6, we briefly cover aspects of Nova background required to describe MProve-Nova. Section 7 describes the design of the MProve-Nova protocol and Section 8 describes the proof of non-collusion. Their security

[1]Key images are one-way functions of outputs that are generated as part of the Monero ring signatures to prevent double spending.

properties are discussed in Section 9. We provide implementation details and performance results in Section 10 followed by conclusions in Section 11.

2 OVERVIEW OF MONERO

Monero [35] is the most popular instantiation of the CryptoNote protocol [47], 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* [41].

2.1 One-Time Addresses

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 $\mathbb{Z}_l^+ = \{1, 2, ..., l-1\}$. For the basepoint $G \in \mathbb{G}$, 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 $(b_{vk}, b_{sk}) \in \mathbb{Z}_l^+ \times \mathbb{Z}_l^+$ denote the corresponding private key pair.
(2) She signs a transaction transferring coins she owns to Bob. This transaction will contain a *one-time address* P that will be controlled by Bob and a random point R.
(3) Alice creates the one-time address P as follows:
   (i) She chooses a random scalar $r \in \mathbb{Z}_l^+$.
   (ii) She computes the one-time address as
$$P = H(rB_{vk} \| o_{index})G + B_{sk}$$
   where $\|$ denotes concatenation, $H : \{0,1\}^* \mapsto \mathbb{Z}_l^+$ is a cryptographic hash function, and o_{index} is the index of the new *output* (defined in Section 2.4) in the transaction.[2] Note that the private key corresponding to P is known *only* to Bob as it equals
$$x = H(rB_{vk} \| o_{index})G + b_{sk},$$
   where $B_{sk} = b_{sk}G$.
(4) Alice computes the random point R as rG.
(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 $P' = H(b_{vk}R \| o_{index})G + B_{sk}$.
   (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.

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

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. In this way, Monero hides the receiver's identity in a transaction.

2.2 Linkable Ring Signatures

Monero uses linkable ring signatures [32, 40] 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
$$\mathcal{P} = \{P_1, P_2, \ldots, P_{n-1}, P\}.$$
   This set is sorted in chronological order (oldest address first) to prevent the ordering of the keys in $\mathcal{P}$ from leaking the identity of P.
(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 Ed25519 point 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 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 $\mathbb{G}$.

2.3 Pedersen Commitments to Amounts

In the 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 [41] to hide the number of coins tied to a one-time address. The Pedersen commitment to an amount $a \in \{0, 1, 2, \ldots, 2^{64} - 1\}$ is given by

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

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.

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 $\{0, 1, 2, \ldots, 2^{64} - 1\}$.

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

$$\begin{aligned} a' &= a \oplus H_K(H_K(rB_{vk})) \\ y' &= y \oplus H_K(rB_{vk}) \end{aligned}$$

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

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

2.4 Monero Outputs

The destination of coin transfers in a Monero transaction is called an *output*. A transaction can have multiple outputs. 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.

Outputs containing Pedersen commitments are created in a Monero transaction type called *ring confidential transaction (RingCT)* [40]. Monero made the RingCT type of transactions mandatory in September 2017 [36].

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 $C(0, a) = aH$, to represent the output.

3 CHALLENGES IN DESIGNING A MONERO POR PROTOCOL

In this section, we describe the challenges involved in designing Monero PoR protocols. We also describe how previous PoR protocols for Monero address these challenges and where they fall short. This description will help clarify our contributions listed in Section 4. 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 [28, 37], it is not possible to know if any of the RingCT outputs (except for 5 of them) have been spent [48, 51].

Any Monero PoR protocol must prove two compound 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.

As discussed in the previous section, one cannot simply reveal the exchange-owned outputs as this would violate the privacy of both the exchange and other Monero users. One solution is to give a privacy-preserving proof of membership of exchange-owned outputs in the set of all outputs, together with a proof of knowledge of private keys for each such output.

But this is not enough. One needs to also give a privacy-preserving proof that the exchange-owned outputs are *unspent*. Furthermore, one needs to ensure that a malicious exchange does not *double count*, i.e. it does not artificially inflate its reserves by counting the coins in an output more than once.

Previous Monero PoR protocols, namely MProve [21] and MProve+ [20], partially achieved the above goals as follows:

- They both proved that the exchange-owned outputs belonged to an *anonymity set*, a subset of the set of all outputs that appeared on the Monero blockchain. For performance reasons, they chose the anonymity set to be much smaller than the set of all outputs.
- They both revealed one key image per exchange-owned output that contributed to the reserves. This had several implications:
  - The verifier could verify that the exchange was using only unspent outputs by checking that the revealed

    key images had not appeared in any transaction on Monero blockchain.
  - A malicious exchange could not double count, as using an output more than once in the same PoR would result in a repeated key image.
  - The verifier could verify that two different exchanges were not colluding by checking that the sets of key images revealed by them were non-overlapping. So the proof of non-collusion was implicitly available in the original protocol itself.

But these approaches had some drawbacks.

- The proof size and proof verification time in both MProve and MProve+ increased linearly with the size of the anonymity set. Additionally, in MProve+ the memory required for generating and verifying proofs also increased linearly with the size of the anonymity set.
- More seriously, revealing the key images negatively impacted the privacy of the exchange and, in some cases, the privacy of regular Monero users.
  - In MProve, if the exchange spent from an output that was used to previously generate a MProve proof, then the output was immediately identified as the spending output in the transaction ring. This meant that all previous transaction rings where this output appeared as a decoy had their effective ring size reduced by one.
  - MProve+ improved upon MProve by breaking the direct link between the key image and an exchange-owned output. If the exchange spent from an output that was used to previously generate a MProve+ proof, then that output could be identified as the spending output only if the transaction ring and the anonymity set had exactly one output in common. But the transaction would always be identified as being initiated by the exchange.

In this paper, we propose a Monero PoR protocol which does not suffer these drawbacks, while preventing malicious provers from using spent outputs or double counting unspent outputs.

4 OUR CONTRIBUTIONS

Our contributions are as follows:

(1) We describe the design and implementation of MProve-Nova, a publicly-verifiable privacy-preserving PoR protocol for Monero. Our design is based on Nova [27], a recursive SNARK that does not require a trusted setup. MProve-Nova has the following advantages:
   (a) It is privacy-preserving in the random oracle model in the sense that it does not reveal any information about the exchange-owned outputs or their key images.
   (b) It is the first Monero PoR protocol with *constant* proof size and proof verification time 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. It achieves this by leveraging the Nova folding scheme to fold all the individual statements proving ownership of an exchange-owned output and the fact that it is unspent into a single statement.

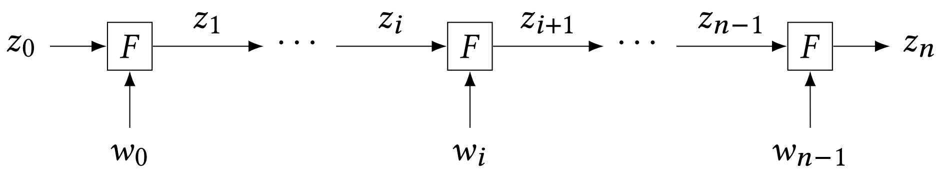
(2) We analyze the security of MProve-Nova. Using the soundness and zero-knowledge properties of Nova, we show that it is
   (a) *inflation resistant*, i.e. a computationally bounded exchange cannot generate an MProve-Nova proof that proves that it owns a number of coins that is greater than the actual number of coins it owns, and
   (b) privacy-preserving in the random oracle model.

(3) The reference implementation of Nova [33] is currently not zero-knowledge [34]. But Angel *et al.* [3, 4] 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 MProve-Nova in Rust [38].
   To leverage Nova, the statements requiring proof need to be expressed using rank-1 constraint system (R1CS) constraints. While it was possible to create an R1CS gadget for MProve-Nova using the circom circuit compiler [23] and the Nova Scotia adapter [10], we chose to implement the gadget directly using the bellpepper library [30].
   To implement the MProve-Nova gadget, we implemented the following component gadgets in bellpepper.
   (a) *Regular and indexed Merkle trees* [5, 45]: While implementations of regular Merkle trees already existed in bellman [53] (which was forked to create bellpepper), our implementation of indexed Merkle trees is new.
   (b) bellpepper-emulated: A gadget for non-native finite field arithmetic inspired by the emulated [18] package (written in Go) from the gnark zkSNARK library [11]. When the field order is a pseudo-Mersenne prime, we added limb folding and efficient field membership checks inspired by techniques from circom-ecdsa [1].
   (c) bellpepper-ed25519: A gadget for Ed25519 elliptic curve operations using bellperson-emulated. This was needed as Monero uses Ed25519 curve points for addresses (outputs) and amount commitments.

(4) We describe the design, implementation [38], and security properties of a protocol for generating a *proof of non-collusion* between a pair of exchanges that use MProve-Nova to generate their respective proofs of reserves. This protocol is also based on Nova, resulting in constant proof sizes and proof verification times.
   This protocol involves a pair of exchanges, where only one of them needs to generate the proof. The exchange generating the proof will discover the number of outputs owned by the other exchange. No other information is revealed. The proof can be verified by third parties, who do not gain any knowledge about the exchange-owned outputs (not even their number).

5 RELATED WORK

Provisions [19], proposed by Dagher *et al.*, is one of the first privacy-preserving proof of solvency protocol for Bitcoin exchanges. It consists of three sub-protocols: proof of reserves, proof of liabilities



$$\begin{tikzcd} z_0 \arrow[r] & \boxed{F} \arrow[r, "z_1"] & \cdots & \arrow[r, "z_i"] & \boxed{F} \arrow[r, "z_{i+1}"] & \cdots & \arrow[r, "z_{n-1}"] & \boxed{F} \arrow[r] & z_n \\ & w_0 \arrow[u] & & & w_i \arrow[u] & & & w_{n-1} \arrow[u] & \end{tikzcd}$$

Figure 1: Incrementally Verifiable Computation

and proof of solvency. In proof of reserves, the exchange generates a Pedersen commitment [41] 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 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 $C_{\text{liabilities}}$ of the exchange. In proof of solvency, the exchange computes $C_{\text{diff}} = C_{\text{res}} - C_{\text{liabilities}}$ 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 [42, 43] 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 [44]. 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 are revealed as part of the proof of reserves. This technique reveals the addresses owned by the prover, their corresponding amounts, and their key images.

Dutta *et al.* [21] proposed MProve, a proof of reserves protocol for Monero exchanges. MProve+ [20] was later proposed which enhanced the privacy of MProve by using techniques from Bulletproofs [12] and Omniring [31]. The details of MProve and MProve+ with drawbacks have been covered in Section 3.

Some notable work on proof of reserves include, gOTzilla [8] proposed by Baldimtsi *et al.*, which is an interactive zero-knowledge protocol for proving disjunctive statements. Additionally, Chatzigiannis and Chalkias [17] proposed proof of assets for account-based blockchains such as Diem, formerly known as Libra [2].

There has been significant work in proof of liabilities (PoL), most notable ones being DAPOL [15] and DAPOL+ [24]. MProve-Nova can be used with DAPOL+ to provide a proof of solvency. Chalkias *et al.* [14] highlighted vulnerabilities in the implementation of PoL used in production. Chatzigiannis *et al.* [16] 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.

6 NOVA

In this section, we cover aspects of the Nova recursive SNARK [27] required to describe the MProve-Nova protocol. Nova introduced a non-interactive folding scheme for committed relaxed rank-1 constraint systems (R1CS) [27]. It consists of two main components: an *incrementally verifiable computation (IVC)* [46] scheme and a zkSNARK to prove knowledge of valid IVC proofs.

6.1 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 inputs $w_0, w_1, \ldots, w_{n-1}$ such that

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

As shown in Figure 1, this is achieved 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.

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. 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 Nova IVC scheme satisfies completeness and knowledge soundness. For more details on the Nova IVC scheme, we refer the reader to Section 5 of the Nova paper [27].

6.2 zkSNARK of 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) to prove knowledge of a valid IVC proof Π_n.

(1) The prover $\mathcal{P}_{zk}$ and verifier $\mathcal{V}_{zk}$ of the zkSNARK are given the instance (n, z_0, z_n).

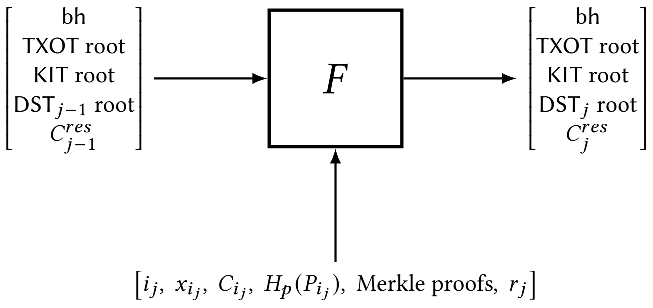

bh
TXOT root
KIT root
DST_{j-1} root
C_{j-1}^{res}
F
bh
TXOT root
KIT root
DST_j root
C_j^{res}
$[i_j,\ x_{i_j},\ C_{i_j},\ H_p(P_{i_j}),\ \text{Merkle proofs},\ r_j]$

Figure 2: Step Function for PoR protocol

(2) The prover $\mathcal{P}_{zk}$ additionally takes the proving key pk and IVC proof Π_n to produce the proof π.

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

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

If the zkSNARK is based on a Pedersen commitment scheme for vectors, then the proof size is $O(\log|F|)$ and the proof verification time is $O(|F|)$. Here $|F|$ is the number of R1CS constraints needed to express the computation of the step function F.

7 MPROVE-NOVA POR PROTOCOL

MProve-Nova requires the specification of a Nova step function F that will prove ownership of an unspent Monero output (see definition in Section 2.4) at each step and accumulate the coins in the output into a running sum. The necessity of the different components in F will become apparent when they are mapped to the requirements of any privacy-preserving Monero PoR protocol. These requirements are listed below:

R1. An exchange must only use outputs present on the Monero blockchain to contribute to its reserves.
R2. An exchange must only use outputs it owns to contribute to its reserves.
R3. An exchange must not use spent outputs to contribute to its reserves.
R4. An exchange must use each owned output at most once to contribute to its reserves.

7.1 Merkle Trees

To satisfy these requirements, we require one regular Merkle tree and two indexed Merkle trees [5]. The latter type of trees were introduced in [45] to generate efficient non-membership proofs inside a SNARK.

The regular Merkle tree is called *transaction outputs tree* (TXOT) and the two indexed Merkle trees are called *key images tree* (KIT) and *double spend tree* (DST). TXOT is constructed using one-time address, Pedersen commitment to amount and hash of one-time address. The purpose of TXOT is to ensure that the exchange only uses outputs which it owns and those outputs are present on the Monero blockchain (satisfying requirements R1 and R2). KIT is constructed using all the key images which have appeared on the Monero blockchain and its main purpose is to prevent the exchange from using a spent output (satisfying requirement R3). DST is constructed using private keys owned by the exchange and block height. Its main purpose is to prevent the exchange from using an owned output more than once (satisfying requirement R4).

All three trees are constructed using the Poseidon hash function [22] to reduce the number of R1CS constraints used to express F. 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 details regarding how these trees are used to satisfy requirements R1-R4 is explained in subsequent sections. We will now explain the construction of each of the three trees in detail.

(1) **Transaction Outputs Tree**: Let $H_p : \mathbb{G} \mapsto \mathbb{G}$ be the cryptographic hash function used to compute the key image of a one-time address $P = xG$ as $xH_p(P)$. Let $\|$ denote the string concatenation operator.
For a Monero block height bh, let $\mathcal{T}_{\mathsf{bh}} = \{(P_1, C_1), (P_2, C_2), \ldots\}$ be the set of all transaction outputs that have appeared in blocks up to height bh. TXOT is constructed using the leaves $\{H_{pos}(P\|C\|H_p(P)) \mid (P, C) \in \mathcal{T}_{\mathsf{bh}}\}$, sorted in the order of appearance of the one-time addresses (the P's) on the blockchain. We omit the dependence of TXOT on bh for simplicity. The motivation for choosing this particular leaf structure is described in Section 7.3.2.
(2) **Key Images Tree**: Let $\mathcal{I}_{\mathsf{bh}}$ be the set of all key images that have appeared on the Monero blockchain up to block height bh. KIT is constructed using the leaves $\{H_{pos}(I) \mid I \in \mathcal{I}_{\mathsf{bh}}\}$, sorted in the order of appearance of the key images on the blockchain. Once again, we omit the dependence of KIT on bh for simplicity.
(3) **Double Spend Tree**: While the previous two trees are populated before the MProve-Nova protocol execution, DST is populated during the protocol execution. DST is initially empty and is later populated with leaves of the form $H_{pos}(x\|\mathsf{bh})$ where x is a private key. Let DST_{j-1} and DST_j denote the state of the double spend tree before and after the jth step, respectively.

7.2 Step Function Inputs and Outputs

In this section, we will explain the inputs and outputs of the step function as shown in Figure 2. Let n be the number of outputs the

Algorithm 1: Overview of MProve-Nova PoR protocol

```
Input  : Public input $z_0$ and private inputs
         $\{w_0, w_1, \cdots, w_{n-1}\}$
Output : Public output after $n$ steps $z_n$
1 foreach $j \in \{1, 2, \cdots, n\}$ do
2     Prove ownership of one-time address $P_{i_j}$
3     Prove membership of $P_{i_j}$ in transactions outputs tree
        TXOT
4     Compute the key image $I_{i_j}$ corresponding to $P_{i_j}$
5     Prove non-membership of $I_{i_j}$ in key images tree KIT
6     Prove non-membership of the private key $x_{i_j}$
        corresponding to $P_{i_j}$ in the double spend tree $\mathsf{DST}_{j-1}$
7     Insert a leaf corresponding to $x_{i_j}$ into $\mathsf{DST}_{j-1}$ to get
        $\mathsf{DST}_j$
8     Accumulate the amount commitment corresponding to
        $P_{i_j}$ into $C^{res}_{j-1}$ to get $C^{res}_j$
9 end
```

exchange will use to contribute to its reserves. For $j = 1, 2, \ldots, n$, the function F takes the following as **public inputs** z_{j-1} in step j:

(i) The block height bh.
(ii) The root hash of TXOT.
(iii) The root hash of KIT.
(iv) The root hash of DST_{j-1}, double spend tree before step j.
(v) A Pedersen commitment C^{res}_{j-1} to the reserves accumulated before step j. C^{res}_0 is set to G, which commits to the zero amount with blinding factor 1.

The **public outputs** z_j of F in step j are the same as above, except for two differences.

(i) The root hash of DST_{j-1} is replaced with the root hash of DST_j, the double spend tree *after* step j.
(ii) The Pedersen commitment C^{res}_{j-1} is replaced with C^{res}_j, the reserves accumulated *after* step j.

F also takes the following as **private inputs** w_{j-1} in step j:

(i) An index i_j such that $1 \le i_j \le |\mathcal{T}_{\mathsf{bh}}|$.
(ii) The private key x_{i_j} corresponding to the one-time address P_{i_j}.
(iii) The values C_{i_j} and $H_p(P_{i_j})$ in the preimage of the leaf $H_{pos}\left(P_{i_j} \| C_{i_j} \| H_p(P_{i_j})\right)$ of TXOT at index i_j.
(iv) A Merkle proof of a leaf in TXOT to prove the membership of the leaf $H_{pos}\left(P_{i_j} \| C_{i_j} \| H_p(P_{i_j})\right)$ in TXOT.
(v) A second Merkle proof of a leaf in KIT to prove the non-membership of the Poseidon hash of the key image $I_{i_j} = x_{i_j} H_p(P_{i_j})$ in the KIT.
(vi) A third Merkle proof of a leaf in DST_{j-1} to prove the non-membership of the leaf $H_{pos}(x_{i_j} \| \mathsf{bh})$ in DST_{j-1}.
(vii) A fourth Merkle proof in DST_{j-1} to aid the insertion of the leaf $H_{pos}(x_{i_j} \| \mathsf{bh})$.
(viii) A random scalar r_j to blind the sum $C^{res}_{j-1} + C_{i_j}$.

7.3 Step Function Computation

Algorithm 1 gives a high level overview of MProve-Nova PoR protocol. For each $j \in \{1, 2, \ldots, n\}$, in step j function F executes the following substeps:

S1. Using the private input x_{i_j}, it computes the point $P_{i_j} = x_{i_j} G$. This computation proves knowledge of the private key corresponding to P_{i_j}.
S2. Using P_{i_j} and the private inputs C_{i_j} and $H_p(P_{i_j})$, it computes the hash $H_{pos}\left(P_{i_j} \| C_{i_j} \| H_p(P_{i_j})\right)$.
S3. Using the first private Merkle proof input, it proves the membership of $H_{pos}(P_{i_j} \| C_{i_j} \| H_p(P_{i_j}))$ in TXOT.
S4. Using the private inputs x_{i_j} and $H_p(P_{i_j})$, it computes the key image $I_{i_j} = x_{i_j} H_p(P_{i_j})$ and its hash $H_{pos}(I_{i_j})$.
S5. Using the second private Merkle proof input, it proves the non-membership of $H_{pos}(I_{i_j})$ in KIT.
S6. Using the private input x_{i_j}, it computes the hash $H_{pos}(x_{i_j} \| \mathsf{bh})$.
S7. Using the third private Merkle proof input, it proves the non-membership of $H_{pos}(x_{i_j} \| \mathsf{bh})$ in DST_{j-1}.
S8. Using the fourth private Merkle proof input, it inserts $H_{pos}(x_{i_j} \| \mathsf{bh})$ into DST_{j-1} to obtain the root hash of the new tree state DST_j.
S9. Using the private input r_j, it computes the point $C^{rand}_j = r_j G$.
S10. Using the private input C_{i_j}, public input C^{res}_{j-1}, and C^{rand}_j, it computes the updated Pedersen commitment $C^{res}_j = C^{res}_{j-1} + C_{i_j} + C^{rand}_j$.

The public outputs after the nth step contain C^{res}_n and the root hash of DST_n.

- The exchange will claim that C^{res}_n is a Pedersen commmitment to its total reserves. The exchange can generate a range proof on the point $C^{res}_n - tH$ to prove that its reserves exceed a threshold t.
- The root hash of DST_n will be used as input to the non-collusion protocol.

7.3.1 Satisfying Monero PoR requirements. Let us revisit the requirements listed at the beginning of this section.

(i) Steps S2 and S3 prove that C_{i_j} is a Pedersen commitment to coins tied to the one-time address P_{i_j} present on the Monero blockchain. Steps S9 and S10 ensure that only such C_{i_j}'s contribute to the total reserves of the exchange. Thus the exchange can only use outputs from the Monero blockchain to contribute to its reserves (requirement R1).
(ii) Step S1 proves that the exchange knows the private key x_{i_j} corresponding to P_{i_j}. Thus requirement R2 is satisfied, i.e. the exchange only uses outputs it owns to contribute to its reserves.
(iii) Steps S4 and S5 prove that the exchange can only use unspent outputs to contribute to its reserves, thus satisfying requirement R3. If an exchange were to use a spent output, the corresponding key image would be present in KIT and the non-membership proof in S5 would fail.
(iv) Steps S6, S7, S8 prove that the coins tied to the one-time address P_{i_j} are used at most once to contribute to the exchange's reserves, thus satisfying requirement R4.

- Suppose an exchange attempts to use the same output twice, in steps j_1 and j_2 where $j_2 > j_1$.
- It would have to repeat the private key x to satisfy the membership proof in step S3.
- The leaf $H_{pos}(x\|\mathsf{bh})$ would be inserted into the double spend tree in substep S8 of step j_1 to get DST_{j_1}.
- Then the non-membership proof of $H_{pos}(x\|\mathsf{bh})$ in DST_{j_2-1} would fail in substep S7 of the later step j_2.

7.3.2 Motivating the Step Function Structure. In this subsection, we justify our choices for the various data structures and computations used to specify the step function.

(i) We chose the TXOT to have leaves of the form $H_{pos}\left(P_{i_j}\|C_{i_j}\|H_p(P_{i_j})\right)$. It may not be clear why we chose to concatenate C_{i_j} and $H_p(P_{i_j})$ with P_{i_j}.
   - The reason for including C_{i_j} in the leaf is to enforce a *copy constraint*. We want to ensure that only the commitment C_{i_j} tied to P_{i_j} contributes to the reserves in step j. It is possible for users to know the amounts and blinding factors of commitments tied to one-time addresses they *do not own* (see Section 2.3). We want to avoid situations where commitments not owned by the exchange are used to generate its reserves.
   - The point $H_p(P_{i_j})$ is included in the leaf to reduce the number of R1CS constraints needed to specify the step function F. If it were not included, then the exchange would have to calculate the hash of P_{i_j} in the step function. This hash involves the Keccak hash function [9], which requires a large number of R1CS constraints.

(ii) We chose to insert leaves of the form $H_{pos}(x_{i_j}\|\mathsf{bh})$ into the DST to prevent an exchange from using an owned output more than once. Such behavior could also be possibly prevented by inserting any one of $H_{pos}(x_{i_j})$, $H_{pos}(P_{i_j})$ or $H_{pos}(P_{i_j}\|\mathsf{bh})$ into the DST.
   - The problem with using $H(P_{i_j})$ or $H_{pos}(P_{i_j}\|\mathsf{bh})$ as leaves to the DST is that an adversary could try combinations of outputs to reconstruct the set of owned outputs from the public root hash of the DST. This attack is feasible for small n as the number of outputs on the Monero blockchain is less than 100 million [48, 51].
   - The problem with using $H(x_{i_j})$ as leaves to 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 reserves at *different* block heights. This leaks information about the asset strategy of the exchange.

(iii) In substep S10 of step j, the Pedersen commitment to the accumulated reserves C_j^{res} is blinded using a random point C_j^{rand}. The point C_j^{rand} is chosen to have the form r_jG to ensure that the amount being contributed by C_{i_j} is unchanged (see Section 2.3). If this blinding point were not present and the accumulated reserves were simply calculated as $C_j^{res} = C_{j-1}^{res} + C_{i_j}$, then an adversary could try combinations of commitments to reconstruct the set of owned outputs from the public output C_n^{res}.

8 PROTOCOL FOR PROVING NON-COLLUSION

We propose a protocol for proving non-collusion between a pair of exchanges that leverages Nova to achieve constant verification time. The proof of non-collusion is publicly-verifiable in the sense that regular users can check the proof. The users need to download the root hashes of the DSTs from a pair of exchanges and a constant-sized Nova proof. For this protocol to work, the exchanges need to generate their respective MProve-Nova proofs (which contains their DSTs) at the same block height bh.

To generate a proof of non-collusion, one of the exchanges will have to share values $H_{pos}(x_{i_j}\|\mathsf{bh})$ which were inserted into the DST in step j of the PoR protocol with the other exchange. We argue in Section 9.2, that this does not affect the exchange's privacy, except for revealing the number of outputs it used to generate the MProve-Nova proof.

Suppose two exchanges Ex1 and Ex2, that have generated MProve-Nova proofs of reserves at the same blockheight bh, want to prove that they have not colluded. They want to prove that they have not used any common outputs to contribute to their respective reserves. Consider the following notation:

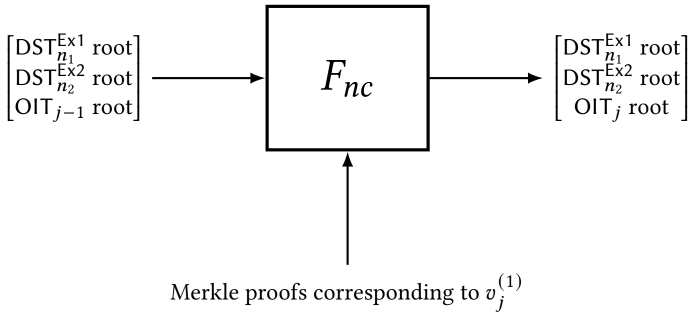
(i) Let n_1 and n_2 be the number of owned outputs of Ex1 and Ex2, respectively.
(ii) Let $\mathsf{DST}_{n_1}^{\mathsf{Ex1}}$ and $\mathsf{DST}_{n_2}^{\mathsf{Ex2}}$ be the double spend trees obtained by the exchanges at the end of their MProve-Nova proof procedures.
(iii) Let $v_1^{(1)}, v_2^{(1)}, \ldots, v_{n_1}^{(1)}$ be the leaves of $\mathsf{DST}_{n_1}^{\mathsf{Ex1}}$ and $v_1^{(2)}, v_2^{(2)}, \ldots, v_{n_2}^{(2)}$ be the leaves of $\mathsf{DST}_{n_2}^{\mathsf{Ex2}}$.
(iv) Note 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 owned by Ex1 and Ex2, respectively.

The non-collusion protocol requires Ex2 to share the leaves $v_1^{(2)}, v_2^{(2)}, \ldots, v_{n_2}^{(2)}$ of its double spend tree with Ex1. The Nova step function F_{nc} for the non-collusion protocol will be **run by** Ex1 for n_1 steps. The roles of Ex1 and Ex2 can be interchanged, but then Ex1 will have to share the leaves of its double spend tree with Ex2. In our protocol description, we assume that Ex1 generates the non-collusion proof.

The protocol requires an indexed Merkle tree called the *output inclusion tree* OIT. It is initially empty and is later populated with the leaves of Ex1's double spend tree $\mathsf{DST}_{n_1}^{\mathsf{Ex1}}$. Let OIT_{j-1} and OIT_j denote the state of the output inclusion tree before and after the jth step, respectively.

The role of the OIT is to enforce a constraint on the outputs, similar to the role of the double spend tree DST in the PoR protocol. While the DST ensures that an exchange does not use an output twice while proving its reserves, the OIT ensures that Ex1 uses the *same* outputs in the PoR protocol and the non-collusion protocol.

Consider the scenario where Ex1 and Ex2 used the same output (P, C) to generate their respective MProve-Nova proofs. At the jth step of the non-collusion protocol, the leaf $v_j^{(1)}$ is inserted into the tree OIT_{j-1}. After n_1 steps, the root hash of OIT_{n_1} is checked to



$\begin{bmatrix}\mathsf{DST}^{\mathsf{Ex1}}_{n_1}\text{ root}\\ \mathsf{DST}^{\mathsf{Ex2}}_{n_2}\text{ root}\\ \mathsf{OIT}_{j-1}\text{ root}\end{bmatrix}$
F_{nc}
$\begin{bmatrix}\mathsf{DST}^{\mathsf{Ex1}}_{n_1}\text{ root}\\ \mathsf{DST}^{\mathsf{Ex2}}_{n_2}\text{ root}\\ \mathsf{OIT}_{j}\text{ root}\end{bmatrix}$
Merkle proofs corresponding to $v_j^{(1)}$

Figure 3: Step Function for Non-Collusion protocol

be equal to the root hash of $\mathsf{DST}^{\mathsf{Ex1}}_{n_1}$. This equality ensures that all the outputs Ex1 used to generate its PoR proof were included while generating the non-collusion proof. Without this check, Ex1 could run the non-collusion protocol for $n_1 - 1$ steps and exclude the shared output (P, C).

Checking for equality between the root hashes of OIT_{n_1} and $\mathsf{DST}^{\mathsf{Ex1}}_{n_1}$ also ensures that Ex1 does not use invalid leaves $v_j^{(1)}$ while proving non-collusion. Invalid leaves were not possible in the PoR protocol because we checked membership of the corresponding outputs in the transaction outputs tree and non-membership of their key images in the key images tree. The non-collusion protocol can avoid repeating these validity checks by simply checking that the root hashes of OIT_{n_1} and $\mathsf{DST}^{\mathsf{Ex1}}_{n_1}$ are equal.

8.1 Step Function Inputs and Outputs

In this section, we will explain the inputs and outputs of the step function for the non-collusion protocol, as shown in Figure 3. It takes the following as **public inputs** z_{j-1} in step j:

(i) The root hashes of the double spend trees $\mathsf{DST}^{\mathsf{Ex1}}_{n_1}$ and $\mathsf{DST}^{\mathsf{Ex2}}_{n_2}$.
(ii) The root hash of OIT_{j-1}, the output inclusion tree *before* step j.

The **public outputs** z_j of F_{nc} in step j are the following:

(i) The root hashes of the double spend trees $\mathsf{DST}^{\mathsf{Ex1}}_{n_1}$ and $\mathsf{DST}^{\mathsf{Ex2}}_{n_2}$.
(ii) The root hash of OIT_j, the output inclusion tree *after* step j.

The step function F_{nc} also takes the following as **private inputs** w_{j-1} in step j:

(i) A Merkle proof to prove the membership of the leaf $v_j^{(1)}$ in $\mathsf{DST}^{\mathsf{Ex1}}_{n_1}$.
(ii) A second Merkle proof to prove non-membership of $v_j^{(1)}$ in $\mathsf{DST}^{\mathsf{Ex2}}_{n_2}$.
(iii) A third Merkle proof to prove the non-membership of the leaf $v_j^{(1)}$ in OIT_{j-1}.
(iv) A fourth Merkle proof in OIT_{j-1} to aid the insertion of the leaf $v_j^{(1)}$ into it.

Algorithm 2: Overview of non-collusion protocol

```
Input  : Public input $z_0$ and private inputs
         $\{w_0, w_1, \cdots, w_{n_1-1}\}$
Output : Public output after $n_1$ steps $z_{n_1}$
1 foreach $j \in \{1, 2, \cdots, n_1\}$ do
2     Prove membership of $v_j^{(1)}$ in $\mathsf{DST}^{\mathsf{Ex1}}_{n_1}$
3     Prove non-membership of $v_j^{(1)}$ in $\mathsf{DST}^{\mathsf{Ex2}}_{n_2}$
4     Prove non-membership of $v_j^{(1)}$ in $\mathsf{OIT}_{j-1}$
5     Insert $v_j^{(1)}$ in $\mathsf{OIT}_{j-1}$
6 end
7 Check that root hash of $\mathsf{OIT}_{n_1}$ equals root hash of $\mathsf{DST}^{\mathsf{Ex1}}_{n_1}$
```

8.2 Step Function Computation

Algorithm 2 gives a high level overview of non-collusion protocol. For $j \in \{1, .., n_1\}$, in step j, function F_{nc} executes the following substeps:

S1. Using the first private Merkle path input, it proves the membership of $v_j^{(1)}$ in $\mathsf{DST}^{\mathsf{Ex1}}_{n_1}$.
S2. Using the second private Merkle path input, it proves the non-membership of $v_j^{(1)}$ in $\mathsf{DST}^{\mathsf{Ex2}}_{n_2}$.
S3. Using the third private Merkle path input, it proves the non-membership of $v_j^{(1)}$ in OIT_{j-1}.
S4. Using the fourth private Merkle path input, it inserts $v_j^{(1)}$ into OIT_{j-1} to obtain the root hash of the new tree state OIT_j.

After step n_1, the root hash of OIT_{n_1} is checked to be equal to the root hash of $\mathsf{DST}^{\mathsf{Ex1}}_{n_1}$.

8.3 Motivating the Step Function Structure

The motivation for the different substeps in F_{nc} is as follows:

(i) Steps S1 and S2 prove that an output used by Ex1 is not used by Ex2.
(ii) Steps S3 and S4 prove that the output $v_j^{(1)}$ is distinct from $v_1^{(1)}, v_2^{(1)}, \ldots, v_{j-1}^{(1)}$.

9 SECURITY PROPERTIES

In this section, we discuss the security properties of MProve-Nova and non-collusion protocols. We model computationally bounded entities as probabilistic polynomial-time (PPT) algorithms.

9.1 Security Properties of MProve-Nova

The MProve-Nova PoR protocol has three properties. It is inflation resistant, preserves exchange privacy and it is collusion resistant. We will now analyze each of these properties.

9.1.1 Inflation Resistance. The inflation resistance property prevents a PPT exchange from creating a commitment to an amount which is greater than its total reserves. At each step j of the protocol, the exchange is allowed to accumulate the commitment C_{i_j} into the total commitment C_j^{res} if and only if

(i) it knows the private key corresponding to P_{i_j},
(ii) the output corresponding to (P_{i_j}, C_{i_j}) is unspent, and
(iii) the output corresponding to (P_{i_j}, C_{i_j}) has not been used by the exchange before.

These properties are enforced by the circuit of Nova step computation described in Section 7. We rely on knowledge soundness of Nova and the discrete log assumption. Since Nova is knowledge sound and these properties are enforced inside circuit, a PPT exchange cannot commit to an amount which is greater than its total reserves and submit a proof such that it is accepted by the Nova verifier.

9.1.2 Exchange Privacy. The exchange privacy property prevents a PPT adversary from identifying the Monero outputs which the exchange used to generate the proof of reserves. In the PoR protocol, the exchange produces an IVC proof Π_n that attests to the correct execution of steps $1, ..., n$. Then the exchange uses the prover $\mathcal{P}_{zk}$ to produce a proof π to show that it knows a valid IVC proof Π_n for the statement (z_0, z_n), where z_0 is the initial public input and z_n is the public output after the nth step. The proof π is submitted to the verifier or customer along with the statement (z_0, z_n). As described in Section 6, the proof π is zero-knowledge and it reveals nothing about the exchange's private inputs to the protocol.

The initial input z_0 is independent of the exchange's private inputs and thus reveals nothing about the exchange's private inputs. The final output of the protocol z_n depends on the exchange's private inputs as

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

We need to show that the output z_n does not reveal any information about the exchange's private input. The output z_n after n steps is as follows:

$$z_n = [\mathsf{bh},\ \mathsf{TXOT}\text{ root},\ \mathsf{KIT}\text{ root},\ \mathsf{DST}_n\text{ root},\ C_n^{res}]$$

The block height bh, root hash of TXOT and root hash of KIT are computed using public data available on the Monero blockchain and reveal no information about the exchange's private input. The root hash of DST_n is computed using exchange's private input. Using the random oracle assumption, the root hash will be distributed uniformly and revealing it does not reveal any information about the exchange's private input. Finally, the Pedersen commitment to total reserves C_n^{res} is perfectly hiding which also does not reveal any information about the exchange's private input. Thus, the verifier or customer given the proof π and the statement (z_0, z_n) learns nothing about the exchange's private input.

9.1.3 Collusion Resistant. The collusion resistance property prevents two exchanges from colluding to generate the proof of reserves. The output of the MProve-Nova PoR can be used as input to a protocol for proving non-collusion. This property follows from the construction of our non-collusion protocol. The protocol guarantees to detect collusion provided that the two exchanges giving the proof of non-collusion have generated the proof of reserves at the same blockheight.

9.2 Security Properties of Non-Collusion protocol

In this section, we will analyze the exchange privacy property of the non-collusion protocol.

9.2.1 Exchange Privacy. Suppose two exchanges Ex1 and Ex2 have generated the proof of reserves for the same blockheight bh and Ex1 wants to prove non-collusion with Ex2. As described in Section 8, Ex2 will have to share the values $\left[v_j^{(2)}\right]_{j=1}^{j=n_2}$ which were inserted in its $\mathsf{DST}^{\mathsf{Ex2}}$ with Ex1. Ex1 will run the non-collusion protocol for n_1 steps and submit the proof π along with the statement (z_0, z_{n_1}) to the verifier or customer. The proof π is zero-knowledge and does not reveal any information about the private input of Ex1. The initial input z_0 is independent of private input of Ex1. The output z_{n_1} after n_1 steps is as follows:

$$z_{n_1} = [\mathsf{DST}_{n_1}^{\mathsf{Ex1}}\text{ root},\ \mathsf{DST}_{n_2}^{\mathsf{Ex2}}\text{ root},\ \mathsf{OIT}_{n_1}\text{ root}]$$

The root hash of $\mathsf{DST}_{n_2}^{\mathsf{Ex2}}$ does not depend on private input of Ex1. Using the random oracle assumption, the root hashes of $\mathsf{DST}_{n_1}^{\mathsf{Ex1}}$ and OIT_{n_1} will be distributed uniformly and do not reveal any information about the private input of Ex1. Thus the protocol has no effect on the privacy of the exchange proving non-collusion i.e. Ex1.

Ex2 has to reveal the values $\left[v_j^{(2)}\right]_{j=1}^{j=n_2}$ to Ex1 so that Ex1 can run the non-collusion protocol. Since $v_j^{(2)} = H_{pos}\left(x_j^{(2)} \| \mathsf{bh}\right)$, using the random oracle assumption $v_j^{(2)}$ will be distributed uniformly and do not reveal any information about the private input of Ex2 except the number of outputs owned by Ex2 i.e. n_2.

10 IMPLEMENTATION AND PERFORMANCE

We have implemented the MProve-Nova PoR protocol and the protocol for proving non-collusion using our modified implementation of Nova. The implementation uses Pasta curves [52], a 2-cycle of elliptic curves and the Posiedon hash function [22, 29]. A detailed explanation of Nova using 2-cycle of elliptic curves and a security fix in the implementation was given by Nguyen *et al.* [39]. The Nova implementation accepts the step computation F, as a bellpepper gadget [30]. The bellpepper gadgets required to implement MProve-Nova were described in Section 4.

In this section, we present the performance of the MProve-Nova PoR protocols. All simulations were run on a 64 core 2.30GHz Intel Xeon Gold 6314U CPU with access to 125GB RAM. Table 1

Table 1: Performance comparison of PoR protocols. $n = |\mathcal{P}_{own}|$**, 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.**

	MProve-Nova			*MProve+*			*MProve*		
n	PT(Hrs)	VT(s)	PS(KB)	PT(Hrs)	VT(s)	PS(KB)	PT(s)	VT(s)	PS(MB)
1,000	0.66	4.54	27.37	2.49	959.05	162.62	99.32	17.36	37.44
3,000	1.99	4.50	27.36	7.46*	2887.54*	482.81*	100.19	17.33	37.44
5,000	3.36	4.53	27.37	12.44*	4817.22*	803.02*	100.21	17.37	37.44
7,000	4.75	4.57	27.36	17.41*	6746.34*	1123.2*	100.27	17.48	37.44
10,000	6.94	4.48	27.36	24.88*	9640.44*	1603.5*	100.19	17.34	37.44
15,000	10.92	4.47	27.36	37.32*	14463.939*	2404*	100.39	17.35	37.44
20,000	15.20	4.41	27.36	49.76*	19287.439*	3204.5*	100.36	17.33	37.44

shows comparison between MProve-Nova, MProve+, and MProve for PoR generation/verification. The open source implementations of MProve+ [7] and MProve [6] were used to perform the simulations. Our implementation of MProve-Nova is available on anonymous Github [38].

The computation times and proof sizes of MProve+ and MProve increase linearly in the size of the anonymity set. All simulations for MProve+ and MProve were run for an anonymity set of 45,000 one-time addresses, whereas for MProve-Nova the anonymity set is the set of all one-time addresses on the Monero blockchain. Thus the comparison is unfair towards MProve-Nova since for an anonymity set of all the one-time addresses, MProve+ and MProve will be impractical. However, despite this, MProve-Nova gives practical results and performs better in terms of verification times and proof sizes.

For MProve-Nova, the proving time is linear in the number of owned one-time addresses, while the proof verification times and proof sizes are constant. The proving time for 10,000 owned addresses is about 7 hours. The verification time and proof size are constant at about 4.5 s and 27 KB, respectively. The performance results are practical since an exchange with up to 10,000 owned addresses can generate proofs every 12 Hrs and an exchange with up to 20,000 owned addresses can generate proofs every 24 Hrs.

MProve-Nova has smaller proof sizes and faster verification time as compared to both MProve+ and MProve. MProve-Nova has higher proving time compared to MProve but it provides better privacy to the exchange which is not the case with MProve.

Table 2 shows the performance of the non-collusion protocol. In all cases, the proof size is 23 KB and proof verification time is about 219 ms.

Table 2: Performance of Proof of Non-Collusion. n_1 **denotes the number of values inserted by exchange Ex1 in its double spend tree** $\mathsf{DST}^{\mathsf{Ex1}}$

n_1	**Proving Time**	**Verification Time**	**Proof Size**
1,000	359.66 s	209.30 ms	23.18 KB
3,000	1165.06 s	222.85 ms	23.18 KB
5,000	2373.40 s	219.54 ms	23.18 KB
7,000	3944.93 s	231.37 ms	23.18 KB
10,000	6407.71 s	218.58 ms	23.18 KB
15,000	12423.20 s	234.17 ms	23.18 KB
20,000	19824.15 s	246.92 ms	23.18 KB

11 CONCLUSION

We described MProve-Nova, the first privacy-preserving proof of reserves for Monero which also enables proofs of non-collusion between exchanges. We compared MProve-Nova with MProve+ and MProve to show that our protocol has practical computation times and proof size along with better privacy.

For MProve-Nova, the proving time for 10,000 owned addresses is about 7 hours. It achieves verification times of about 4.5 seconds and proof sizes of 27 KB, irrespective of the size of the anonymity set. Our goal with the performance evaluation is to provide an upperbound on the computation times since they can be further reduced using GPU acceleration or using non-uniform IVC schemes like SuperNova [26].

The non-collusion protocol reveals the number of outputs owned by the exchange which shares the values inserted in its double spend tree 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 prove non-collusion such that the proof is publicly-verifiable is a direction for future work.

The MProve-Nova protocol does not construct the transactions output tree and key image tree inside a circuit. It works with the assumption that the tree roots which are input to the protocol are valid. Any deviation from the correct values for the tree roots can be detected by anyone with a copy of the Monero blockchain. The verification of the validity of these tree roots inside an R1CS circuit from Monero blockchain data is a challenging direction for future work.

REFERENCES

[1] 0xPARC. 2021. circom-ecdsa: Implementation of ECDSA operations in circom. https://github.com/0xPARC/circom-ecdsa

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

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

[4] 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.

[5] Aztec. 2023. Indexed Merkle Tree. https://docs.aztec.network/aztec/protocol/trees/indexed-merkle-tree

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

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

[8] 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

[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] Nalin Bhardwaj. 2022. Nova Scotia: Middleware to compile Circom circuits to Nova prover. https://github.com/nalinbhardwaj/Nova-Scotia

[11] 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

[12] 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

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

[14] 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

[15] 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

[16] 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

[17] 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

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

[19] 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

[20] 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

[21] 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

[22] 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

[23] iden3. 2024. Circom circuit compiler. https://github.com/iden3/circom

[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, 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

[28] 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

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

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

[31] 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

[32] 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

[33] Microsoft. 2021. Nova Implementation. https://github.com/microsoft/Nova

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

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

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

[37] 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

[38] MProve Nova. 2024. Implementation of MProve-Nova. https://anonymous.4open.science/r/5t384rtcbf57fkbvksdncoir893457022f674r3658h32y8cxny87/README.md

[39] 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

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

[41] 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

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

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

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

[45] 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

[46] 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

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

[48] 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

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

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

[51] 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

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

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