# Democracy Enhancing Technologies

Jeremy Clark

A Thesis in
The Concordia Institute for Information Systems Engineering (CIISE)

Presented in Partial Fulfillment of the Requirements
For the Degree of
Doctor of Philosophy
(Information and Systems Engineering)
at
Concordia University
Montréal, Québec, Canada Test

September 2023

# Concordia University
School of Graduate Studies

This is to certify that the thesis prepared

By: **Jeremy Clark**

Entitled: **Title**

and submitted in partial fulfillment of the requirements for the degree of

**Doctor of Philosophy (Information and Systems Engineering)**

complies with the regulations of this University and meets the accepted standards with respect to originality and quality.

Signed by the final examining committee:

_____ Chair
*Walter Lucia*

_____ External Examiner
*Kaiwen Zhang (ETS)*

_____ Examiner
*Amr Youssef*

_____ Examiner
*M. Mannan*

_____ Examiner
*Carol Fung*

_____ Supervisor
*Jeremy Clark*

Approved by _____
*Zachary Patterson, Graduate Program Director (CIISE)*

01 Sept 2023 _____
*Mourad Debbabi, Dean (GCS)*

# Abstract

Name: **Jeremy Clark**

Title: **Democracy Enhancing Technologies**

Hello. No more than 250 words.

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Diam donec adipiscing tristique risus nec feugiat in fermentum posuere. Et netus et malesuada fames ac turpis. Nullam non nisi est sit. Felis eget velit aliquet sagittis id. Mauris commodo quis imperdiet massa tincidunt. Tellus molestie nunc non blandit massa enim nec. Facilisis mauris sit amet massa. Et molestie ac feugiat sed. Metus vulputate eu scelerisque felis imperdiet proin.

# Acknowledgments

Hello.

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Diam donec adipiscing tristique risus nec feugiat in fermentum posuere. Et netus et malesuada fames ac turpis. Nullam non nisi est sit. Felis eget velit aliquet sagittis id. Mauris commodo quis imperdiet massa tincidunt. Tellus molestie nunc non blandit massa enim nec. Facilisis mauris sit amet massa. Et molestie ac feugiat sed. Metus vulputate eu scelerisque felis imperdiet proin.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

Focus on making proof of liabilities for the real world that can be used by marketplaces.

**Scope.** Blah blah blah.

**Contributions.** Our primary contributions are as follows.

1. Blah blah blah.
2. Blah blah blah.
3. Blah blah blah.

# Chapter 2

# Background

In the dynamic landscape of cryptocurrencies, ensuring transparency, accountability, and financial stability is paramount for marketplaces. This section provides an overview of the concepts and mechanisms necessary to follow along the conception of a daily proof of liabilities. We explore the various approaches adopted by leading exchanges to demonstrate their financial health through proof of reserves or solvency mechanisms. Additionally, we highlight the shortcomings and challenges associated with current solvency verification practices, mainly the lack of recurrent proof of reserves, paving the way for a daily proof in the subsequent sections.

## 2.1 Bitcoin

Bitcoin is recognized as the world's first successful cryptocurrency and decentralized digital currency. The goal of Bitcoin is to allow financial transactions to be settled

on its own, without the need of a middleman, typically financial institutions. Bitcoin is built on a peer-to-peer network, which means that every participant helps in safe keeping the transactions history, and propagating the new transactions. There is no single point of failure. This allows for transactions to occur in real time, in contrast to the delays encountered in the traditional finance world. Bitcoin defines two different concepts: Bitcoin the cryptocurrency, and Bitcoin the blockchain. The cryptocurrency resides on the blockchain. The Bitcoin blockchain is a decentralized ledger that records all Bitcoin (the cryptocurrency) transactions immutably and transparently. This blockchain serves as a verifiable record of all Bitcoin transactions, accessible to every participant in the network. The transparency afforded by the public blockchain engenders trust and accountability.

### 2.1.1 Transactions

For every participant of the network, there is a public key, a private key and a wallet address associated with the participant. The public key is derived from the private key using elliptic curve multiplication, and the wallet address is derived from the public key using a hashing function. Both are one way function, meaning you cannot derived the other way around. The public key serves as the unique identifier in the network, but it is the wallet address that typically defines a participant. The wallet address can be seen as a bank account number. When you send bitcoin to someone, you send it to their wallet address. To be able to send some bitcoin, you need to create a transaction and send it to the network. When transactions are sent on the

network, there is no way of knowing who propagated the transaction first. We need to make sure a transaction originates from the sender. The way to do that is to sign your transaction. The digital signature is created from the transaction data and the private key, which is only known by the owner of the address. The public key is then used to verify the authenticity of the signature. Sending a transaction is the easiest problem to solve. The real challenge is to keep track of who owns what, and to avoid the double spending problem. The methodology for managing this is to keep the history of every single transactions. The transactions are bundled up into blocks, and the chain of blocks create the blockchain.

### 2.1.2 Network

The challenge of the network, is to have every single node achieve consensus on the transaction history. Nodes are computers connected to the network, working on publishing new blocks. The nodes work collectively to establish order of transactions (sequencing). Every new transaction is broadcasted to all nodes. The nodes puts the transactions into a block, and try to publish that block. In order to publish a block, each node needs to solve a proof-of-work challenge. When a node solves the challenge, it broadcasts the block to every nodes. The nodes accepts the block if all transactions are valid. There is no formal way of approving a new block. A node show its acceptance by starting to work on a new block using the hash of the accepted block as previous hash. Some nodes might accept different blocks, if multiple blocks are propagated at the same time. This would create multiple chains. To solve this

issue, the longest chain is considered to be the correct one. If two chains have the same length, nodes keep working on their respective chains untill one of the chains receives a new block, breaking the tie.

### 2.1.3 Proof-of-work

In order to submit a new block, a node have to find a hash with a specific number of leading zero bits. It is exponentially more difficult for every zero bit. This cryptographic puzzle serves as a barrier to entry, ensuring that a lot of computational power was spent on creating the block. The way to test different hash values, is to change the block timestamp, and the nonce value. The nonce value is there solely for that purpose. Once a block is published, you cannot change any value inside of it because the hash value would change. The immutability of the older blocks is what makes Bitcoin secure. To modify a block in the middle of the chain, you would need to redo the work of every single blocks made after that. The longest chain is determined by the cumulative proof-of-work invest in it. This is why we say that Bitcoin is secured as long as 51 pourcent of the nodes are honest. The chain with the majority of nodes working on it will grow up the fastest, thus will be the accepted chain. The difficulty of the new block is determined by an average, in order to generate blocks at a steady pace. There is also a bitcoin reward associated with mining (publishing) a block.

### 2.1.4 Merkle Tree

In the architecture of the blockchain, only the merkle root is stored in the block header. Nodes only keep the recent blocks in memory. For the older blocks, they keep only the block header in memory. This storage ensures the integrity of the blockchain, while decreasing the memory required to have the full blockchain history. Since the hash of a block is the hash of the block header, this strategy does not impact the integrity checks of the blockchain. The merkle root is the top of the Merkle Tree, and is a unique identifier of the full tree. A merkle tree is a tree where the parent node is the hash of the child nodes. The tree is immutable because changing a single node would have an impact on the merkle root. For instance, in figure 2.1 changing the transaction 0 would change the hash 0, which in turn would change the hash 01, subsequently changing the root hash.

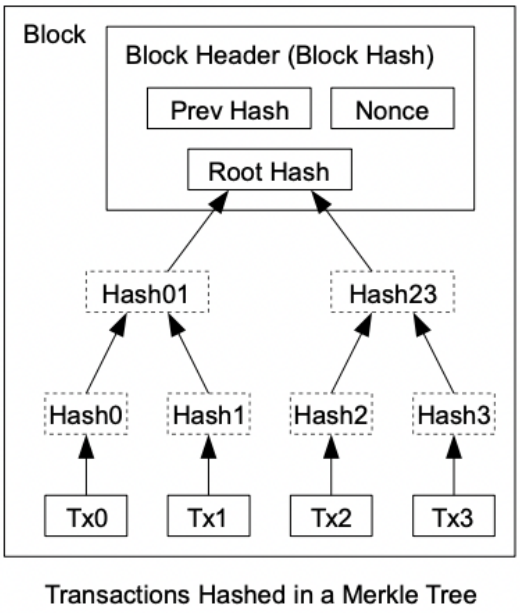


Figure 2.1: Bitcoin Merkle Tree [1]

## 2.2 Marketplaces

The best way to buy bitcoin for the first time is through marketplaces. Marketplaces facilitate the exchange of traditional currency for bitcoin. However, it is crucial to understand that the Bitcoin obtained remain into the custody of the platform, rather than being deposited to a personal wallet address. In order to gain custody of your bitcoins in your wallet, you need to enter a transfer request. This is similar to traditional finance, where you trust the bank (marketplace), to go ahead with your transaction. Once you have bitcoins in your wallet, you can transact on the network without needing a 3rd party. Unless you are running a node, you will need to trust a 3rd party, whether it is a marketplace, or over the counter, to first acquire bitcoin.

When the bitcoin you own is in custody of the marketplace, there is no way to see your bitcoin onchain. Marketplaces have many wallets, some are made public and some are not. While this allo for the privacy of your marketplace deposits and withdraws, it represent an fondamental contradiction. Bitcoin was designed to be transparent and public, without the need of a 3rd party to do a transaction. Not being able to track your bitcoin in the marketplace pauses a significant challenge. A user has no proof that the marketplace solvency to reimburse every client. However, this problem is being actively worked on. Marketplaces have begin to use proof of solvency (or proof of reserve) to demonstrate that they are solvent. While this is a step in the right direction, the current proof of solvency used by the marketplaces have many default, and they are not sufficient to prove that they are solvent.

## 2.3 Zero Knowledge

Zero-knowledge proofs is a cryptographic technique to prove some knowledge without divulging any informations.For instance, the classic way of proving that you know the solution to an equation, is to reveal the solution to the equation itself. However, with zero knowledge you are able to prove that you know the solution, without discolsing the solution. To construct a zero knowledge proof, you need to construct a proof that is sound and complete. You also need your proof to be zero knowledge[4].

- **Completeness**: If the statement is true, an honest verifier will be convinced by an honest prover.
- **Soundness**: If the statement is false, no dishonest prover can convince the honest verifier (except with some infinitysimal probability).
- **Zero-Knowledge**: If the statement is true, a verifier learns nothing other than the fact that the statement is true. [5]

### 2.3.1 Non interactive proofs

Zero knowledge proofs were originally designed as interactive, that is multiple rounds of interaction between the prover and the verifier [6]. leading to what are called interactive zero-knowledge proofs. This interaction allows the prover to demonstrate knowledge of the solution without revealing any additional information. An alternative model was then proposed where the verifier and prover use a reference string that is shared during a trusted setup. Once we have the reference string, a single message

is needed between the prover and the verifier. The elimination of multiple rounds of interaction simplifies the verification process and reduces the computational power required. Therefore, noninteractive zero-knowledge proofs offer enhanced efficiency and scalability, which will be needed later on. [7] [8]

### 2.3.2 SNARKS

One of the recent advancement for non-interactive proofs is what is known as SNARK (non-interactive argument of knowledge). This means a proof that is:

- **Succinct**: the size of the proof is very small compared to the size of the witness.
- **Non-interactive**: No rounds of interactions between the prover and the verifier.
- **Argument**: Secured only for provers with bounded computational ressources, that is a dishonest prover with unlimited computational power could prove a wrong statement.
- **Knowledge-sound**: If the statement is true, a verifier learns nothing other than the fact that the statement is true. [10]

Moreover, a SNARK can also be zero-knowedge, where the prover demonstrate knowledge without revealing any additional information about the witness. We call such proof a zk-SNARK.

### 2.3.3 Arithmetic circuit

Arithmetic circuits are a core component of SNARKS. An arithmetic circuit is a set gates, each assigned a distinct set of inputs corresponding to the numbers to be processed in the operation. These gates are configured to execute arithmetic operations such as addition, subtraction, multiplication, or division. The outputs of the gate circuit represent the digits of the resulting computation. This structure allows SNARKs to efficiently verify complex mathematical computations while preserving succinctness and scalability.

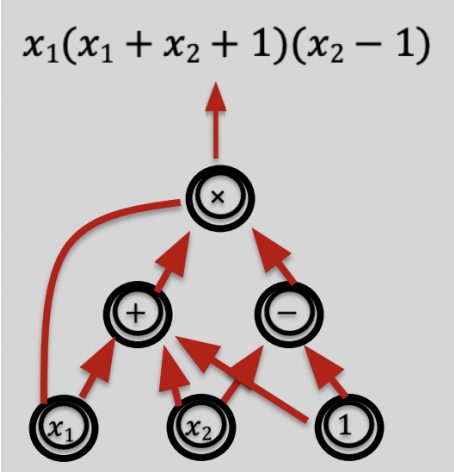


Figure 2.2: General Arithmetic circuit [**?**]

The prover's process in SNARKs is to create a proof using the setup parameter, a private witness, and public input. The proof shows that the arithmetic circuit is equal to 0. Using the same setup parameter and the public input, the verifier confirms the accuracy of the proof by making sure it aligns with the parameters.

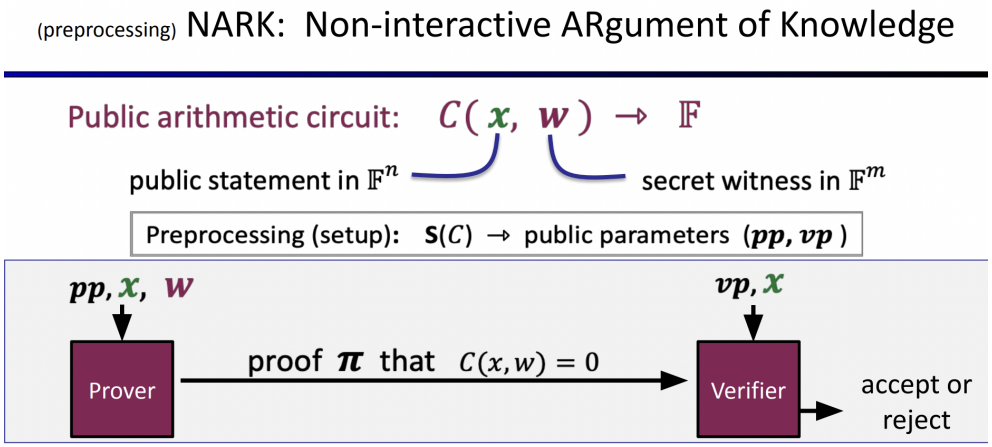


Figure 2.3: Arithmetic circuit for SNARK [?]

- **S(C)**: Public parameters (pp,vp) for prover and verifier
- **P(pp,x,w)**: Proof $\pi$
- **V(vp,x,$\pi$)**: Accept or reject
- **C(x,w)**: Arithmetic circuit
- **w**: Private witness
- **x**: Public input

## 2.4 Proof of solvency

A proof of solvency is a system where we prove that an entity, in most cases an exchange, hold enough assets to cover the balance of every customers. In traditional finance, this would be done thourgh an audit. While an audit can be usefell, it has many limitations. Obviously it requires the trust of another 3rd party, but it poses

also a time constraint. Thus, it is impractical, even impossible to hold an audit every single day, and in the bitcoin world things move fast. It is essential to be able to fill a prove of solvency every single day. The implementation of a mechanism to produce daily proofs of solvency is therefore of the utmost importance.

A proof of solvency is composed by a proof of assets, where we verify what assets the marketplace as control over, and the proof of liabilities, where we confirm that the total amount of user deposits is smaller than the marketplaces assets.

The first paper about a proof of solvency focuses only on the proof of liabilities. In his paper Gregory Maxwell addresses the issue of verifying the solvency of Bitcoin exchanges. [12] Maxwell's system ensures user privacy by maintaining confidentiality of individual account balances, while only revealing aggregate information in the proof of liabilities. This is achieved through the application of Merkle trees.

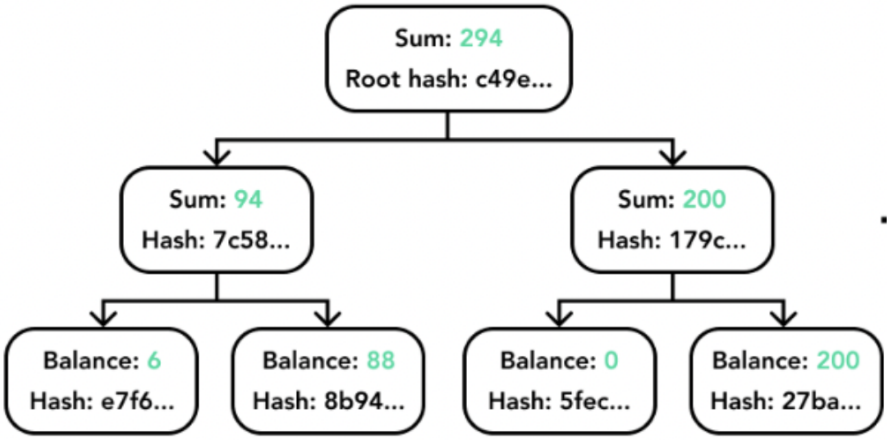


Figure 2.4: Merkle tree for proof of liabilities

In the Merkle tree, every node contains a user's balance along with a hash-based

commitment incorporating the customer ID and a nonce. The root of the tree is the sum of the balances. To verify their inclusion in the total liabilities, users receive a subset of the hash tree from the exchange. This subset includes the user's nonce and the sibling nodes along the unique path from the user's leaf node to the root. By comparing the received information to the exchange's broadcasted root node, users can confirm the inclusion of their balance. While elegant, the protocol does have privacy implications. The exact value of the exchange's total liabilities, published in the root node, may be sensitive data. Additionally, the proof of inclusion reveals the neighbor's balance, and the subtree's balance along the Merkle path.

The proof of liabilities is only part of the proof of solvency. A complete proof of solvency needs a proof of asset as well. Provision describes the first preserving privacy proof of asset [11].

In Provisions, the focus shifts towards preserving privacy while still proving ownership of assets. Instead of publicly demonstrating control over specific addresses, Provisions enables exchanges to prove ownership of an anonymous subset of addresses sourced from the blockchain.

Since these 2 papers, a lot of work was done to make the proof of solvency evolve. However, marketplaces still do not implement a proof of solvency, or a flawed and limited version of it.

### 2.4.1 Real world proof of solvency

In recent years, several major cryptocurrency exchanges, including Binance, Crypto.com, and Kraken, have taken steps to enhance transparency by provinding various proof of reserves. Binance has implemented a proof of reserves system where they publish a monthly Merkle tree as their proof of liabilities, and disclose a list of their assets [2]. Crypto.com published a one-time audit [13]. Kraken is also publishing a proof of liabilities every few months, without a proof of assets. [14].

Although these proof of reserves may seem promising initially, they are primarily superficial. The proofs have many shortcomings, they are not sufficient to prove that the marketplaces are solvent. The first concern is the lack of proof of assets, or in Binance's case the lack of evidence demonstrating control over the wallets. Without a reliable proof of assets, the proof of liabilities is worthless because it has nothing to compare against.

Moreover, the frequency of reporting is another area of concern. Given the dynamic nature of cryptocurrencies, a monthly report is not sufficient at all. Binance is the only marketplace describing their proof of solvancy, and we can see that it is not built with recurrence in mind. The proof is created from scrach every time. They would need 150 servers to produce a daily proof [3].

# Chapter 3

# Recursion proofs

This chapter serves as an intermediary background exploration, diving into more advanced concepts beyond last chapters concepts. The realm of zero knowledge is dynamic and continuously evolving, marked by ongoing advancements and active research endeavors. Among these advancements, recursion proofs is one of the most important and active subject. Recursion proofs are created to accelerate the generation of multiple zero-knowledge proofs. Not all recursion proofs are created equal, they can vary significantly and serve different use cases. In this section, we will examine these variations, distinguishing between them. We aim to identify the most suitable method for our daily proof of liabilities and proof of inclusion.

[15] for aggregation, recusrion and folding

## 3.1 Aggregation

Aggregation is the first and simples type of recursion for zk proofs. It comprises of 2 different phases. Initially, you create a standard individual proof for multiple blocks. In the second phase, a proof of proof is created. You could either merge all proofs into a single one, or do tree like structure where each parent node proove its child nodes.

This is a quite simple process. Each block as its own proof, and then you prove that the other proofs are valid, giving you only one prove to verify at the end. Since there is 2 different phase, 2 different circuits needs to be constructed. On the surface, you can parallize the initial proofs, which decrease the proving time, and you only have one proof to verify, which decrease the verfying time. However, there are still some issues with aggregation. The most important issue is that the proof time of the second circuit grows linearly with the number of blocks to verify, making it less scalable. [15]

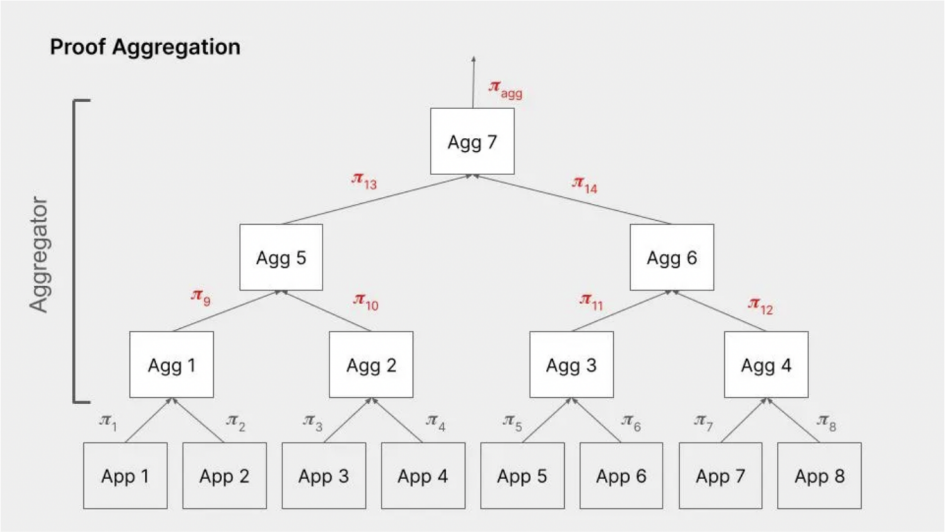


Figure 3.1: Aggregation Proof [16]

## 3.2 Recursion scheme

The next technique is the recursion scheme, or Incrementally Verifiable Computation (IVC). Like the aggregation, the proof is separated into blocks. However, in this case each block serves two purposes. Provings that the block itself is valid, and that the previous proof is valid. The number of constraints will always stay the same because it is only proving 2 things of fixed sizes. This creates a chain structure, where veryfing the latest block is the equivalent of veryfing every block in the chain. It is an improvement from the previous technique, because this allows to maintain a constant number of constraints in the curcuit. This implies that the latest does not take longer to verify thant the second block. However, each block block still has to verify the previous block, which takes additionnal. [15]

## 3.3 Accumulation

To mitigate the exponential growth of recursion proofs, the new scheme created by nova defers every computationnaly heavy tasks to the end. All the defered parts are accumulate at a later point. Unlike the aggregation scheme, the last part of accumulation does not grow with every new step. Instead of doing n times heavy work, we are doing it only once. The heavy part of the verification step in most SNARKS is found when opening the polynomial commitment.

### 3.3.1 Polynomial Commitments

A polynomial commitment is a cryptographic technique that allows to commit (lock) data, and reveal (unlock) it later.

The commitments are binding, meaning there is no way to alter data once it is commited. They are also significantly shorter than the data commited. Collisions are inevitable given the vastness of potential data sets mapped onto compact commitments (pigeonhole principle). However, it is computationally infeasible to find a second dataset matching the same commitment.

The commimtents can also be hidden. This means that no information is shared to the receiving party. An example of a commitment scheme would be to hash a dataset with a collision-resistant hash function such as SHA256, and the hash value is shared. To add the hiding property, a random string could be added at the beginning of the dataset.

Instead of commiting to a dataset, we could be commiting to a polynomial. While a commitment to the coefficient would work, we would be revealing the polynomial when opening the commitment. We would like to be able to open the commitment only to a certain point of the polynomial. We are also interested in keeping the polynomial secret. An interesting property of polynomials, is that is is evaluatable at specific points. A polynomial commitment is proving to another party that we have a commitment to a plynomial that evaluates to a certain value at a certain point.

Polynomials support linear opperations like addition or multiplication. A commitment can be additively homomorphic if the sum of the commitment values is equal to the sum of the polynomials.

Polynomial commitments are essentials to SNARKs. Verifying opening claims of polynomial commitments constitues the computationally intensive task we mentionned earlier. [**?**]

### 3.3.2 Halo accumulation

The concept of deferring the polynomial commitment opening checks, and consolidating them into a single operation, was introduced by Bowe, Grigg, and Hopwood Halo.[**?**] The task of checking an opening claim for a polynomial commitment is defined in two steps. The first part is fast, where you just output a pair of a polynomial and its commitment. The second part is expensive, where we verify the pair $(f_1, c_1)$ of polynomial $f_1$ and its commitment $c_1$. The second part can be accumulated if the commitment scheme is additively homomorphic. Instead of verifying each pair ($c_1$

is a commitment for $f_1$, $c_2$ is a commitment for $f_2$, etc.), we can verify the linear combination of every pairs ($c_1 + c_2 + ...$ is a commitment for $f_1 + f_2 + ...$). [**?**]

## 3.4 Folding scheme

Nova takes accumulation a setp further. Instead of accumulating the hard part at the end, the folding scheme accumulates everything up untill the end. The setup is similar to the recursion scheme, but instead of computing the proof of the previous block, the R1CS are folded together at every block. Instead of having $n$ set of R1CS, we are left with a single set. The new R1CS are called relaxed R1CS, and are used to compute a single proof at the end of the folding. [15] [**?**]

### 3.4.1 R1CS

Going back on the previous section, we saw the definition of an airthmetic circuit. R1CS is simply a representation of an arithmetic circuit. A R1CS constraint has the form $a * b = y$, where $a$,$b$ and $y$ are a combinations of variables. Lets define our previous circuit as a R1CS:

$$w_1 = x_1 + x_2 + 1$$

$$w_2 = x_2 - 1$$

$$x = x_1 * w_1 * w_2$$

Now the 3rd constraint does not follow the rules, because there are 3 variables being multiplied. We need to change it using an intermediary value:

$w_1 = x_1 + x_2 + 1$

$w_2 = x_2 - 1$

$w_3 = w_1 * w_2$

$x = x_1 * w_3$

[**?**]

If we define $z = (x, w)$, we can express the arithmetic circuit as $Az \circ Bz = Dz$ where $\circ$ can be define as: $(x_1, x_2) \circ y_1, y_2 = (x_1 y_1, x_2 y_2)$ [**?**]

### 3.4.2 Relaxed R1CS

The goal is to combine 2 R1CS and obtain another R1CS. If we are able to do that, we can fold every R1CS together and be left with only one.

If we **define our R1CS**:

**fix an R1CS** program $A, B, D \in \mathbb{F}_p^{u \times v}$

instance 1: public $x_1 \in \mathbb{F}_p^n$, witness $z_1 = (x_1, u_1) \in \mathbb{F}_p^v$

instance 2: public $x_2 \in \mathbb{F}_p^n$, witness $z_2 = (x_2, u_2) \in \mathbb{F}_p^v$

We know $Az_i \circ Bz_i = Dz_i$ for $i = 1, 2$

**First attempt**:

Lets define r as a random variable:

$r \leftarrow \mathbb{F}_p, x \leftarrow x_1 + rx_2$

$z \leftarrow z_1 + rz_2 = (x_1 + rx_2, w_1 + rw_2)$

Then:

$Az \circ Bz = A(z_1 + rz_2) \circ B(z_1 + rz_2)$

$= (Az_1) \circ (Bz1) + r^2(Az_2) \circ (Bz2) + (r(Az_2) \circ (Bz_1) + r(Az_1) \circ (Bz_2))$

$= Dz_1 + r^2 Dz_2 + E$

Where $E$ is a combination of the remaining values.

This is not quite an R1CS, because it is not of the form $Az_i \circ Bz_i = Dz_i$. We need to modify the R1CS so that it can be folded.

Let's **define a relaxed R1CS**:

$A, B, D \in \mathbb{F}_p^{u \times v}, (x_1 \mathbb{F}_p^n, c \in \mathbb{F}_p, E \in \mathbb{F}_p^u)$

Witness: $z = (x, w) \in \mathbb{F}_p^v s.t. (Az) \circ (Bz) = c(Dz) + E$

Lets **fix the R1CS** program once again:

$A, B, D \in \mathbb{F}_p^{u \times v}$

instance 1: public $(x_1, c_1, E_1)$,witness$z_1 = (x_1, w_1) \in \mathbb{F}_p^v$

instance 2: public $(x_2, c_2, E_2)$,witness$z_2 = (x_2, w_2) \in \mathbb{F}_p^v$

We know $(Az_i) \circ (Bz_i) = c_i(Dz_i) + E_i$ for $i = 1, 2$

**Second attempt**:

$T \leftarrow (Az_2) \circ (Bz_1) + (Az_1) \circ (Bz_2) - c_1(Dz_2) - c_2(Dz_1)$

$x \leftarrow x_1 + rx_2, c \leftarrow c_1 + rc_2, E \leftarrow E_1 + rT + r^2 E_2$

$$z \leftarrow z_1 + rz_2 = (x_1 + rx_2, w_1 + rw_2)$$

$$Az \circ Bz =$$

$$= A(z_1) \circ rB(z_1) + r^2(Az_2) \circ (Bz_2) + r(Az_2) \circ (Bz_1) + r(Az_1) \circ (Bz_2)$$

$$= c_1(Dz_1) + E_1 + r^2 c_2(Dz_2) + r^2 E_2 + r((Az_2) \circ (Bz_1) + (Az_1) \circ (Bz_2))$$

$$= (c_+ rc_2)(Dz_1 + rDz_2) + E_1 + r^2 E_2 + rT$$

$$= c(Dz) + E$$

We have a valid relaxed R1CS.[**?**] [**?**]

# Chapter 4

# Proof construction

We will be using Circom to generate an airthmetic circuit, and SnarkJS for the proof generation.

**Circom** is a domain-specific language for creating arithmetic circuits, which are used in zk-SNARKs. The circuit code can be written to specify the desired constraints. Circom allows expressing the circuit's arithmetic operations, constraints, input and output in a concise and readable manner. Once the circuit is designed, it needs to be compiled into a format suitable for zk-SNARKs.

**SnarkJS** is a JavaScript library that provides tools for working with zk-SNARKs, including circuit compilation. After compilation, SnarkJS facilitates generating zk-SNARK proofs for specific instances of the circuit. SnarkJS also provides utilities for verifying the proofs.

## 4.1 Proof of liabilities

The proof of liabilities operates on a list of balances and a list of email hashes as private inputs. The first purpose of the circuit is to validate that all values are non-negative and that all balances fall within a specified range. These verifications are crucial to prevent overflow or underflow issues, given that the operations occur within a finite field.

Subsequently, the proof of liabilities constructs a Merkle tree and provides outputs the total balance sum and the root hash of the Merkle tree.

**Inputs**

1. List of balance (private)
2. List of email hash (private)

**Outputs**

1. Balance Sum (public)
2. Root hash (public)
3. No negative values (private) - boolean
4. All small range (private) - boolean

This proof of liabilities operates as intended because it returns the sum of the liabilities, which is exact because of the checks. It also returns the root hash, insuring

you cannot alter any values insiside the merkle tree. The merkle tree is hidden so that we do not give any information about users and their balances. The root hash will be used to verify the inclusion of the balances.

In a complete proof of reserves, the balance sum would be a private output. We would have another circuit proving that the sum of liabilities is smaller than the sum of assets, without revelaing the balance.

## 4.2 Proof of inclusion

The proof of inclusion aims to prove that the balance of a user is included in the Merkle Tree created in the proof of liabilities. To prove that a balance is included, it is sufficient to show that you know the Merkle path of a user balance, which we define using the list of neighbors sum, hash and binary. The neighbors binary variable indicates whether the neighbor is on the left or the right. The root hash, root sum, user balance and user email hash are public because it needs to be shown which user is in which tree.

In the figure 4.1, Charlie is defined as the user. The merkle path is composed of the three blue nodes. In each blue nodes we can find the variables of the lists, namely the binary (left or right), the sum and the hash.

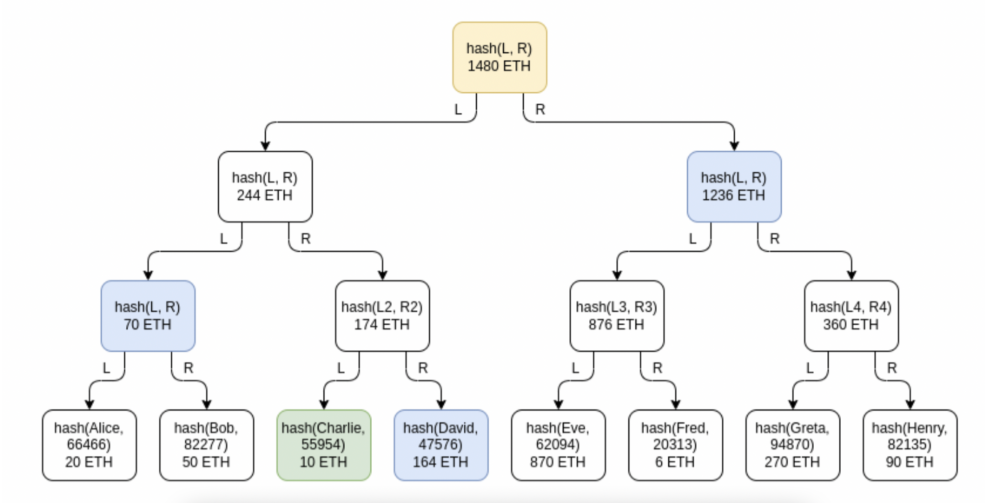


Figure 4.1: Merkle Path [**?**]

### Inputs

1. List of neighbors sum (private)
2. List of neighbors hash (private)
3. List of neighbors binary (private)
4. Root hash (public)
5. Root sum (public)
6. User balance (public)
7. User email hash (public)

### Outputs

1. Balance included (public) - boolean

In the circuit, we verify that the combination of the user balance,sum and mekle path gives the right root hash and root sum. There is no additionnal verifications since they are already done for this root hash, in the proof of liabilities.

## 4.3 Daily proof of liabilities

## 4.4 Daily proof of inclusion

# Bibliography


[1] S. Nakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System. Technical report, Bitcoin.org, 2008.

[2] Binance. Proof of Reserves. `https://www.binance.com/en/proof-of-reserves`.

[3] Binance. Proof of solvency. `https://gusty-radon-13b.notion.site/Proof-of-solvency-61414c3f7c1e46c5baec32b9491b2b3d`.

[4] Boaz Barak. Lecture 14: Zero knowledge proofs. `https://www.boazbarak.org/cs127spring16/chap14_zero_knowledge.html`.

[5] LCX Team. Introduction to Zero-Knowledge Proofs. November 3, 2023. `https://www.lcx.com/introduction-to-zero-knowledge-proofs/`.

[6] Goldwasser, S., Micali, S., Rackoff, C. The Knowledge Complexity of Interactive Proof Systems. *SIAM Journal on Computing*, 18(1), 186–208. 1989.

[7] Blum, M., Feldman, P., Micali, S. Non-Interactive Zero-Knowledge and Its Applications. *STOC '88: Proceedings of the twentieth annual ACM symposium on Theory of computing*, 103–112. 1988.

[8] Goldreich, O., Micali, S., Wigderson, A. Interactive and Noninteractive Zero Knowledge are Equivalent in the Help Model? *STOC '91: Proceedings of the twenty-third annual ACM symposium on Theory of computing*, 113–131. 1991.

[9] Dan Boneh, Shafi Goldwasser, Dawn Song, Justin Thaler, Yupeng Zhang. Zero Knowledge Proofs: Introduction to Modern SNARKs. 2023.

[10] Nitulescu, Anca. *zk-SNARKs: A Gentle Introduction*. 2020.

[11] Dagher, Gaby G., Bunz, Benedikt, Bonneau, Joseph, Clark, Jeremy, and Boneh, Dan. Provisions: Privacy-preserving proofs of solvency for Bitcoin exchanges. October 26, 2015.

[12] Malkhi, Dahlia. Exploring Proof of Solvency and Liability Verification Systems. Published January 5, 2023. Updated November 27, 2023. `https://blog.chain.link/proof-of-solvency/`.

[13] Mazars. Proof of Reserve Report. December 9, 2022. https://www.crypto.com/proof-of-reserve.

[14] Kraken. Proof of Reserves. https://www.kraken.com/proof-of-reserves.

[15] Jim.ZK, zkLink Labs. Nova Studies I: Exploring Aggregation, Recursion, and Folding. Published in zkLinkBlog, Nov 6, 2023. https://blog.zk.link/nova-studies-i-exploring-aggregation-recursion-and-folding-23b9a67000cd.

[16] Trace. The Proof Supply Chain. Figment Capital, Jan.16, 2024. https://figmentcapital.medium.com/.