PEDECIBA

UNIVERSIDAD DE LA REPUBLICA URUGUAY

Facultad de Ingeniería
Universidad de la República
Montevideo, Uruguay

MASTER THESIS

Thesis submitted in fulfillment of the requirements for the degree of Magister en Informática by

Adrián Gerardo SILVEIRA LAPENNE

June 2022

A Formal Analysis of the Mimblewimble Cryptocurrency Protocol with a Security Approach

Directors: Dr. Carlos Luna and Dr. Gustavo Betarte

Abstract

A cryptocurrency is a digital currency that can be exchanged online for goods and services. Cryptocurrencies are deployed over public blockchains which have the transactions duplicated and distributed across the nodes of a computer network. This decentralized mechanism is devised in order to achieve reliability in a network consisting of unreliable nodes. Privacy, anonymity and security have become crucial in this context. For that reason, formal and mathematical approaches are gaining popularity in order to guarantee the correctness of the cryptocurrency implementations. Mimblewimble is a privacy-oriented cryptocurrency technology which provides security and scalability properties that distinguish it from other protocols of its kind. It was proposed by an anonymous developer, who posted a link to a text file on the IRC channel by the name Tom Elvis Jedusor (french name for Voldemort) in mid-2016. Mimblewimble's cryptographic approach is based on Elliptic Curve Cryptography which allows to verify a transaction without revealing any information about the transactional amount or the parties involved. Mimblewimble combines Confidential transactions, CoinJoin and cut-through to achieve a higher level of privacy and security, as well as, scalability. In this thesis, we present and discuss these security properties and outline the basis of a model-driven verification approach to address the certification of the correctness of the protocol implementations. In particular, we propose an idealized model that is key in the described verification process. The main components of our idealized model are transactions, blocks and chain. Then, we identify and precisely state the conditions for our model to ensure the verification of relevant security properties of Mimblewimble. In addition, we analyze the `Grin` and `Beam` implementations of Mimblewimble in their current state of development. We present detailed connections between our model and their implementations regarding the Mimblewimble structure and its security properties.

Keywords: cryptocurrency, mimblewimble, idealized model, formal verification, security.

List of publications

The following publications have been generated in the context of the thesis:

1. Gustavo Betarte, Maximiliano Cristiá, Carlos Daniel Luna, Adrián Silveira, and Dante Zanarini. Towards a formally verified implementation of the Mimblewimble cryptocurrency protocol. Applied Cryptography and Network Security Workshops - ACNS 2020 Satellite Workshops, AIBlock. Rome, Italy, October 19-22, 2020, Proceedings, volume 12418 of Lecture Notes in Computer Science, pages 3–23. Springer, 2020. [BCL$^+$20]
2. Adrián Silveira, Gustavo Betarte, Maximiliano Cristiá, and Carlos Luna. A formal analysis of the Mimblewimble cryptocurrency protocol. Special Issue "Security, Trust and Privacy in New Computing Environments", Sensors, 21(17):5951, 2021. [SBCL21a]
3. Adrián Silveira, Gustavo Betarte, Maximiliano Cristiá, and Carlos Luna. A range proof scheme analysis for the Mimblewimble cryptocurrency protocol. IEEE UruCon 2021. [SBCL21b]

Contents

1
Introduction

A cryptocurrency is a digital currency that can be exchanged online for goods and services. Through a cryptocurrency exchange it can be converted into cash and vice versa. Many cryptocurrencies work using a technology called blockchain which is a distributed ledger of transactions that is duplicated and distributed across the nodes of a computer network. A defining feature of cryptocurrencies is that there is no central trusted authority. The ledger is maintained by using a consensus-based validation protocol where transactions are constructed in a peer-to-peer fashion and broadcasted to the entire set of participants who work to validate them and construct blocks. Therefore, the consensus algorithm is what decides which is the next block to be appended to the blockchain. This decentralized mechanism is devised in order to achieve reliability in a network consisting of unreliable nodes. Next, we present relevant security aspects of the cryptocurrencies and the importance of applying formal methods for the verification of their implementations. Then, we state the aims and the structure of this thesis.

1.1 Cryptocurrency Security

Cryptocurrency protocols deal with virtual money. Thus, they are a valuable target for highly skilled attackers. Several attacks have already been mounted against cryptocurrency systems, causing irreparable losses of money and credibility (e.g. [But]). Furthermore, it is necessary to understand virtual money in the context of a global financial system which has been deteriorated due to the COVID-19 pandemic [MV].

In this context, security and confidential properties have become even more crucial. For this reason the cryptocurrency community is seeking approaches, methods, techniques and development practices that can reduce the chances of successful attacks. One such approach is the application of formal methods to software implementation. In particular, interest in formal proofs and formally certified implementations has increased [Ros20, GKGG21]. Formal and mathematical approaches are also gaining popularity in the field of service industries, where they are used to analyze and propose solutions to problems. For instance, An et al. [AMJ21] modeled a network revenue management problem and proposed a genetic algorithm to maximize revenue providing computational experiments. Results demonstrated the effectiveness of this approach.

Security (idealized) models have played an important role in the design and evaluation of high assurance security systems. Their importance was already pointed out in the Anderson report [And]. The paradigmatic Bell-LaPadula model [LBL], conceived in 1973, constituted the first big effort to provide a formal setting to study and reason on confidentiality properties of data in time-sharing mainframe systems. In that work, the authors provide a general descriptive model of a computer system involving formally precised security concepts like simple-security, discretionary-security and the star-property over State machines. *State machines* can be employed as the building block of a security model. The basic features of a state machine model are the concepts of state and state change. A *state* is a representation of the system under study at a given time, which should capture those aspects of the system that are relevant to the analyzed problem. State changes are modeled by a state transition function that defines the next state based on the current state and input. If one wants to analyze a specific safety property of a system using a state machine model, one must first specify what it means for a state to satisfy the property, and then check if all state transitions preserve it. Thus, state machines can be used to model the enforcement of a security policy.

Mimblewimble (MW) is a privacy-oriented cryptocurrency technology encompassing security and scalability properties that distinguish it from other technologies of its kind. MW was first proposed in 2016 [Jed]. The idea was then further developed by Poelstra [Poe]. In MW, unlike Bitcoin [Nak], there is no such concept as an address, and all the transactions are confidential. The approach of this thesis is based on formal software verification aimed at formally verifying the basic mechanisms of MW and its implementations [Grid, Fou]. Security issues that pertain to the realm of critical mechanisms of the MW protocol are explored on an idealized model of this system. Such model abstracts away the specifics of any particular implementation, and yet provides a realistic setting. Verification should be then performed on more concrete models, where low level mechanisms are specified. Finally the low level model should be proved to be a correct implementation of the idealized model.

1.2 Related Work

Developers of cryptocurrency software have already shown interest in using mathematics as a tool to describe software. In fact, both Nakamoto, in his seminal paper on Bitcoin [Nak], and Wood, in his description of the Ethereum Virtual Machine (EVM) [Woo], make use of mathematical constructs. In particular, in the latter work it is explained the Ethereum blockchain as a transaction-based state machine and the programs to be executed on the EVM are called smart contracts. However, those descriptions can not be understood as Formal Methods (FM) because they are neither based on standardized notations nor on clear mathematical theories.

Sestrem et. al [SOASdM$^+$20] have pointed out the importance of privacy and security over Smart grid systems. They proposed a sidechain architecture which is built up of three different blockchains. They claimed that their solution ensures system privacy, security, and reliability. In particular, one of those blockchains is responsible for ensuring such properties in the system, which are verified performing several tests using the Loom Platform. In this kind of scenarios, a formal definition of those security properties would help to understand what is required to be proved, independently of any particular implementation.

In addition, the FM community has started to pay attention to cryptocurrency software. Idelberger et al. [IGRS16] proposed to use defeasible logic frameworks such as Formal Contract Logic for the description of smart contracts. However, in that work the authors did not analyze cryptocurrency protocols nor the necessary conditions to guarantee security properties that those protocols should satisfy.

Bhargavan et al. [BDLF$^+$16] compiled SOLIDITY programs into a verification-oriented functional language where they can verify the source code. Although the paper describes a framework to analyze and verify both the runtime security and the functional correctness of blockchain systems, the work only focuses on smart contracts. Luu et al. [LCO$^+$16] used the OYENTE tool to find and detect vulnerabilities in smart contracts. Hirai [Hir17] used Lem to formally specify the EVM; Grishchenko, Maffei and Schneidewind [GMS18] also formalized the EVM but in F*; and Hildenbrandt et al. did the same but with the reachability logic system known as $\mathbb{K}$. Pîrlea and Sergey [PS18] presented a Coq [The, BC04] formalization of a blockchain consensus protocol where some properties are formally verified.

More recently, Rosu [Ros20] presented academic and commercial results about the development of blockchain languages and virtual machines that come directly equipped with formal analysis and verification tools. Hajdu et al. [HJC20] developed a source-level approach for the formal specification and verification of Solidity contracts with the primary focus on events. Santos Reis et al. [RCdS20] introduced Tezla, an intermediate representation of Michelson smart contracts that eases the design of static smart contract analyzers. In [BGW20], Boyd et al. presented a blockchain model in Tamarin, that is useful for analyzing certain blockchain based protocols. On the other hand, Garfatta et al. [GKGG21] described a general overview of the different axes investigated actually by researchers towards the (formal) verification of Solidity smart contracts. Tolmach et al. [TLL$^+$21] investigated formal models and specifications of smart contracts and presented a systematic overview to understand common trends, although they did not specifically consider security in cryptocurrency protocols.

Additionally, Metere and Dong [MD17] presented a mechanized formal verification of the Pedersen commitment protocol using EASYCRYPT [BDG$^+$13] and Fuchsbaue et al. [FOS19] introduced an abstraction for the analysis of some security properties of MW. Our work assumes some of these results to formalize and analyze the MW protocol, to then propose a methodology to verify their implementations.

1.3 Aims and structure of the thesis

The goal of the present work was to identify and analyze the main components of the MW protocol in order to build an idealized model and verify security properties. The specific objectives were to i) identify and specify the based schemes and protocols of MW, ii) identify and define the main components of our model, iii) define and verify relevant security properties and iv) compare our model with the most popular implementations of MW: Grin and Beam.

This thesis builds upon and extends previously published papers [BCL$^+$20, SBCL21a, SBCL21b]. In those papers, we have presented elements that comprise the essential steps towards the development of an exhaustive formalization of the MW cryptocurrency protocol and the analysis of some of its properties. In particular, we have defined the main components of our idealized model and provided validity conditions to guarantee the correctness of the blockchain. In addition, we have identified and precisely stated the conditions for our model to ensure the verification of relevant security properties of MW. We have studied the protocol and security properties a MW blockchain should have as well as security properties of the Pedersen commitment and range proofs schemes. The proposed idealized model constitutes the main contribution together with the analysis of the essential properties it is shown to verify. We have also introduced and discussed the basis of a model-driven verification approach to address the certification of the correctness of a protocol's implementation. Finally, we have compared two MW implementations, `Grin` [Grid] and `Beam` [Fou], with our model and we have discussed some features that set them apart.

The work presented here closely follows the approach of [BCL$^+$21] where we have outlined some formal methods related techniques that we consider particularly useful for cryptocurrency software.

My main contributions to this work were, firstly, to conceptualize and model the main components of the MW protocol. Secondly, to define the schemes the model is based on. Thirdly, to analyze some relevant security properties a cryptocurrency protocol should have. Then, to define and prove these properties on the model. Finally, to analyze and compare the main two MW implementations with our model.

The thesis is organized as follows. Chapter 2 provides a brief description of MW. Chapter 3 defines the schemes and protocols our model is based on. Chapter 4 describes the building blocks of a formal idealized model (abstract state machine) of the computational behavior of MW. Chapter 5 states the verification activities we are putting in place in order to verify the protocol and its implementation. Then, Chapter 6 analyzes the `Grin` and `Beam` implementations of MW in their current state of development. Final remarks and directions for future work are presented in Chapter 7.

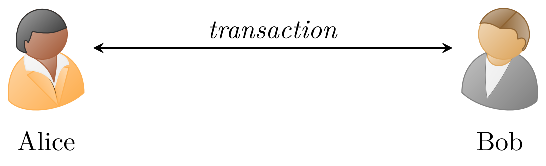
Table 8.1 in Appendix 8.1 shows the list of abbreviations and acronyms. Table 8.2 provides the meaning of the math symbols used throughout this work.

2
The Mimblewimble Protocol

In August 2016, someone called "Tom Elvis Jedusor" (french name for Voldemort in Harry Potter) posted a link to a text file on the IRC Channel describing a cryptocurrency protocol with a different approach from BitCoin. This article titled 'Mimblewimble' [Jed] addressed some privacy concerns and the ability of compressing the transaction history of the chain without losing validity verification. Since this document left some questions open, in October 2016 Andrew Poelstra published a paper [Poe] where he described, in more detail, the design of a blockchain based on MW. Next, we describe how money transfer is carried out on the MW protocol and the schemes and protocols our model is based on.

2.1 Transactions

Suppose Alice and Bob agree on a money transfer. Alice wants to send v coins to Bob. They communicate off-chain and create the MW transaction which includes the transaction amount v.



In MW, transactions are Confidential transactions [Maxb, Gib]. A transaction allows a sender (Alice) to encrypt the amount v of bitcoins by using blinding factors. In a confidential transaction only the two parties involved know the amount of bitcoins being exchanged. However, for anyone observing the transaction it is possible to verify its validity by comparing the number of inputs and outputs; if both are the same, then the transaction will be considered valid. Such procedure ensures that no money have been created from nothing and is key in preserving the integrity of the system. In MW transactions, the recipient (Bob) randomly selects a range of blinding factors provided by the sender, which are then used as proof of ownership by the receiver.

The MW protocol aims at providing the following properties [Jed, Grid]:

- Verification of zero sums without revealing the actual amounts involved in a transaction, which implies confidentiality.
- Authentication of transaction outputs without signing the transaction.
- Good scalability, while preserving security, by generating smaller blocks—or better, reducing the size of old blocks, producing a blockchain whose size does not grow in time as much as, for instance, Bitcoin's.

The first two properties are achieved by relying on Elliptic Curves Cryptography (ECC) operations and properties. The third one is a consequence of the first two.

2.2 Verification of transactions

If v is the value of a transaction (either input or output) and H is a point over an elliptic curve, then $v.H$ encrypts v because it is assumed to be computationally hard to get v from $v.H$ if we only know H. However, if w and z are other values such that $v+w=z$, then if we only have the result of encrypting each of them with H we are still able to verify that equation. Indeed:

$$v + w = z \Leftrightarrow v.H + w.H = z.H$$

due to simple properties of scalar multiplication over groups. Therefore, with this simple operations, we can check sums of transactions amounts without knowing the actual amounts.

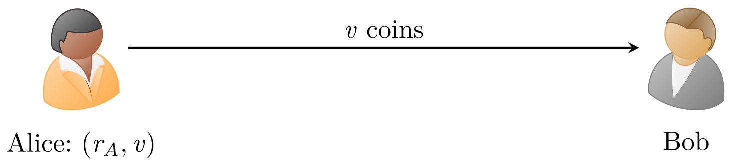
Nevertheless, if we have previously encrypted v with H and now we see $v.H$, we know that it is the result of encrypting v. In the context of blockchain transactions this is a problem because once a block holding $v.H$ is saved in the chain it will reveal all the transactions of v coins. For such problems, MW encrypts v as $r.G+v.H$ where r is a scalar and G is another point in H's elliptic curve, r is called *blinding factor* and $r.G + v.H$ is called *Pedersen commitment*. By using Pedersen commitments, MW allows the verification of expressions such as $v + w = z$ providing more privacy than the standard scheme. In effect, if $v + w = z$ then we choose r_v, r_w and r_z such that $r_v.G + r_w.G = r_z.G$ and so the expression is recorded as:

$$\overbrace{(r_v G + v.H)}^{v} + \overbrace{(r_w.G + w.H)}^{w} = \overbrace{r_z.G + z.H}^{z}$$

making it possible for everyone to verify the transaction without knowing the true values.

2.3 Authentication of transactions

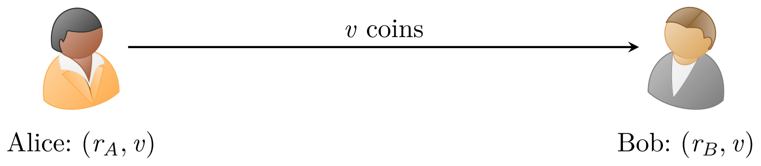
Consider that Alice has received v coins and this was recorded in the blockchain as $r_A.G + v.H$, where r_A was chosen by her to keep it private. Now she wants to transfer these v coins to Bob.



As a consequence, Alice looses v coins and Bob receives the very same amount, which means that the transaction adds to zero: $r_A.G+v.H-(r_A.G+v.H) = 0.G-0.H$. However, Alice now knows Bob's blinding factor because it must be the same chosen by her (so the transaction is balanced). In order to protect Bob from being stolen by Alice, MW allows Bob to add his blinding factor, r_B, in such a way that the transaction is recorded as:

$$(r_A + r_B).G + v.H - (r_A.G + v.H) = r_B.G - 0.H$$

although now it does not sum zero.



However, this *excess value* is used as part of an authentication scheme. Indeed, Bob uses r_B as a private key to sign the empty string (ϵ). This signed document is attached to the transaction so in the blockchain we have:

- Input: I.
- Output: O.
- Bob's signed document: S.

In this way, the transaction is valid if the result of decrypting S with $I - O$ (in the group generated by G) yields ϵ. If $I - O$ does not yield something in the form of $r_B.G - 0.H$, then ϵ will not be recovered and so we know there is an attempt to create money from thin air or there is an attempt to steal Bob's money.

3 Schemes and Protocols

A commitment scheme is a cryptographic primitive that allows a player in a protocol to choose a value and commit to his choice such that he can no longer change his mind. The value is kept hidden to others with the ability to reveal the committed value later. In MW transactions, the transaction amounts and blinding factors are hidden in Pedersen commitments. It means that for someone observing the transaction, it is hard (in terms of complexity) to know any of them. In addition, since the transaction amounts are hidden, it should be possible to verify that the values are positive without revealing any information about them. Range proofs should be provided to guarantee the transactional amount lies in some range. Moreover, a transaction contains a signature to guarantee it was honestly constructed. Next, we define the schemes and protocols our model is based on.

3.1 Commitment Scheme

A commitment scheme [Cré11] is a two-phase cryptographic protocol between two parties: a sender and a receiver. At the end of the commit phase the sender is committed to a specific value that he cannot change later and the receiver should have no information about the committed value.

A non-interactive commitment scheme [BBB$^+$18] can be defined as follows:

Definition 3.1 (Non-interactive Commitment Scheme). *A non-interactive commitment scheme $\zeta(Setup, Com)$ consists of two probabilistic polynomial time algorithms, Setup and Com, such that:*

- *Setup generates public parameters for the scheme depending on the security parameter λ.*
- *Com is the commitment algorithm: $Com : M \times R \to C$, where M is the message space, R the randomness space and C the commitment space. For a message $m \in M$, the algorithm draws uniformly at random $r \leftarrow R$ and computes the commitment $com \leftarrow Com(m, r)$.*

We have simplified the notation, but it is essential to remember that *Com*, M, R and C depend on the parameters generated by *Setup*.

The commitment scheme is homomorphic if:

$$\forall\, m_1, m_2 \in M, r_1, r_2 \in R:$$

$$Com(x_1, r_1) + Com(x_2, r_2) = Com(x_1 + x_2, r_1 + r_2)$$

In other words, *Com* is additive in both parameters.

Transactions in MW are derived from Confidential transactions [Maxb], which are enabled by Pedersen commitments with homomorphic properties over elliptic curves. We define the non-interactive Pedersen commitment scheme we will use in our model, based on Definition 3.1, as follows:

Definition 3.2 (Pedersen Commitment Scheme with Elliptic Curves). *Let M and R be the finite field $\mathbb{F}_n$ and let $\mathcal{C}$ be an elliptic curve of prime order n.*

As in Definition 3.1, the probabilistic polynomial time algorithms are defined as:

- *Setup generates the order n (dependent on the security parameter λ) and two generator points G and H on the elliptic curve $\mathcal{C}$ of prime order n whose discrete logarithms relative to each other are unknown.*
- *$Com(v, r) = r.G + v.H$, with v the transactional value and r the blinding factor choosen randomly in $\mathbb{F}_n$.*

Security properties of this commitment scheme (for MW) will be analyzed in Chapter 5.2.1.

Each MW transaction contains a list of range proofs of the transactional values. Next, we define the range proof scheme we will analyze in our model.

3.2 Range Proof Scheme

Range proofs aim at proving that a secret value is in a certain range without revealing the value. Transactions in MW contain a list of range proofs proving that the transactional values are positive and less than a certain upper bound to avoid overflow errors. We define a non-interactive zero-knowledge (NIZK) range proof scheme from a commitment scheme as follows:

Definition 3.3 (NIZK Range Proof Scheme). *Let $\zeta(Setup, Com)$ be a Pedersen commitment scheme as in Definition 3.2. Let P be the range proof space for the values $v \in M$ such that v lies in some range $[a, b]$. A non-interactive zero-knowledge range proof scheme $\eta(Prove, Verify)$ consists of two probabilistic polynomial time algorithms, Prove and Verify, such that:*

- *$Prove : M \times R \times C \to P$, which receives a value v, the random value r and the commitment $c = Com(v, r)$ and computes the range proof for the value v; and*
- *$Verify : C \times P \to bool$, that given a commitment value and a range proof, decides if the value is in the range.*

Notice that, the *Prove* algorithm computes a zero-knowledge proof to the commitment in order to verify that the committed value is in a certain range. In other words, the proof enables a prover to convince a verifier that the statement holds without revealing any information about the secret value.

3.3 Schnorr Signature Protocol

The construction of the MW transaction is made off-chain by the parties. For simplicity, we will work with a signature protocol between two parties but this can be generalized to multi-parties.

During the transaction construction, as we will see in Chapter 4.2, Alice needs to verify Bob's Schnorr signature. Schnorr signature protocols can be applied over any group where discrete logarithm is hard, in our case, over an elliptic curve $\mathcal{C}$.

Next, we define the Schnorr signature protocol used by them during the transaction construction. The message m can be the empty string.

Definition 3.4 (Schnorr Signature Protocol). *Let $\mathcal{C}$ be an elliptic curve of prime order n with generator G. Let hash : $\{0,1\}^* \to \mathbb{F}_n$ be a cryptographic hash function over the finite field $\mathbb{F}_n$. Alice secretly knows $k_A \in \mathbb{F}_n$ whose public key is $K_A = k_A.G$*

Signing

The following steps are followed to create a signature on a message $m \in \{0,1\}^*$:

1. *Alice chooses nonce $n_A \xleftarrow{\$} \mathbb{F}_n$*
2. *She computes public key $N_A = n_A.G$*
3. *She computes $e = hash(N_A \mid K_A \mid m)$ and $s_A = n_A + e.k_A$ where $\mid$ denotes concatenation and N_A, K_A are represented as a bit string.*
4. *The signature σ is defined as follows:*

$$\sigma = (s_A, N_A)$$

with public key K_A

Validating

A signature $\sigma = (s_A, N_A)$ is valid if the following holds:

$$s_A.G = N_A + e.K_A$$

Each MW transaction contains a signature σ made by the parties during the transaction construction which can be seen as a Schnorr multi-signature.

Next, a Schnorr signature protocol aggregation is defined according to our model.

Definition 3.5 (Schnorr Signature Protocol Aggregation). *Let $\mathcal{C}$ be an elliptic curve of prime order n with generator G. Let hash : $\{0,1\}^* \to \mathbb{F}_n$ be a cryptographic hash function over the finite field $\mathbb{F}_n$. Alice and Bob secretly know $k_A, k_B \in \mathbb{F}_n$ whose public keys are $K_A = k_A.G$ and $K_B = k_B.G$ respectively.*

Signing

The following steps are followed to create a multisignature on a message $m \in \{0,1\}^*$:

1. *Alice and Bob choose nonces $n_A, n_B \xleftarrow{\$} \mathbb{F}_n$ respectively*
2. *They compute public keys $N_A = n_A.G$ and $N_B = n_B.G$*
3. *They compute $e = hash(N_A + N_B \mid K_A + K_B \mid m)$ and respectively compute:*

$$s_A = n_A + e.k_A$$

$$s_B = n_B + e.k_B$$

4. *The agregate singature σ is defined as follows:*

$$\sigma = (s_a + s_B, N_A + N_B)$$

with the aggregate public key $K_A + K_B$

Validating

A signature $\sigma = (s_a + s_B, N_A + N_B)$ is valid if the following holds:

$$(s_A + s_B).G = N_A + N_B + e.(K_A + K_B) \tag{3.1}$$

Next, we show that a signature σ honestly constructed will be valid.

If we consider the signing process, we know that: $s_A = n_A + e.k_A$ and $s_B = n_B + e.k_B$

By applying algebraic properties on elliptic curves, the left term on the equality 3.1 can be written as:

$$(s_A + s_B).G = (n_A + e.k_A).G + (n_B + e.k_B).G =$$

$$n_A.G + e.k_A.G + n_B.G + e.k_B.G = n_A.G + n_B.G + e.(k_A.G + k_B.G)$$

So, if we substitute the left term on the equality 3.1, we have:

$$n_A.G + n_B.G + e.(k_A.G + k_B.G) = N_A + N_B + e.(K_A + K_B)$$

The above equality holds because:

$$N_A = n_A.G,\ K_A = k_A.G \text{ and } N_B = n_B.G,\ K_B = k_B.G$$

(N_A, K_A) are Alice's public keys and (N_B, K_B) are Bob's public keys. Since we are working over the elliptic curve $\mathcal{C}$ where the discrete logarithm is hard, the only ones who know the private keys (n_A, k_A) and (n_B, k_B) are Alice and Bob respectively.

4 Idealized Model

The basic elements of our model are transactions, blocks and chains. Each node in the blockchain maintains a local state. The main components are the local copy of the chain and the set of transactions waiting to be validated and added to a new block. Moreover, each node keeps track of unspent transaction outputs (UTXOs). Properties such as zero-sum and the absence of double spending in blocks and chains must be proved for local states. The blockchain global state can be represented as a mapping from nodes to local states. Next, we define all the elements which compose our idealized model.

4.1 Transactions

Given two fixed generator points G and H on the elliptic curve $\mathcal{C}$ of prime order n (whose discrete logarithms relative to each other are unknown), we define a single transaction between two parties as follows:

Definition 4.1 (Transaction). *A single transaction t is a tuple of type:*

$$\mathit{Transaction} \overset{\text{def}}{=} \{i : I^*,\ o : O^*, tk : \mathit{TxKernel}, tko : \mathit{KOffset}\}$$

with X^* representing the lists of elements of type X and where:

- *$i = [c_1, ..., c_n]$ and $o = [o_1, ..., o_m]$ are the lists of inputs and outputs. Each input c_i and output o_i are points over the curve $\mathcal{C}$ and they are the result of computing the Pedersen commitment $r.G + v.H$ with r the blinding factor and v the transactional value in the finite field $\mathbb{F}_n$.*
- *$tk = \{rp, ke, \sigma\}$ is the transaction kernel where:*
  - *$rp = [rp_1, ..., rp_m]$ is a list of range proofs of the outputs. The $j-th$ item rp_j in rp corresponds to the $j-th$ item o_j in o*
  - *ke is the transaction excess represented by $(\sum_1^m r' - \sum_1^n r - tko).G$*

  - *σ is the kernel Schnorr signature (for simplicity, fees are left aside)*
- *$tko \in \mathbb{F}_n$ is the transaction kernel offset.*

The transaction kernel offset will be used in the construction of a block to satisfy security properties.

The ownership of a coin is given by the following definition:

Definition 4.2 (Ownership). *Given a transaction t, we say S owns the output o if S knows the opening (r, v) for the Pedersen commitment $o = r.G + v.H$.*

The strength of this security definition is directly related to the difficulty of solving the logarithm problem. If the elliptic curve discrete logarithm problem in $\mathcal{C}$ is hard then given a multiple Q of G, it is computationally infeasible to find an integer r such that $Q = r.G$.

It is important to notice that during the construction of the transaction the sender and the receiver do not learn their respective blinding factors. Instead, they build a Schnorr signature that is used to guarantee the authenticity of the transaction excess value.

We say that a transaction is valid if the following property holds:

Property 4.3 (Valid Transaction). A transaction t is valid ($valid\_transaction(t)$) if t satisfies:

i. The range proofs of all the outputs are valid.
ii. The transaction is balanced.
iii. The kernel signature σ is valid for the excess.

These three properties have a straightforward formalization in our model.

The first property we should guarantee is that all the range proofs of all the outputs are valid.

Definition 4.4 (Valid Range Proof Outputs Transaction). *Let $t = \{i, o, tk, tko\}$ be a transaction as in Definition 4.1 with transaction kernel $tk = \{rp, ke, \sigma\}$ where $o = [o_1, ..., o_m]$ is the list of outputs and $rp = [rp_1, ..., rp_m]$ is the list of the range proof outputs. Let $\eta(Prove, Verify)$ be a NIZK scheme as in Definition 3.3 with P the range proof space where $rp_j \in P$ proves that o_j lies in the range $[0, 2^n]$ where n is small enough to not cause overflow errors. We say all the range proof output transactions are valid if: for all $rp_j \in rp$, $Verify(o_j, rp_j) = true$.*

The list of range proof outputs provide a proof that each transactional output is positive, without revealing further information.

The second property is defined as follows:

Definition 4.5 (Balanced Transaction). *A transaction $t = \{i, o, tk, tko\}$, with transaction kernel $tk = \{rp, ke, \sigma\}$, is balanced if the following holds:*

$$\sum_{o_j \in o} o_j - \sum_{c_j \in i} c_j = ke + tko.G$$

A balanced transaction guarantees no money is created from thin air.

The kernel signature σ is a Schnorr signature aggregation with the kernel excess ke as the public key.

Note that, for simplicity during the transaction construction, in Definition 3.5 we consider a Schnorr signature aggregation between two parties, however, once the transaction is constructed it is not necessary to know the parties involved.

The third property is defined as follows:

Definition 4.6 (Valid Signature for the kernel excess). *Let $t = \{i, o, tk, tko\}$ be a transaction as in Definition 4.1 with transaction kernel $tk = \{rp, ke, \sigma\}$ where:*

- *rp is a list of range proofs of the outputs.*
- *ke is the transaction excess.*
- *$\sigma = (s, N)$ is the kernel Schnorr signature aggregation as in Definition 3.5 on the empty string m.*

We say the kernel signature σ is valid with public key ke if the following holds: $s.G = N + e.ke$ such that $e = hash(N \mid ke)$

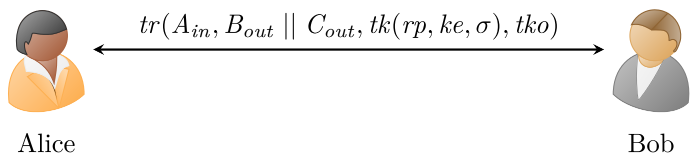
To illustrate the above definition, we detail the transaction construction between two parties.

4.2 Transaction construction

Suppose *Alice* wants to send v_B coins to *Bob*. They need to construct a transaction tr, as in Definition 4.1, which contains:

- Alice's Input A_{in}, such that she knows the opening (r_A, v_A) with r_A the blinding factor.
- Bob's Output B_{out} such that he knows the opening (r_B, v_B) with r_B the blinding factor and Alice's change C_{out} such that she knows the opening (r_C, v_C) with r_C the blinding factor. Let v_C be $v_A - v_B$.

The following image illustrates the target transaction tr.


$tr(A_{in}, B_{out} \mid\mid C_{out}, tk(rp, ke, \sigma), tko)$
Alice
Bob

The symbol || is the list concatenation operator.

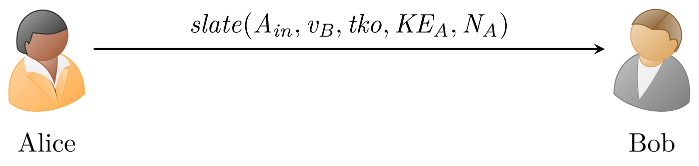
To construct the transaction, Alice and Bob will exchange the information using a data structure called *slate*.

Step 1

- Alice adds A_{in} and the amount v_B to the slate.
- Alice chooses $r_C \xleftarrow{\$} \mathbb{F}_n$ (blinding factor) and computes $C_{out} = r_C.G + v_C.H$ (Definition 3.2). Additionally, she computes the output range proof $rp_C = Prove(v_C, r_C, C_{out})$ (Definition 3.3) which will be added to the transaction in the step 3.
- Alice chooses the kernel offset $tko \xleftarrow{\$} \mathbb{F}_n$ and computes Alice's kernel excess secret key as:

$$ke_A = r_C - r_A - tko \tag{4.1}$$

- Alice adds to the slate: *tko* and Alice's kernel excess as $KE_A = ke_A.G$
- Alice chooses nonce $n_A \xleftarrow{\$} \mathbb{F}_n$ and adds the public key $N_A = n_A.G$ to the slate.
- Alice sends the slate to Bob.

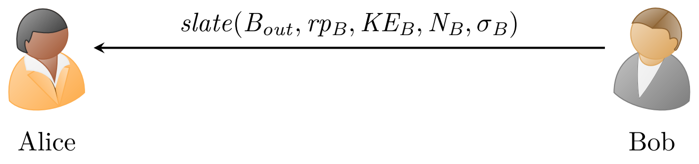


Step 2

- Bob chooses $r_B \xleftarrow{\$} \mathbb{F}_n$ (blinding factor) and computes $B_{out} = r_B.G + v_B.H$ Additionally, he computes the output range proof $rp_B = Prove(v_B, r_B, B_{out})$. He adds B_{out} to the slate.
- Bob computes Bob's kernel excess as:

$$KE_B = r_B.G \tag{4.2}$$

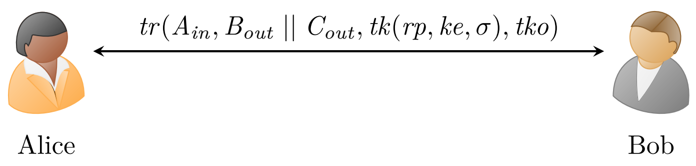
  and adds it to the slate.
- Bob chooses nonce $n_B \xleftarrow{\$} \mathbb{F}_n$ and adds the public key $N_B = n_B.G$ to the slate.
- Bob calculates the receiver Schnorr signature on the empty string as $\sigma_B = (s_B, N_B)$ where:
  $s_B = n_B + e.r_B$ such that $e = hash(N_A + N_B \mid KE_A + KE_B)$
- Bob adds $\sigma_B = (s_B, N_B)$ to the slate.
- Bob sends the slate to Alice.



Step 3

- Alice verifies Bob's signature $\sigma_B = (s_B, N_B)$ as in Definition 3.4:
  $s_B.G = N_B + e.KE_B$ such that $e = hash(N_A + N_B \mid KE_A + KE_B)$
- Alice computes the sender Schnorr signature on the empty string as $\sigma_A = (s_A, N_A)$ where:
  $s_A = n_A + e.ke_A$ such that $e = hash(N_A + N_B \mid KE_A + KE_B)$
- Alice sets the kernel excess $ke := KE_A + KE_B$.
- Alice sets the kernel signature $\sigma := (s_A + s_B, N_A + N_B)$.

Finally, Alice and Bob have computed all the remaining fields of the transaction:


$tr(A_{in}, B_{out} \mid\mid C_{out}, tk(rp, ke, \sigma), tko)$
Alice
Bob

where

- $rp := [rp_A, rp_B]$
- $ke := KE_A + KE_B$
- $\sigma := (s_A + s_B, N_A + N_B)$

It is important to notice that the kernel excess ke is the same as in Definition 4.1 represented by $(\sum_1^m r' - \sum_1^n r - tko).G$ as shown by the following equations:

- by Equation 4.1, $KE_A = ke_A.G = (r_C - r_A - tko).G$
- by Equation 4.2, $KE_B = r_B.G$

Then, $ke = KE_A + KE_B = (r_C - r_A - tko).G + r_B.G = (r_C + r_B - r_A - tko).G$. which is the sum of the output blinding factors minus the input blinding factor minus the transaction kernel offset.

In addition, we can highlight that during the transaction construction:

- Alice does not learn Bob's blinding factor r_B.
- Bob does not learn Alice's blinding factor r_C.
- Bob does not learn Alice's change amount $v_A - v_B$.

4.3 Aggregate Transactions

A single transaction can be seen as the sending of money between two parties. An aggregate transaction represents transactions between multiple parties:

Definition 4.7 (Aggregate Transaction). *An aggregate transaction tx is a tuple of type:*

$$TransacAgg \overset{\text{def}}{=} \{i : I^*,\ \ o : O^*, tks : TxKernel^*, tko : KOffset\}$$

A single transaction (Definition 4.1) can be seen as a particular case where the transaction kernel list contains a single element. Besides, the ownership of the coins is between two parties.

We say that an aggregate transaction is valid if the following property holds:

Property 4.8 (Valid Aggregate Transaction). Let $tx = \{i, o, tks, ko\}$ be an aggregate transaction with $tks = [tk_1, ..., tk_t]$ the list of transaction kernels where the j-th item in tks is of the form $tk_j = \{rp_j, ke_j, \sigma_j\}$. Then tx is valid if the following are satisfied:

i. all the range proofs rp_j are valid.
ii. the transaction is balanced.
iii. all the kernel signatures σ_j are valid for the excess ke_j.

A balanced aggregate transaction is defined as follows:

Definition 4.9 (Balanced Aggregate Transaction). *Let $tx = \{i, o, tks, ko\}$ be an aggregate transaction with $tks = [tk_1, ..., tk_t]$ the list of transaction kernels and where the j-th item in tks is of the form $tk_j = \{rp_j, ke_j, \sigma_j\}$. We say tx is balanced if the following holds:*

$$\sum_{o_j \in o} o_j - \sum_{c_j \in i} c_j = ko.G + \sum_{ke_j \in tks} ke_j$$

Transactions can be merged non-interactively to construct an aggregate transaction. This process can be applied recursively to add more transactions into one aggregate transaction. The CoinJoin mechanism [Maxa] makes it possible. It combines all inputs and outputs from separate transactions to form a single transaction, and the signatures can be composed by the parties. A Transaction Join can be understood as a simple way to perform CoinJoin with no composite signatures.

Definition 4.10 (Transaction Join). *Given a valid transaction t_0 and an aggregate transaction tx:*

$$t_0 = \{i_0, o_0, tk_0, tko_0\} \text{ and } tx = \{i, o, tks, tko\}$$

a new aggregate transaction can be constructed as:

$$tx = \{i_0 \mid\mid i, o_0 \mid\mid o, tk_0 \mid\mid tks, tko_0 + tko\}$$

The validity of an aggregate transaction is guaranteed by the validity of the transactional parties during the construction process.

Lemma 4.11 (Invariant: CoinJoin Validity). *Given a valid transaction t_0 and a valid aggregate transaction tx.Let tx' be the result of aggregating t_0 into tx as in Definition 4.10. Then, tx' is valid.*

Proof.

Let $t_0 = \{i_0, o_0, tk_0, tko_0\}$ be a transaction with $tk_0 = \{rp_0, ke_0, \sigma_0\}$. Let $tx = \{i, o, tks, tko\}$ be an aggregate transaction with $tks = [tk_1, ..., tk_t]$, the list of transaction kernels where each tk_i is $tk_i = \{rp_i, ke_i, \sigma_i\}$.

Applying Definition 4.10, we have that the resulting tx' is of the form:

$$tx' = \{i', o', tks', ko'\}$$
$$\text{with } i' = i_0 \mid\mid i,\ o' = o_0 \mid\mid o,\ tks' = [tk_0, tk_1, ..., tk_t],\ ko' = tko_0 + ko$$

According to Property 4.8, we need to show that:
i) *the range proofs of all the transaction outputs are valid.*
It means that, according to Definition 4.4, it is necessary to prove that:
for all $rp_j \in rp,\ Verify(o_j, rp_j) = true$ where $rp = [rp_0, rp_1, ..., rp_t]$
Since, t_0 and tx are valid transactions, in particular it holds that the range proofs of all the transaction outputs are valid:

- transaction t_0: $Verify(o_0, rp_0) = true$
- transaction tx: *for all* $rp_j \in rp,\ Verify(o_j, rp_j) = true$ where $rp = [rp_1, rp_1, ..., rp_t]$

ii) *the transaction tx' is balanced.*
According to Definition 4.9, we need to prove the following equality holds for the aggregate transaction tx':

$$\sum_{o_j \in o'} o_j - \sum_{c_j \in i'} c_j = ko'.G + \sum_{ke_j \in tks'} ke_j$$

Each term can be written as follows:

$$(\sum_{o_j \in o_0} o_j + \sum_{o_j \in o} o_j) - (\sum_{c_j \in i_0} c_j + \sum_{c_j \in i} c_j) = (tko_0 + ko).G + ke_0 + \sum_{ke_j \in tks} ke_j$$

Rearranging the equality and using algebraic properties on elliptic curves, we have:

$$(\sum_{o_j \in o_0} o_j - \sum_{c_j \in i_0} c_j) + (\sum_{o_j \in o} o_j - \sum_{c_j \in i} c_j) = (ke_0 + tko_0.G) + (ko.G + \sum_{ke_j \in tks} ke_j) \tag{4.3}$$

Now, we apply the hypothesis concerning the validity of t_0 and tx. In particular, applying Definition 4.5 for t_0 and Definition 4.9 for tx, we have the following equalities are true:

$$\sum_{o_j \in o_0} o_j - \sum_{c_j \in i_0} c_j = ke_0 + tko_0.G \tag{4.4}$$

and

$$\sum_{o_j \in o} o_j - \sum_{c_j \in i} c_j = ko.G + \sum_{ke_j \in tks} ke_j \tag{4.5}$$

Now, if we substitute the left part of Equation 4.3 with the right parts of Equation 4.4 and Equation 4.5, we have:

$$(ke_0 + tko_0.G) + (ko.G + \sum_{ke_j \in tks} ke_j) = (ke_0 + tko_0.G) + (ko.G + \sum_{ke_j \in tks} ke_j)$$

iii) *all the kernel signatures are valid for the excess.*

The list of transaction kernels of tx' is $tks = [tk_0, tk_1, ..., tk_t]$ where each tk_i is $tk_i = \{rp_i, ke_i, \sigma_i\}$.

We need to prove that, for each $i \in \{0, .., t\}$, σ_i is valid for the excess ke_i which holds trivially:

- since t_0 is a valid transaction, according to Property 4.3, σ_0 is valid for the excess ke_0 .
- since tx is a valid aggregate transaction, according to Property 4.8, for each $i \in \{1, .., t\}$, σ_i is valid for the excess ke_i.

□

Although in our model aggregate transactions and blocks are essentially the same, we are interested in distinguishing them. That is because the unconfirmed transaction pool will contain aggregate transactions and the chain will contain blocks. Since our idealized model is being built in an incremental iterative way, this distinction allow us to identify and add components in a separate way. For instance, we could add and analyze block headers to state validity conditions over the chain. On the other hand, aspects of different security properties will be analyzed on aggregate transactions and blocks. In an aggregate transaction, an adversary could find out which input cancel which output. They could try all possible permutations and verify if they summed to the transaction excess. The property of transaction unlinkability will be proved over blocks, as we will see in Property 5.10.

4.4 Unconfirmed Transaction Pool

The unconfirmed transaction pool (mempool) contains the transactions which have not been confirmed in a block yet.

Definition 4.12 (Mempool). *A mempool mp is a list of type:*

$$\mathit{Mempool} \overset{\text{def}}{=} \mathit{AggregateTransaction}^*$$

4.5 Blocks and chains

The genesis block *Gen* is a special block since it is the first block ever recorded in the chain. Transactions can be merged into a *block*. We can see a block as a big transaction with aggregated inputs, outputs and transaction kernels.

Definition 4.13 (Block). *A Block b is either the genesis block Gen, or a tuple of type:*

$$Block \overset{\text{def}}{=} \{i : I^*,\ o : O^*,\ tks : TxKernel^*, ko : KOffset\}$$

where:

- *$i = [c_1, ..., c_n]$ and $o = [o_1, ..., o_m]$ are the lists of inputs and outputs of the transactions.*
- *$tks = [tk_1, ..., tk_t]$ is the list of t transaction kernels.*
- *$ko \in \mathbb{F}_n$ is the block kernel offset which covers all the transactions of the block.*

We can say a block is balanced if each aggregated transaction is balanced.

Definition 4.14 (Balanced Block). *Let $b = \{i, o, tks, ko\}$ be a block with $tks = [tk_1, ..., tk_t]$ the list of transaction kernels where the j-th item in tks is of the form $tk_j = \{rp_j, ke_j, \sigma_j\}$. We say the block b is balanced if the following holds:*

$$\sum_{o_j \in o} o_j - \sum_{c_j \in i} c_j = ko.G + \sum_{ke_j \in tks} ke_j$$

We assume the genesis block Gen is valid. We define the notion of block validity as follows:

Property 4.15 (Valid Block). A block b is valid ($valid\_block(b)$) if b is the genesis block Gen or it satisfies:

i. The block is balanced.
ii. For every transaction kernel, the range proofs of all the outputs are valid and the kernel signature σ is valid for the transaction excess.

Blocks can be constructed by aggregating transactions as follows:

Definition 4.16 (Block Aggregation). *Given a valid transaction t_0 and a valid block b as follows:*

$$t_0 = \{i_0, o_0, tk_0, tko_0\} \text{ and } b = \{i, o, tks, ko\}$$

a new block can be constructed as:

$$b' = \{i_0 \mid\mid i, o_0 \mid\mid o, tk_0 \mid\mid tks, tko_0 + ko\}$$

Block aggregation preserves the validity of blocks; i.e. block validity is invariant with respect to block aggregation.

Lemma 4.17 (Invariant: Block Validity). *Given a valid transaction t_0 and a valid block b as in Definition 4.16. Let b' be the result of aggregating t_0 into b. Then, b' is valid.*

Proof.

Let $t_0 = \{i_0, o_0, tk_0, tko_0\}$ be a transaction with $tk_0 = \{rp_0, ke_0, \sigma_0\}$. Let $b = \{i, o, tks, ko\}$ be a block with $tks = (tk_1, ..., tk_t)$, the list of transaction kernels.

Applying Definition 4.16, we have that the resulting b' is of the form:

$$b' = \{i', o', tks', ko'\}$$
$$\text{with } i' = i_0 \mid\mid i,\ o' = o_0 \mid\mid o,\ tks' = (tk_0, tk_1, ..., tk_t),\ ko' = tko_0 + ko$$

According to Property 4.15, we need to prove:

i) *the block b' is balanced.*

According to Definition 4.14, we need to prove the following equality holds for the block b':

$$\sum_{o_j \in o'} o_j - \sum_{c_j \in i'} c_j = ko'.G + \sum_{ke_j \in tks'} ke_j$$

Each term can be written as follows:

$$(\sum_{o_j \in o_0} o_j + \sum_{o_j \in o} o_j) - (\sum_{c_j \in i_0} c_j + \sum_{c_j \in i} c_j) = (tko_0 + ko).G + ke_0 + \sum_{ke_j \in tks} ke_j$$

Rearranging the equality and using algebraic properties on elliptic curves, we have:

$$(\sum_{o_j \in o_0} o_j - \sum_{c_j \in i_0} c_j) + (\sum_{o_j \in o} o_j - \sum_{c_j \in i} c_j) = (ke_0 + tko_0.G) + (ko.G + \sum_{ke_j \in tks} ke_j)$$

Now, we apply the hypothesis concerning the validity of t_0 and b. In particular, applying Definition 4.5 for t_0 and Definition 4.14 for b, we have the following equalities are true:

$$\sum_{o_j \in o_0} o_j - \sum_{c_j \in i_0} c_j = ke_0 + tko_0.G$$

and

$$\sum_{o_j \in o} o_j - \sum_{c_j \in i} c_j = ko.G + \sum_{ke_j \in tks} ke_j$$

ii) *for every transaction kernel in b', the range proofs of all the outputs are valid and the kernel signature is valid for the transaction excess.*

Since t_0 is a valid transaction and b is a valid block it holds trivially as we have shown in Lemma 4.11. □

In our model, a chain is defined as a list of blocks.

Definition 4.18 (Chain). *A chain is a non-empty list of blocks:*

$$Chain \overset{\text{def}}{=} Block^*$$

For a chain c and a valid block b, we can define a predicate $validate(c, b)$ representing the fact that is correct to add b to c. This relation must verify, for example, that all the inputs in b are present as outputs in c, in other words, they are UTXOs.

4.6 Validating a chain

The model formalizes a notion of valid state that captures several well-formedness conditions. In particular, every block in the blockchain must be valid. A predicate $validChain$ can be defined for a chain $c = (b_0, b_1, \dots b_n)$ by checking that:

- b_0 is a valid genesis block
- For every $i \in \{1, \dots n\}$, $validate((b_0, \dots, b_{i-1}), b_i)$

The axiomatic semantics of the system are modeled by defining a set of transactions, and providing their semantics as state transformers. The behavior of transactions is specified by a precondition *Pre* and by a postcondition *Post*:

$$Pre \subseteq State \times Transaction$$

$$Post \subseteq State \times Transaction \times State$$

This approach is valid when considering local (nodes) or global (blockchain) states (of type *State*) and transactions (of type *Transaction*). Different sets of transactions, pre and postcondition are defined to cover local or global state transformations. At a general level, *State* is *Chain*.

5 Properties

Since we are dealing with virtual money, we should guarantee privacy and security properties on our idealized model. In particular, the property of ownership ensures that only the owner of the coins can spend them. Furthermore, we should prevent an attacker from spending a coin more than once and creating virtual money from thin air. Next, we detail some relevant properties that can be verified in our model. In addition to some of the properties mentioned in previous chapters, we have included other properties such as those formulated in [PS18], and various security properties considered in [GKL15, KRDO17, FOS19].

5.1 Protocol Properties

The property of *no coin inflation* or *zero-sum* guarantees that no new funds are produced from thin air in a valid transaction. The property can be stated as follows.

Lemma 5.1 (No Coin Inflation). *Given a valid transaction $t = \{i, o, tk, tko\}$ with transaction kernel $tk = \{rp, ke, \sigma\}$, then the transaction excess only contains the blinding factor and the kernel offset.*

Proof.

We know the transaction t is valid, in particular, the transaction is balanced. Applying Definition 4.5, we know that:

$$\sum_{o_j \in o} o_j - \sum_{c_j \in i} c_j = ke + tko.G$$

Using Definition 4.1, we start to unfold the terms in the equality:

$$\sum_1^m r'.G + v'.H - \sum_1^n r.G + v.H = (\sum_1^m r' - \sum_1^n r - tko).G + tko.G$$

Applying algebraic properties on elliptic curves, we have:

$$\sum_1^m v'.H - \sum_1^n v.H = (\sum_1^m r'.G - \sum_1^n r.G) - (\sum_1^m r'.G - \sum_1^n r.G) - tko.G + tko.G = 0$$

Therefore,

$$(v'_1 + ... + v'_m).H - (v_1 + ... + v_n).H = (v'_1 + ... + v'_m - v_1 - ... - v_n).H = 0.H = 0$$

It means that all the inputs and outputs add to zero. In other words, they summed to the commitment to the kernel offset plus the commitment to the excess blinding factor. □

Thus, we have proved that no coins are being created or destroyed in the transaction. In addition, we have seen that a valid transaction guarantees that all the range proof outputs are valid, which means that every transactional output is positive.

An important feature of MW is the cut-through process. The purpose of this process is to erase redundant outputs that are used as inputs within the same block. Let C be a list of coins that appear as an output in the block b. If the same coins appear as an input within the block, then C can be removed from the list of inputs and outputs after applying the cut-through process. In this way, the only remaining data are the block headers, transaction kernels and UTXOs. After applying cut-through to a valid block b it is important to ensure that the resulting block b' is still valid. We can say that the validity of a block should be invariant with respect to the cut-through process.

Lemma 5.2 (Invariant: Cut-through Block Validity). *Let $b = \{i, o, tks, ko\}$ be a block with i and o the list of inputs and outputs, $tks = [tk_1, ..., tk_t]$ the list of transaction kernels and ko the block kernel offset. Let $b' = \{i', o', tks, ko\}$ be the resulting block after applying the cut-through process to b where:*

- $i' = i \setminus (i \cap o)$
- $o' = o \setminus (i \cap o)$

Hence, if b is a valid block, then b' is valid too.

Proof.

Let $b = \{i, o, tks, ko\}$ be a block with $tks = [tk_1, ..., tk_t]$ the list of transaction kernels, where the j-th item in tks is of the form $tk_j = \{rp_j, ke_j, \sigma_j\}$.

Let r be $r = i \cap o = \{r_0, r1, ..., r_n\}$ where we assume $r \neq \emptyset$ because otherwise the lemma holds trivially as $b' = b$.

Let $b' = \{i', o', tks, ko\}$ be a block with $tks = [tk_1, ..., tk_t]$, the list of transaction kernels, $i' = i \setminus r$ and $o' = o \setminus r$.

We need to prove that b' is valid. According to Property 4.15, we need to show that:
i) *The block b' is balanced.*

According to Definition 4.14, we need to prove:

$$\sum_{o_j \in o'} o_j - \sum_{c_j \in i'} c_j = ko.G + \sum_{ke_j \in tks} ke_j$$

By hypothesis, we know that b is a valid block. Applying Property 4.15, we know that b is balanced. According to Definition 4.14, the following equality holds for block b:

$$\sum_{o_j \in o} o_j - \sum_{c_j \in i} c_j = ko.G + \sum_{ke_j \in tks} ke_j$$

Applying the definition of r, we can rewrite the above equality as follows:

$$(\sum_{o_j \in o \setminus r} o_j + \sum_{o_j \in r} o_j) - (\sum_{c_j \in i \setminus r} c_j + \sum_{c_j \in r} c_j) = ko.G + \sum_{ke_j \in tks} ke_j$$

Rearranging the equality, we have:

$$(\sum_{o_j \in o \setminus r} o_j - \sum_{c_j \in i \setminus r} c_j) + (\sum_{o_j \in r} o_j - \sum_{c_j \in r} c_j) = ko.G + \sum_{ke_j \in tks} ke_j$$

Now, we can observe that we are subtracting the sum of all the elements belonging to the same set r. Thus, the term is equal to zero.

Then, if we remove the term we have:

$$\sum_{o_j \in o \setminus r} o_j - \sum_{c_j \in i \setminus r} c_j = ko.G + \sum_{ke_j \in tks} ke_j$$

By hypothesis, we know that $i' = i \setminus r$ and $o' = o \setminus r$; therefore we can rewrite the above equality as:

$$\sum_{o_j \in o'} o_j - \sum_{c_j \in i'} c_j = ko.G + \sum_{ke_j \in tks} ke_j$$

ii) *For every transaction kernel, the range proofs of all the outputs are valid, and the kernel signature σ is valid for the transaction excess.*

Since the range proofs of the remaining outputs are the same, they remain valid. According to definition 4.6, a valid signature for the kernel excess is defined in terms of the list of range proofs of the outputs rp, transaction excess ke, and the kernel signature σ. Notice that these three elements remain unchanged during the cut-through process. Since b is a valid block, it holds trivially.

$\square$

5.2 Privacy and Security Properties

In blockchain systems the notion of privacy is crucial: sensitive data should not be revealed over the network. In particular, it is desirable to ensure properties such as confidentiality, anonymity and unlinkability of transactions. Confidentiality refers to the property of preventing other participants from knowing certain information about the transaction, such as the amounts and addresses of the owners. Anonymity refers to the property of hiding the real identity of the parties involved in a transaction. In contrast, unlinkability refers to the inability of linking different transactions of the same user within the blockchain.

In the case of MW no addresses or public keys are used; there are only encrypted inputs and outputs. Privacy concerns rely on confidential transactions, cut-through and CoinJoin. As we have seen, CoinJoin combines inputs and outputs from different transactions into a single aggregated transaction and cut-through removes outputs and inputs spent within the same block. We have shown in Lemma 4.11 and Lemma 5.2 that the validity of transaction and block is guaranteed after applying both processes.

The security problem of double spending refers to spending a coin more than once. All the nodes keep track of the UTXO set, so before confirming a block to the chain, the node checks that the inputs come from it. If we refer to our model, that validation is performed in the predicate *validate* mentioned in Chapter 4.5.

5.2.1 Security properties of Pedersen commitments

In MW transactions, input and output amounts are hidden in Pedersen commitments. In Chapter 3.1 we have introduced the definition of a commitment scheme (Definition 3.1).

A commitment scheme is expected to satisfy the following two security properties:

- Hiding: the receiver, who received the commitment, does not learn anything about the original value.
- Binding: after the commit stage, there is at most one value that the sender can successfully open.

In the cryptocurrencies world, these two properties should be understood this way:

- Hiding: a commitment scheme is used to keep the transactions secure. The sender commits to an amount of coins and this should remain private for the rest of the network over time.
- Binding: senders cannot change their commitments to a different transaction amount. If that were possible, it would mean that an adversary could spend coins which have already been committed to an UTXO, what would allow to create coins out of thin air.

There are two possible specifications for these properties. Computational hiding or binding is when all adversaries, running in polynomial time, can break the security property with negligible probability. This asymptotic security is parameterized by a security parameter λ and adversaries run in polynomial time in λ and their other inputs. On the other hand, we talk about perfectly hiding or binding, when even with infinite computing power it would be not possible to break the security property.

Notice that a commitment scheme cannot be perfectly hiding and binding at the same time:

i. If the scheme is perfectly hiding, there must exist several inputs committing to the same value. Otherwise, an adversary with infinite computing power attempting to find out which input committed to a certain output, could try all possible inputs finding out the corresponding output. This shows that this scheme cannot be perfectly binding.
ii. If the scheme is perfectly binding, it means that there is at most one input that committed to an output. Imagine an adversary with infinite computing power attempting to find out which input committed to a target output. It would be possible to try all inputs and find which one verifies the commitment. Thus, this scheme cannot be perfectly hiding.

So, for cryptocurrencies systems is better to provide stronger security in order to guarantee the hiding property. In other words, we prefer a commitment scheme with computational binding and perfectly hiding. We can understand this by first assuming adversaries break the binding property. It means that they could create money from thin air from a certain point in time but this would not affect the blockchain history. On the other hand, if the adversary breaks the hiding property, history could be inspected and all the transactions revealed which breaks one of the main principles of a privacy-oriented cryptocurrency.

Pedersen commitments are computational binding

This property relies on the discrete logarithm assumption. In provable security, security is proved to hold against any probabilistic time adversary by showing an efficient way to break the cryptography protocol implies a way to break the underlying mathematical problem which is supposed to be hard (security reduction). The adversary is modeled as a procedure.

Definition 5.3 (Computational Binding Commitment).
Let $\zeta(Setup, Com)$ be a Pedersen commitment scheme as in Definition 3.2. Let $\mathcal{A}_{Binding}$ be a polynomial probabilistic time adversary against the binding property running in the context of the game $G_{Binding}$ as in Figure 5.1. We say that the Pedersen commitment scheme ζ is computational binding if the success probability of $\mathcal{A}_{Binding}$ winning game $G_{Binding}$ is negligible.

In game $G_{Binding}$, the scheme is first set up by choosing two generator points, G and H, over the elliptic curve $\mathcal{C}$ of prime order n. All these parameters are public.

Secondly, the adversary $\mathcal{A}_{Binding}$ performs the attack attempting to find out two different transactional values v_1 and v_2 that commit to the same commitment. Once the adversary finishes the attack, two pair of different opening values are returned. The adversary succeeds if both pairs commit to the same value and $v_1 \neq v_2$.

```
Game $\mathsf{G}_{\mathsf{Binding}} \overset{\mathrm{def}}{=}$
    $(G, H, n) \leftarrow SetUp(1^\lambda)$
    $(v_1, r_1), (v_2, r_2) \leftarrow \mathcal{A}_{Binding}(G, H, n)$;
    return $Com(v_1, r_1) = Com(v_2, r_2) \wedge v_1 \neq v_2$
```

Fig. 5.1. Game Binding Commitment

As we mentioned before, the computational binding property is based on a security reduction. In terms of Pedersen commitment, it means that if the adversary $\mathcal{A}_{Binding}$ could perform the attack in the context of game $G_{Binding}$ and could win with non-negligible probability, an adversary $\mathcal{E}_{Dlog}$ attacking a game against the discrete logarithm problem on the group $\mathcal{C}$ could use $\mathcal{A}_{Binding}$ to win the game with non-negligible probability.

Recall that MW uses Pedersen commitment with elliptic curves (Definition 3.2). The discrete logarithm problem on this context means: given a point y over the elliptic curve $\mathcal{C}$ with generator G, it is hard to find x such that $y = x.G$. In this case, its security is shown against the discrete logarithm relation assumption which is as hard as breaking the discrete logarithm problem. It means that an adversary cannot find a non-trivial discrete logarithm relation between generators of a group independently chosen. In our case, finding a non-trivial discrete logarithm relation between G and H over the elliptic curve $\mathcal{C}$.

The following lemma captures the semantics of that security reduction.

Lemma 5.4 (Computational Binding). *Let $\zeta(Setup, Com)$ be a Pedersen commitment scheme as in Definition 3.2. Let $\mathcal{A}_{Binding}$ be an adversary against the computational binding commitment (Definition 5.3) in the commitment scheme ζ.*

Let us assume that $\mathcal{A}_{Binding}$ succeeds in finding two distinct pair of opening values that commit to the same commitment with ϵ probability. Therefore, there exists an extractor $\mathcal{E}_{Dlog}$ which can find out a non-trivial discrete logarithm relation between the generators G and H, independently chosen, on the elliptic curve $\mathcal{C}$ with ϵ' probability using the adversary $\mathcal{A}_{Binding}$.

Hence, if ϵ is non-negligible, ϵ' is non-negligible too.

Proof.

The goal of the proof is to show how to transform the efficient adversary $\mathcal{A}_{Binding}$ that is able to break the computational binding commitment into an algorithm that efficiently solves the discrete logarithm assumption. The extractor $\mathcal{E}_{Dlog}$ will provide a simulation context in which the adversary $\mathcal{A}_{Binding}$ will perform its attack. The attack of the extractor $\mathcal{E}_{Dlog}$ will be successful if $\mathcal{A}_{Binding}$ is successful and the simulation does not fail.

According to game $G_{Binding}$, when the adversary $\mathcal{A}$ succeeds we have two identical commitments $Com(v_1, r_1) = Com(v_2, r_2)$ and $v_1 \neq v_2$ such that (Definition 3.2):

$$r_1.G + v_1.H = r_2.G + v_2.H$$

So we can compute:

$$H = \frac{r_1 - r_2}{v_2 - v_1}.G$$

which means that we have computed the discrete logarithm of H with respect to G.

Figure 5.2 shows the game G_{DLog} which captures the semantic of the reduction. The failure event captures when the adversary $\mathcal{A}_{Binding}$ fails and therefore, the adversary $\mathcal{E}_{Dlog}$ fails too.

The probability of success of $\mathcal{E}_{Dlog}$ is equal to the probability of success of $\mathcal{A}_{Binding}$. □

```
Game $\mathsf{G}_{\mathsf{Dlog}} \overset{\text{def}}{=}$
    $(G, H, n) \leftarrow SetUp(1^\lambda)$
    $y \leftarrow \mathcal{E}_{Dlog}(G, H, n)$
    if $y = failure$ then
        return $failure$
    else
        return $H = y.G$

$\mathcal{E}_{Dlog}(G, H, n) \overset{\text{def}}{=}$
    $(v_1, r_1), (v_2, r_2) \leftarrow \mathcal{A}_{Binding}(G, H, n)$
    if $Com(v_1, r_1) = Com(v_2, r_2) \wedge v_1 \neq v_2$ then
        return $\frac{r_2 - r_1}{v_2 - v_1}$
    else
        return $failure$
```

Fig. 5.2. Game Extractor DLog

Pedersen commitments are perfectly hiding

Basically, it is because, given a commitment $Com(v, r) = r.G + v.H$, there are many combinations of (v', r') that satisfies $Com(v, r) = r'.G + v'.H$. Despite the adversary have infinite computing power and could attempt all possible values, there would be no way to know which opening values (v', r') were the original ones. Furthermore, r is a random value of the finite field $\mathbb{F}_n$ so $r.G + v.H$ is a random element of $\mathcal{C}$.

Definition 5.5 (Perfectly Hiding Commitment). *Let $\zeta(Setup, Com)$ be a Pedersen commitment scheme as in Definition 3.2. Let $\mathcal{A}_{Hiding}$ be a computationally unbounded adversary against the hiding property running in the context of the game G_{Hiding} as in Figure 5.3. We say that the Pedersen commitment scheme ζ is perfectly binding if the success probability of $\mathcal{A}_{Hiding}$ winning game G_{Hiding} holds:*

$$Pr(b = b') = \frac{1}{2}$$

```
Game $\mathsf{G}_{\mathsf{Hiding}} \overset{\mathrm{def}}{=}$
    $(G, H, n) \leftarrow SetUp(1^{\lambda})$
    $(v_0, v_1) \leftarrow \mathcal{A}_{Hiding}(G, H, n)$
    $b \xleftarrow{\$} \{0, 1\}$
    $r \xleftarrow{\$} \mathbb{F}_n$
    $com = Com(v_b, r)$
    $b' \leftarrow \mathcal{A}_{Hiding}(com, G, H, n)$
    return $b = b'$
```

Fig. 5.3. Game Hiding Commitment

In the game described in Figure 5.3, first the game is set up and then the adversary chooses two distinct transactional values v_0 and v_1. Then, one of these values is randomly chosen as v_b, as well as with the blinding factor r. The commitment of (v_b, r) is computed and the adversary $\mathcal{A}_{Hiding}$ performs the attack attempting to find out which one of the values was committed.

5.2.2 Switch commitments

As already mentioned, if an attacker succeeds in breaking the computational bindingness property of a commitment then money can be created from thin air. Switch commitments [RM17] were introduced to enable the transition from *computational bindingness* to *statistical bindingness*, specially to the commitments stored in the blockchain. The notion of statistical security implies that a computationally unbounded adversary cannot violate the property except with negligible probability.

If in a certain moment we believe that the bindingness of the commitment scheme gets broken, we could make a soft fork on the chain and switch existing commitments to this new validation scheme which is backwards compatible.

Below, we show the changes that are needed for our model to also encompass Switch commitments. In Pedersen commitment definition (Definition 3.2) we add a third point generator J of the elliptic curve $\mathcal{C}$ whose discrete logarithm relative to G and H is unknown. We define the new commitment algorithm as follows:

$$Com(v, r) = r'.G + v.H, \textit{with } v \textit{ the transactional value and}$$
$$r' = r + hash(v.H + r.G, r.J),$$

where r is the blinding factor randomly chosen in the finite field $\mathbb{F}_n$ and J is the third point generator.

Note that r is still randomly distributed and the hash value of *ElGamal* commitment is computed which is the combination of ElGamal encryption [Gam84] and a commitment scheme.

5.2.3 Security properties of range proofs

The goal of zero-knowledge proofs is to prove that a statement is true without revealing any information beyond the verification of the statement. In MW we need to ensure that in every transaction the amount is positive so that users cannot create coins. Here, the hard part is to prove that without revealing the amount. In our model, the output amounts are hidden in the form of a Pedersen commitment, and the transaction contains a list of range proofs of the outputs to prove that the amount is positive. In our model, this verification is performed as the first step of the validation of the transaction (Property 4.3).

In Chapter 3.2 we have defined a range proof scheme (Definition 3.3) where rp is a range proof for a certain value v in some range $[a, b]$. Range proofs are generated by the *Prove* algorithm and verified by the *Verify* algorithm.

The aim of the prover is to convince the verifier that the following statement holds: the Pedersen commitment $c = Com(v, r) = r.G+v.H$ contains a transactional value v such that $v \in [0, 2^n]$. We say that the range proof is an argument and v is a witness. In other words, the prover should provide a zero-knowledge (ZK) proof about the range of the output. Range proofs will commit to the range of $[0, 2^n]$ where n is small enough to no cause overflow errors. The transaction kernel tk of each transaction (Definition 4.1) has a list rp of range proofs of the outputs.

A range proof scheme is expected to satisfy the following three properties:

- Completeness: if the statement holds for a witness v, the argument provided by the prover is able to convince the verifier;
- Soundness: if the statement does not hold for a witness v, the prover cannot convince the verifier about the statement; and
- Zero-knowledge: the argument does not leak any information about the witness, except for whether the statement is true or false.

In our model, a range proof scheme is expected to have perfect completeness and computational soundness with negligible error ϵ_S. We define these properties as follows:

Definition 5.6 (Perfect Completeness NIZK Scheme). *Let $\zeta(Setup, Com)$ be a Pedersen commitment as in Definition 3.2 where $c = Com(v, r)$ is the commitment to the transactional value v and the blinding factor r. Let $\eta(Prove, Verify)$ be a NIZK scheme as in Definition 3.3. We say $\eta(Prove, Verify)$ has perfect completeness if for every v in the range $[a, b]$, exists a range proof rp such that $rp = Prove(v, r, c)$ and $Verify(c, rp) = true$.*

Definition 5.7 (Computational Soundness NIZK Scheme). *Let $\zeta(Setup, Com)$ be a Pedersen commitment as in Definition 3.2 where $c = Com(v, r)$ is the commitment to the transactional value v and the blinding factor r. Let $\eta(Prove, Verify)$ be a NIZK scheme as in Definition 3.3. We say $\eta(Prove, Verify)$ has $\epsilon_S -$ soundness if for every $v \notin [a, b]$ and $rp = Prove(v, r, c)$, it holds that $Pr(Verify(c, rp) = true) \leq \epsilon_S$.*

Recall that, the *Prove* algorithm computes a non-interactive zero-knowledge proof to the commitment in order to verify that the committed value is in certain range. We need to ensure that the proof does not leak any information about the secret value, other than the fact it lies in a certain range.

Definition 5.8 (Perfect ZK Range Proof). *Let $\zeta(Setup, Com)$ be a Pedersen commitment as in Definition 3.2, where $c = Com(v, r)$ is the commitment to the transactional value v and the blinding factor r. Let $\eta(Prove, Verify)$ be a NIZK scheme as in Definition 3.3 where rp is a range proof such that $rp = Prove(v, r, c)$. Let $\mathcal{A}_{ZKw}$ be a computationally unbounded adversary running in the context of the game G_{ZKw} as in Figure 5.4. We say that the range proof rp is perfect zero-knowledge if the success probability of $\mathcal{A}_{ZKw}$ winning game G_{ZKw} is $Pr(b = b') = 0.5$. In other words, the adversary cannot learn anything about the opening value v from rp.*

Fig. 5.4. Game Zero-knowledge range proof

```
Game $\mathsf{G}_{\mathsf{ZKw}} \overset{\text{def}}{=}$
    $(G, H, n) \leftarrow SetUp(1^\lambda)$
    $(v_0, v_1) \leftarrow \mathcal{A}_{ZKw}(G, H, n)$
    $b \xleftarrow{\$} \{0, 1\}$
    $r_0 \xleftarrow{\$} \mathbb{F}_n$
    $r_1 \xleftarrow{\$} \mathbb{F}_n$
    $com_0 = Com(v_0, r_0)$
    $com_1 = Com(v_1, r_1)$
    $rp = Prove(v_b, r_b, com_b)$
    $b' \leftarrow \mathcal{A}_{ZKw}(rp, com_0, com_1, G, H, n)$
    return $b = b'$
```

In Figure 5.4, first the game is set up and then the adversary chooses two distinct transactional values v_0 and v_1. Then, one of these values is randomly chosen as v_b, as well as with the blinding factors r_0 and r_1. Both commitments of (v_0, r_0) and (v_1, r_1) are computed. Then, a zero-knowledge proof *rp* is generated for the value v_b. Finally, the adversary $\mathcal{A}_{ZKw}$ performs the attack attempting to find out which value $\{v_0, v_1\}$ corresponds to the range proof *rp*; in other words, for which value holds $Verify(com_b, rp) = true$.

Bulletproofs [BBB+18] is a non-interactive zero-knowledge proof protocol. A bulletproof is a short proof (logarithmic in the witness size) with the aim of proving that the committed value is in a certain range without reveling it. Proof generation and verification times are linear in the length of the range. Moreover, aggregation of range proofs is supported which enables the parties to generate a single proof, without revealing their inputs to each other. They do not require a trusted setup and are non-interactive applying the Fiat-Shamir heuristic [BR93] [FS87] which replaces the verifier's random challenge by a hash. MW uses Bulletproofs, which helps to maintain its size more compact than other Bitcoin blockchains as it uses a compact version of the inner product argument (*rp* in Definition 3.3). Bulletproofs are computational soundness and rely on the discrete logarithm assumption. In this case, its security is shown against the discrete logarithm relation assumption which means that an adversary cannot find a non-trivial discrete logarithm relation between generators of a group independently chosen. In our model, it means to be proved over the elliptic curve $\mathcal{C}$ and it could be finding a non-trivial discrete logarithm relation between G and H (Definition 4.1). Solving this is as hard as breaking the discrete logarithm problem. We could say that the discrete logarithm relation problem can be reduced to the computational soundness security property.

Lemma 5.9 (Computational Soundness Bulletproofs). *Let $\zeta(Setup, Com)$ be a Pedersen commitment scheme as in Definition 3.2. Let $\eta(Prove, Verify)$ be a NIZK scheme as in Definition 3.3. Let $\mathcal{A}_{ZK\_Soundness}$ be a ppt adversary against the computational soundness bulletproofs in the NIZK scheme η. Let us assume that $\mathcal{A}_{ZK\_Soundness}$ succeeds in finding a range proof rp' for a transactional value $v' \notin [a, b]$ and $c' = Com(v', r')$ such that $Verify(c', rp') = true$ with ϵ probability. Therefore, there exists an adversary $\mathcal{E}_{Dlog}$ which can find out a non-trivial discrete logarithm relation between the generators G and H, independently chosen, on the elliptic curve $\mathcal{C}$ with ϵ' probability using the adversary $\mathcal{A}_{ZK\_Soundness}$. Hence, if ϵ is non-negligible, ϵ' is non-negligible too.*

This proof is discussed for an arithmetic circuit protocol over a group G in Theorem 5 of the Bulletproof paper [BBB$^+$18]. The protocol shown has a knowledge extractor which either extracts a discrete logarithm relation or a valid witness. However, if the generators were independently generated, then finding a discrete logarithm relation between them is as hard as breaking the discrete log problem.

Hence, it follows that if breaking computational soundness is easy, then so is breaking the logarithm discrete problem which is supposed to be hard. In other words, the computational soundness property is secure against the stated attack.

So far we have analyzed that Pedersen commitments are computational binding and perfectly hiding [SBCL21a]. On the other hand, Bulletproofs are perfect zero-knowledge and computational soundness. Let us suppose we have a commitment c such that $c = Com(r, v) = r.G + v.H$ and the *Prover* computes the range proof rp for the value v as in the range proof scheme Definition 3.3. Now, we need to pass c and rp to the *Verifier* in order to verify whether rp the argument is valid. Since we know the property of perfect completeness holds, the *Verifier* will accept rp. In other words, an honest *Prover* succeed in convincing the *Verifier*. Moreover, since the commitment is perfectly hiding we have not revealed any information about the commitment by passing it to the *Verifier*.

5.3 Unlinkability and Untraceability

As we specified in our model, each node has a pool of unconfirmed transactions in the *mempool*. This transactions are waiting for the miners in order to be included in a block. We can distinguish two security properties of the transactions. Untraceability refers to the transactions in the mempool and unlinkability to the transactions in the block. In our model, this two notions are formalized as follows.

Property 5.10 (Transaction Unlinkability). Given a valid block b, it is computationally infeasible to know which input cancels which output.

The following lemma captures the semantics of this property. Moreover, the operations cut-through and CoinJoin, which were described above, also contribute to this property.

Lemma 5.11 (Transaction Unlinkability). *For any valid block b and for any polynomial probabilistic time adversary $\mathcal{A}$, the probability of $\mathcal{A}$ in finding a balanced transaction within b is negligible.*

Proof.

Let $b = \{i, o, tks, ko\}$ be a valid block with $tks = (tk_1, ..., tk_t)$ the list of transaction kernels. The j-th item in tks is of the form $tk_j = \{rp_j, ke_j, \sigma_j\}$.

The goal of the adversary $\mathcal{A}$ is to find a tuple of the form $\{i', o', ke'\}$ where the list of inputs i' is a subset of i and the list of outputs o' is a subset of o, satisfying Definition 4.5 of a balanced transaction. It means that, the following equality must be true for the tuple:

$$\sum_{o_j \in o'} o_j - \sum_{c_j \in i'} c_j = ke' + tko'.G$$

where ke' is the transaction excess and tko' the transaction kernel offset.

If we refer to the construction process in Definition 4.16, the transaction kernel offsets were added to generate a single aggregate offset ko to cover all transactions in the block. It means that we do not store the individual kernel offset tko' of the transaction in b once the transaction is aggregated to the block.

The challenge is trying to solve the adversary $\mathcal{A}$ could be seen as the subset sum problem (NP-complete) but, in this case, tko' is unrecoverable. So, although many transactions have few inputs and outputs, it is computationally infeasible, without knowing that value, to find the tuple. □

Then, we can define:

Definition 5.12 (Transaction Unlinkable). *We say block b is transaction-unlinkable, if the probability of any polynomial probabilistic time adversary $\mathcal{A}$ in finding a balanced transaction within b is negligible.*

Hence, due to Lemma 5.11 we conclude that in MW all blocks are transaction-unlinkeable.

Property 5.13 (Transaction Untraceability). For every transaction in the mempool, it is not possible to relate the transaction to the IP address of the node which originated it.

Regarding this property, we should refer mainly, to the broadcast of the transactions. Once the transactions are created, they are broadcasted to the network and they are included in the mempool. Each node could track the IP address from the node which received the transaction. At that point nodes could record the transactions, allowing them to build a transaction graph.

We define that the broadcast of a transaction can be performed with or without confusion. Without confusion means that, once the transactions are created, they are broadcasted immediately to all the network. However, if someone controls enough nodes on the network and discovers how the transaction moves, he could find out the IP address node from which the transaction comes from.

On the other hand, we define the broadcast with confusion as a way to obscure the IP address node.

Definition 5.14 (Broadcast with confusion). *Let's say node A sends a transaction to node B. We say B receives the transaction with confusion if given the IP address of node A, the node B does not know if the transaction was originated by the node A or not.*

In other words, it can be said that if some malicious nodes, working together, construct a graph of the pairs $(transaction, IP\ address\ node)$, the IP address node will not convey information about what node originated the transaction. Therefore, in our model, we require Property 5.14 to hold before the broadcast takes place. In order to achieve this, we can establish that the node broadcasting the transaction should be far enough from the one which originated it. Moreover, CoinJoin could be performed before the broadcast.

Dandelion, proposed by Bojja et al. [BVFV17], is a protocol for transaction broadcasting intended to resist that deanonymization attack. Dandelion is not part of the MW protocol, however this kind of protocols should be implemented by each node to lower the risk of creating the transaction graph. In Dandelion, broadcasting is performed in two phases: the "steam" phase and the "fluff" phase. In the "steam" phase the transaction is broadcasted randomly to one node, which then randomly sends it to another, and so on. This process finishes when the "fluff" phase is reached, and the transaction is broadcasted to the network using a gossip protocol.

Next, we define the following routines which capture the semantic of the phases:

```
$routine\ steam(tx : Transaction)\{$
$c \xleftarrow{\$} \{0,1\}$ (* random decision *)
$if\ \ c == 0\ then$
  $node \leftarrow select\_random\_node()$
  $node.steam(tx)$
$else$
  $this.fluff(tx)$
$\}$

$routine\ fluff(tx : Transaction)\{$
$broadcast(tx)$
$\}$
```

Each node, besides having the local state, should implement these two routines. Once the transaction is created and is ready to be included in the mempool, its broadcasting start in the "steam" phase. When it reaches the "fluff" phase, it is broadcasted to the network and added in the mempool.

Dandelion relies on the following three rules: all nodes obey the protocol, each node generates exactly one transaction, and all nodes on the network run Dandelion. The problem is that an adversary can violate them. For that reason, `Grin` implements a more advanced protocol called Dandelion++ [FVB$^+$18] which intends to prevent that [Gria]. However, it is believed that Dandelion++ is not good enough to guarantee the privacy of a virtual coin [Grie]. For instance, the flashlight attack [Mie] is an open problem still under investigation [Grif]. The scenario here is when an 'activist' want to accept donations but he cannot reveal his identity. At some point, he will deposit those payments to an exchange and his identity would be compromised. The adversary injects 'tainted coins' and could build a 'taint tree' looking through all deposits to the exchange. This way, he could link those deposits to the 'activist'.

The combined use of the MW protocol with a Zerocash-style commitment-nullifier schema has been put forward in [Lim] as a countermeasure to the above attack. In the case of Zcash, every shielded transaction has a large anonymity set, namely, a set of transactions form which it is indistinguishable from. In the case of Spectrecoin (now Alias) [KMS] the main idea is the use, only once, of public addresses (XSPEC) to receive the payments combined with an anonymous staking protocol.

6 Implementations

Because of the its robust security, privacy and scalability, there are several implementations of Mimblewimble. In 2019, the first two practical implementations were launched: `Grin` and `Beam`. Although, there are some design and technical differences in both projects, they implement and extend the core of the MW protocol. Next, we first describe the main features of their design and compare them with our model. Then, we discuss features that set apart `Grin` from `Beam`.

6.1 Grin

`Grin` [Grid] is an open source software project with a simple approach to MW. As we will see below, its design is a straightforward interpretation of our model.

6.1.1 Blocks and Transactions

In order to provide privacy and confidentiality guarantees, `Grin` transactions are based on confidential transactions. In Figure 8.1 (Appendix 8.2), we can observe that each transaction contains a list of inputs and outputs. Each input and output is in the form of a Pedersen commitment, i.e, a linear combination of the value of the transaction and a blinding factor. For instance, in the input structure (Appendix 8.2, Figure 8.2, line 1729), there is a field that stores the commitment pointing to the output being spent.

In addition, the transaction structure has a list of transaction kernels (of type *TxKernel*) with the transaction excess and the kernel signature. All this data has a straightforward relation to our definition of transaction (Definition 4.1).

However, it is important to notice that the transaction kernel structure differs from our model since it does not contain the list of range proofs of the outputs. In `Grin`, it is part of the output structure (Appendix 8.2, Figure 8.3, line 2045).

Moreover, a Grin transaction also includes the block number at which the transaction becomes valid. We have not added this data to the transaction structure yet and we also should include it in the signature process. In Grin, not only the transaction fee is signed, the signing process also takes into account the absolute position of the blocks in the chain. In this way, if a kernel block points to a height greater than the current one, it is rejected. If the relative position points to a specific kernel commitment, Grin has the same behavior.

Grin Blocks also stores a kernel offset which is the sum of all the transaction kernel offsets added to the block. In our model, the kernel offset is defined within a block (Definition 4.13) and the notion of adding a transaction into a block is formalized on the block aggregation (Definition 4.16). Besides, the single aggregate offset allows to prove Lemma 5.11 as part of the Transaction Unlinkability property (Property 5.10).

6.1.2 Privacy and Security Properties

The cut-through process, as explained in Chapter 5.1, provides scalability and further anonymity. Grin performs this process in the transaction pool, which we formalized as *mempool* (Definition 4.12). Outputs which have already been spent as new inputs are removed from the mempool, using the fact that every transaction in a block should sum to zero.

CoinJoin, as we have mentioned in Chapter 5.2, combines inputs and outputs from multiple transactions into a single transaction in order to obfuscate them. In Grin, every block is a CoinJoin of all other transactions in the block.

In addition, Grin supports a *pruning process*. This process could be applied to past blocks. Outputs that have been spent in a previous block are removed from the block. Block validity (Property 4.15) should be invariant w.r.t. the pruning process. Each node maintains a local state with a local copy of the chain. The pruning process can be applied recursively to the chain and keep it as compact as possible. Pruning is useful to free space. As a consequence, when a new node wants to join the network, it can receive just a pruned (i.e. partial) chain and the node needs to validate it, which makes the synchronization process faster. In Chapter 4.6, *validChain* should be modified to guarantee the validity of a partial chain.

As we have mentioned in Chapter 5.2, Switch commitments provides perfect hiddenness and statistical bindingness. Grin implements a switch commitment [Gric] as part of a transaction output in order to provide more security than computational bindingness (Definition 5.3), which is crucial for the age of quantum adversaries.

6.2 Beam

Beam [Bead] was the other Mimblewimble project launched on January 2019. This open source system has a founding model and a dedicated development team.

6.2.1 Blocks and Transactions

`Beam` transactions are confidential transactions implemented by the Pedersen commitment scheme. This follows the same approach as our model.

Figure 8.4 in Appendix 8.2 shows (line 439) how `Beam`'s input stores the commitment, i.e, a point over the elliptic curve (class of *ECC::Point*).

In Chapter 4 we have described how each node maintains a local state. The state keeps track of the UTXO set. `Beam` extends the behavior of that set, supporting the incubation period on a UTXO. This means that `Beam` sets the minimum number of blocks created after the UTXO entered the blockchain, before it can be spent in a transaction. This number is included in the transaction signature. Figure 8.5 in Appendix 8.2 shows (line 510) how `Beam`'s output stores the number of blocks corresponding to the incubation period.

If we specify this functionality in our model, the predicate *validChain* (Chapter 4.6) should check that every output with certain incubation period on a block was 'lawfully' spent for the entire blockchain (global state). In other words, if we have an output transaction o with an incubation period d on a confirmed block b over the chain and a later confirmed block b' containing o as an input, then b' should be, at least, d blocks away from b on the blockchain.

6.2.2 Privacy and Security Properties

`Beam` supports cut-through as we described above. In addition, `Beam` adds a scalable feature to eliminate all intermediate transaction kernels [Beaa] in order to keep the blockchain as compact as possible. It would be important to prove that the resulting transaction is still valid in Property 4.3.

6.3 Discussion

Both `Grin` and `Beam` implementations address the main features of the MW protocol, namely the properties of confidentiality, anonymity and unlinkability comprised in our work.

6.3.1 Broadcasting Protocol

Both `Grin` and `Beam` use the Dandelion scheme as broadcasting protocol [BVFV17]. We have formalized that a broadcasting protocol should hold Property 5.13 of Transaction Untraceability. It should not be possible to link transactions and their originating IP addresses, in other words, to deanonymize users. Broadcast with confusion, as we describe in Property 5.14, should be carried out to satisfy Transaction Untraceability. We have also described the steam and fluff phases of the Dandelion scheme.

`Grin`'s implementation, in the steam phase, allows for transaction aggregation (CoinJoin) and cut-through, which provides greater anonymity to the transactions before they are broadcasted to the entire network.

In addition, in order to improve privacy, `Beam`'s implementation adds dummy transaction outputs at the steam phase. Each output has a value of zero and it is indistinguishable from regular outputs. Later, after a random number of blocks, the UTXOs are added as inputs to new transactions, i.e., they are spent and removed from the blockchain.

In Chapter 5.3 we have specified the steam routine. Following, we extend the routine to capture `Beam`'s behavior:

```
$subroutine\ steam(tx : Transaction)\{$
$i \xleftarrow{\$} \{min, .., max\}$ (* incubation period random choice *)
(* create zero value output with incubation period i *)
$zeroOut \leftarrow createZeroValueOutput(i)$
$addOutputTransactionUTXO(zeroOut)$
$addOutputTransaction(zeroOut, tx)$
...
$\}$
```

To capture that semantic, we have combined two `Beam`'s features: incubation period on UTXO and aggregation of zero value transaction outputs. Firstly, we randomly choose i as an incubation period. Then, we create a zero value transaction output ($zeroOut$) with incubation period i. The incubation period will ensure not to spend the dummy output before i blocks are confirmed on the chain. After that, $zeroOut$ is added to the UTXO set (which is maintained in the local state of the node) and to the transaction tx that is being broadcasted. Finally, the routine continues as we specified in Chapter 5.3.

6.3.2 Range proofs

`Grin` and `Beam` implement range proofs using Bulletproofs [BBB$^+$18]. Bulletproofs are a non-interactive zero-knowledge proof protocol. They are short proofs (logarithmic in the witness size) with the aim of proving that the committed value is in a certain (positive) range without reveling it. Proof generation and verification times are linear in the length of the range. Regarding our model, it is the first property a transaction should satisfy to be valid (Property 4.3). Furthermore, for every transaction in a bock, the range proofs of all the outputs should be valid too (Property 4.15).

6.3.3 Some Design Decisions

Emission Scheme

It is known that BitCoin has a limited and finite supply of coins. Nowadays, new coins come from the process called “mining” where miners are paid because of their work of aggregating new blocks to the chain besides of the transaction fees. However, once the maximum amount of coins in circulation is reached, there will not be new coins and the miners will be paid only with the transaction fees.

`Grin` has a different approach. It has a static emission rate, where a fixed number of coins is released as a reward for aggregating a new block to the chain. This algorithm has no upper bound for new coins. However, `Beam` has a capped total supply standing at 262M. The reward algorithm is decreasing over the years [Beab].

Parties Negotiation

Mimblewimble establishes that communication between the parties to construct a new transaction is made off-chain. Parties should collaborate in order to choose blinding factors and construct a valid transaction, in particular, a balanced transaction as in Definition 4.5. `Grin` offers this process synchronously. Both parties are connected directly to one another and they should be online simultaneously.

On the other hand, in order to construct a new transaction, `Beam` offers a non-interactive negotiation between the parties. The Secure Bulletin Board System (SBBS) [Beae] runs on the nodes and it allows the parties to communicate off-line. Moreover, `Beam` also presents a one-side payment scheme. This scheme allows senders to pay a specified value to a particular receiver, without any interaction from the receiver side. The key here is not revealing blinding factors. It is addressed with a process called kernel fusion. Basically, both parties construct a half kernel and both kernels should be present in the transaction.

Chain Synchronization

`Grin` allows partial history synchronization. When a new node wants to enter the network, it is not necessary to download the full history of the chain but it will query the block header at a horizon. The node can increase this limit as necessary. Then, it will download the full UTXO set of the horizon block.

`Beam` improves node synchronization using macroblocks. A macroblock is a compressed version of blockchain history after applying the cut-through process. Each node stores macroblocks locally. When a new node connects to the network, it will download the latest macroblock and will start working from that point.

7 Conclusions and Future Work

7.1 Final remarks

MW constitutes an important step forward in the protection of anonymity and privacy in the domain of cryptocurrencies. Since it facilitates traceability and the validation process, both `Grin` and `Beam` have adopted the MW protocol for their implementations.

We have highlighted elements that constitute essential steps towards the development of an exhaustive formalization of the MW cryptocurrency protocol, the analysis of its properties and the verification of its implementations. The proposed idealized model is key in the described verification process. First of all, we have defined the main components of our idealized model: transactions, blocks and chain. Then, we have provided validity conditions to guarantee the correctness of the blockchain. We have stated precise conditions for a valid transaction and a valid block. Furthermore, we have defined and proved that the validity of a block is invariant with respect to the cut-through process and CoinJoin.

The main difficulty we have faced during that process was the lack of "official" documentation, so we have made an exhaustive literature review in order to analyze and conceptualize the main components of MW. Furthermore, even when Grin and Beam have documentation available on-line, the main challenge was to build a model which abstracts away the specifics of their implementations.

We have also identified and precisely stated the conditions for our model to ensure the verification of relevant security properties of MW which is an important contribution of this work. Firstly, we have proved that no new funds are produced from thin air in a valid transaction. Secondly, since MW transactions are in the form of Pedersen commitments, we have analyzed the strength of the scheme regarding the main security properties a cryptocurrency protocol must have. In particular, we have defined the computational binding commitment property and we have shown a security reduction using a game-based cryptographic proof approach. Thirdly, we have discussed zero-knowledge proofs to prove that the transaction output is positive without revealing the amount. Furthermore, we have described some security properties a range proof scheme should satisfy.

Then, we have defined and analyzed two important security properties: unlinkability and untraceability. In particular, we have proved that the probability in finding a balanced transaction within a valid block is negligible. Moreover, we have defined what broadcast with confusion is in order to obscure the IP address from which the transaction comes from.

Finally, we have analyzed and compared the `Grin` and `Beam` implementations in their current state of development, considering our model and its properties as a reference base. We have presented detailed connections between our model and their implementations regarding the MW structure and its security properties. In particular, we have extended our *steam* routine abstraction to capture Beam's behavior combining dummy transactions and incubation period in order to improve privacy.

The main challenge we have faced to address the comparison between our model and the implementations was reading the source code of Grin and Beam to identify the components of our model and analyze how they implement them.

The present work is not free of limitations. First, transaction construction is made off-chain between the parties so the protocol construction should be analyzed in order to guarantee security and privacy properties during the communication. In addition, some of the security properties rely on an underlying hard problem in terms of provable security. On the other hand, the consensus protocol plays an important role to ensure the integrity of the recorded information. In order to validate blocks and add them to the chain, the network must agree on a consensus algorithm. Two of the more commonly used are *Proof of Work* and *Proof of Stake*. Weaknesses in the protocol could result in various attack and security should be studied.

7.2 Future work

Since cryptographic proofs are becoming increasingly error-prone and difficult to check, we plan to carry out a specification of our MW model using an interactive prover, in order to provide an automated verification of the model. Security goals and hardness assumptions shall be modeled in order to verify the security properties we have stated. Firstly, we plan to evaluate tools for the verification of cryptographic protocols and implementations, such as `EasyCrypt` [BDG+13], `ProVerif` [Bla01], `CryptoVerif` [Bla18] and `Tamarin` [Tam]. In particular, we are especially interested in using `EasyCrypt`, an interactive framework for verifying the security of cryptographic constructions in the computational model. Secondly, we will specify our model using the tool, according to all definitions we have stated in this work. Then, all the properties we have presented and proved should be verified using the interactive prover. Furthermore, we shall specify and verify the security properties. In particular, the game-based cryptographic proof in Lemma 5.4 where the goal is to construct a security reduction as a sequence of games proving that any attack against the security of the system would lead to an efficient way to solve the discrete logarithm problem.

The results presented in this work constitute a relevant contribution in order to analyze the correctness of the MW protocol and its security properties over an idealized model beyond any particular implementation. Directions for future research are to verify that Grin and Beam are a correct implementation of the idealized model in order to guarantee security properties with a formal approach. Indeed, Guy Corem, part of the *beam.mw* foundation and one of Beam's founders, contacted us and showed interest in our work. He spread it by his personal Twitter account and Beam's Twitter account. In addition, our paper is mentioned on Beam's website [1].

[1] `https://beam.mw/resources`

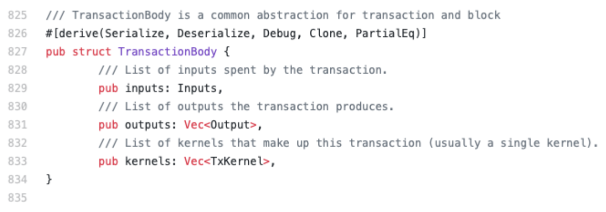
8 Appendix

8.1 Math symbols and abbreviations

Table 8.1 shows the meaning of the abbreviations and acronyms used throughout this work. Table 8.2 describes the significance of the math symbols used throughout this work.

8.2 Grin and Beam source code

Figures 8.1, 8.2 and 8.3 show a part of the Grin transaction source code where each transaction contains a list of inputs and outputs in the form of a Pedersen commitment.

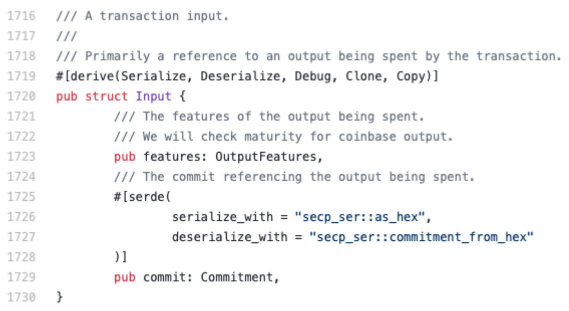


```
/// TransactionBody is a common abstraction for transaction and block
#[derive(Serialize, Deserialize, Debug, Clone, PartialEq)]
pub struct TransactionBody {
        /// List of inputs spent by the transaction.
        pub inputs: Inputs,
        /// List of outputs the transaction produces.
        pub outputs: Vec<Output>,
        /// List of kernels that make up this transaction (usually a single kernel).
        pub kernels: Vec<TxKernel>,
}

```

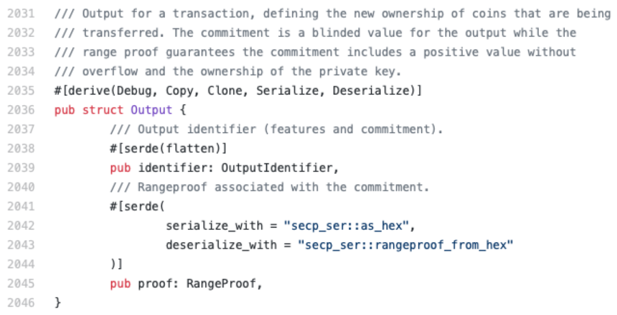
Fig. 8.1. `Grin` transaction body source code [Grib]

Figure 8.4 exhibits a part of the Beam transaction source code where it stores the commitment i.e, a point over the elliptic curve (class of ECC::Point). In addition, Figure 8.5 shows how Beam's output stores the number of blocks corresponding to the incubation period.



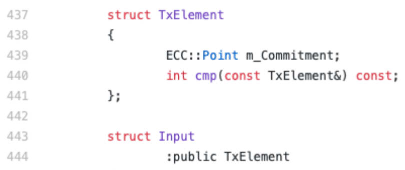
```rust
/// A transaction input.
///
/// Primarily a reference to an output being spent by the transaction.
#[derive(Serialize, Deserialize, Debug, Clone, Copy)]
pub struct Input {
        /// The features of the output being spent.
        /// We will check maturity for coinbase output.
        pub features: OutputFeatures,
        /// The commit referencing the output being spent.
        #[serde(
                serialize_with = "secp_ser::as_hex",
                deserialize_with = "secp_ser::commitment_from_hex"
        )]
        pub commit: Commitment,
}
```

Fig. 8.2. `Grin` input source code [Grib]



```rust
/// Output for a transaction, defining the new ownership of coins that are being
/// transferred. The commitment is a blinded value for the output while the
/// range proof guarantees the commitment includes a positive value without
/// overflow and the ownership of the private key.
#[derive(Debug, Copy, Clone, Serialize, Deserialize)]
pub struct Output {
        /// Output identifier (features and commitment).
        #[serde(flatten)]
        pub identifier: OutputIdentifier,
        /// Rangeproof associated with the commitment.
        #[serde(
                serialize_with = "secp_ser::as_hex",
                deserialize_with = "secp_ser::rangeproof_from_hex"
        )]
        pub proof: RangeProof,
}
```

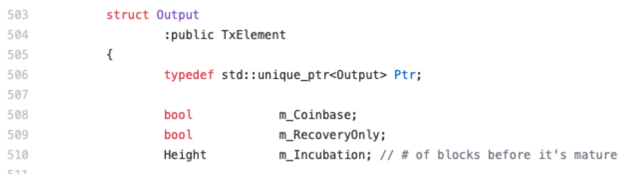
Fig. 8.3. `Grin` output source code [Grib]



```cpp
        struct TxElement
        {
                ECC::Point m_Commitment;
                int cmp(const TxElement&) const;
        };

        struct Input
                :public TxElement
```

Fig. 8.4. `Beam` input source code [Beac]



```cpp
        struct Output
                :public TxElement
        {
                typedef std::unique_ptr<Output> Ptr;

                bool            m_Coinbase;
                bool            m_RecoveryOnly;
                Height          m_Incubation; // # of blocks before it's mature
```

Fig. 8.5. Output incubation period source code [Beac]

Abbreviation	Meaning
MW	mimblewimble
EVM	ethereum virtual machine
FM	formal methods
ECC	elliptic curves cryptography
UTXO	unspent transaction output
NIZK	non-interactive zero-knowledge
IP	internet protocol
ZK	zero-knowledge
Pr	probability
XSPEC	SpectreCoin
IRC	internet relay chat
ECC	elliptic-curve cryptography
SBBS	secure bulletin board system

Table 8.1. Abbreviations

Math symbol	Meaning
$+$	addition
$-$	subtraction
$.$	multiplication
$\Leftrightarrow$	if and only if
$\in$	set membership
X^*	list of elements of type X
$\sum$	summation - sum of all values in range of series
$\leq$	less than equal
$x \overset{\$}{\leftarrow} X$	x is drawn uniformly at random from X
$\|\|$	list concatenation
$\|$	bitstring concatenation
$\forall$	for all

Table 8.2. Math symbols

References

[AMJ21] Jaehyung An, Alexey Mikhaylov, and Sang-Uk Jung. A linear programming approach for robust network revenue management in the airline industry. *Journal of Air Transport Management*, 91:101979, 2021.

[And] James P. Anderson. Computer Security Technology Planning Study, u.s. air force electronic systems division, 1972. Available online: `https://apps.dtic.mil/sti/citations/AD0758206` (accessed on February 23, 2022).

[BBB$^+$18] B. Bünz, J. Bootle, D. Boneh, A. Poelstra, P. Wuille, and G. Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 315–334, May 2018.

[BC04] Yves Bertot and P. Castéran. *Interactive Theorem Proving and Program Development : Coq'Art : The Calculus of Inductive Constructions.* Texts in Theoretical Computer Science. Springer, Isbn: 3540208542, 2004 edition, May 2004.

[BCL$^+$20] Gustavo Betarte, Maximiliano Cristiá, Carlos Daniel Luna, Adrián Silveira, and Dante Zanarini. Towards a formally verified implementation of the mimblewimble cryptocurrency protocol. In Jianying Zhou et al, editor, *Applied Cryptography and Network Security Workshops - ACNS 2020 Satellite Workshops, AIBlock, AIHWS, AIoTS, Cloud S&P, SCI, SecMT, and SiMLA, Rome, Italy, October 19-22, 2020, Proceedings*, volume 12418 of *Lecture Notes in Computer Science*, pages 3–23. Springer, 2020.

[BCL$^+$21] Gustavo Betarte, Maximiliano Cristiá, Carlos Luna, Adrián Silveira, and Dante Zanarini. Set-based models for cryptocurrency software. *Clei-electronic-journal*, abs/1908.00591, 2021.

[BDG$^+$13] G. Barthe, F. Dupressoir, B. Grégoire, C. Kunz, B. Schmidt, and P. Strub. Easycrypt: A tutorial. In Alessandro Aldini, Javier López, and Fabio Martinelli, editors, *Foundations of Security Analysis and Design VII - FOSAD 2012/2013 Tutorial Lectures*, volume 8604 of *Lecture Notes in Computer Science*, pages 146–166. Springer, 2013.

[BDLF$^+$16] K. Bhargavan, A. Delignat-Lavaud, C. Fournet, A. Gollamudi, G. Gonthier, N. Kobeissi, N. Kulatova, A. Rastogi, T. Sibut-Pinote, N. Swamy, and S. Zanella-Béguelin. Formal verification of smart contracts: Short paper. In *Proceedings of the 2016 ACM Workshop on Programming Languages and Analysis for Security*, PLAS '16, pages 91–96, New York, NY, USA, 2016. ACM.

[Beaa] Beam. Beam description. Comparison with classical MW, 2018. Available online: `https://docs.beam.mw/BEAM_Comparison_with_classical_MW.pdf` (accessed on February 23, 2022).

[Beab] Beam. Beam emission schedule. Available online: `https://docs.beam.mw/BEAM_Position_Paper_v0.2.2.pdf` (accessed on February 23, 2022).

[Beac] Beam. Beam project github. Available online: `https://github.com/BeamMW/beam` (accessed on February 23, 2022).

[Bead] Beam. Beam the scalable confidential cryptocurrency. Available online: `https://docs.beam.mw/BEAM_Position_Paper_0.3.pdf` (accessed on February 23, 2022).

[Beae] Beam. Secure bulletin board system (sbbs). Available online: `https://github.com/BeamMW/beam/wiki/Secure-bulletin-board-system-(SBBS)` (accessed on February 23, 2022).

[BGW20] C. Boyd, K. Gjøsteen, and S. Wu. A Blockchain Model in Tamarin and Formal Analysis of Hash Time Lock Contract. In Bruno Bernardo and Diego Marmsoler, editors, *2nd Workshop on Formal Methods for Blockchains (FMBC 2020)*, volume 84 of *OpenAccess Series in Informatics (OASIcs)*, pages 5:1–5:13, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum für Informatik.

[Bla01] Bruno Blanchet. An efficient cryptographic protocol verifier based on prolog rules. In *14th IEEE Computer Security Foundations Workshop (CSFW-14 2001), 11-13 June 2001, Cape Breton, Nova Scotia, Canada*, pages 82–96. IEEE Computer Society, 2001.

[Bla18] Bruno Blanchet. Composition theorems for cryptoverif and application to TLS 1.3. In *31st IEEE Computer Security Foundations Symposium, CSF 2018, Oxford, United Kingdom, July 9-12, 2018*, pages 16–30. IEEE Computer Society, 2018.

[BR93] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *ACM Conference on Computer and Communications Security*, pages 62–73, 1993.

[But] Vitalik Buterin. Critical update re: Dao vulnerability, 2017. Available online: `https://blog.ethereum.org/2016/06/17/critical-update-re-dao-vulnerability` (accessed on February 23, 2022).

[BVFV17] Shaileshh Bojja Venkatakrishnan, Giulia Fanti, and Pramod Viswanath. Dandelion: Redesigning the bitcoin network for anonymity. *Proc. ACM Meas. Anal. Comput. Syst.*, 1(1), June 2017.

[Cré11] Claude Crépeau. Commitment. In Henk C. A. van Tilborg and Sushil Jajodia, editors, *Encyclopedia of Cryptography and Security, 2nd Ed*, pages 224–227. Springer, 2011.

[FOS19] G. Fuchsbauer, M. Orrù, and Y. Seurin. Aggregate cash systems: A cryptographic investigation of mimblewimble. In Y. Ishai and V. Rijmen, editors, *Advances in Cryptology - EUROCRYPT 2019 - 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, May 19-23, 2019,* *Proceedings, Part I*, volume 11476 of *Lecture Notes in Computer Science*, pages 657–689. Springer, 2019.

[Fou] Beam Foundation. Beam confidential cryptocurrency, 2020. Available online: `https://beam.mw/` (accessed on February 23, 2022).

[FS87] Amos Fiat and Adi Shamir. How to prove yourself: practical solutions to identification and signature problems. In *Proceedings on Advances in cryptology—CRYPTO '86*, pages 186–194, London, UK, 1987. Springer-Verlag.

[FVB$^+$18] Giulia Fanti, Shaileshh Bojja Venkatakrishnan, Surya Bakshi, Bradley Denby, Shruti Bhargava, Andrew Miller, and Pramod Viswanath. Dandelion++: Lightweight cryptocurrency networking with formal anonymity guarantees. *Proc. ACM Meas. Anal. Comput. Syst.*, 2(2), June 2018.

[Gam84] Taher El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In *Advances in Cryptology, Proceedings of CRYPTO '84, Santa Barbara, California, USA, August 19-22, 1984, Proceedings*, volume 196 of *Lecture Notes in Computer Science*, pages 10–18. Springer, 1984.

[Gib] A. Gibson. An investigation into confidential transactions, 2018. Available online: `https://github.com/AdamISZ/ConfidentialTransactionsDoc/blob/master/essayonCT.pdf` (accessed on February 23, 2022).

[GKGG21] Ikram Garfatta, Kais Klai, Walid Gaaloul, and Mohamed Graiet. A survey on formal verification for solidity smart contracts. In *2021 Australasian Computer Science Week Multiconference*, ACSW '21, New York, NY, USA, 2021. Association for Computing Machinery.

[GKL15] Juan A. Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol: Analysis and applications. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part II*, volume 9057 of *Lecture Notes in Computer Science*, pages 281–310. Springer, 2015.

[GMS18] Ilya Grishchenko, Matteo Maffei, and Clara Schneidewind. A semantic framework for the security analysis of ethereum smart contracts. In Lujo Bauer and Ralf Küsters, editors, *Principles of Security and Trust - 7th International Conference, POST 2018, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2018, Thessaloniki, Greece, April 14-20, 2018, Proceedings*, volume 10804 of *Lecture Notes in Computer Science*, pages 243–269. Springer, 2018.

[Gria] Grin. Dandelion++ in grin: Privacy-preserving transaction aggregation and propagation, 2019. Available online: `https://github.com/mimblewimble/grin/blob/master/doc/dandelion/dandelion.md` (accessed on February 23, 2022).

[Grib] Grin. Grin project github. Available online: `https://github.com/mimblewimble/grin` (accessed on February 23, 2022).

[Gric] Grin. Grin source code switch commitments. Available online: `https://github.com/mimblewimble/secp256k1-zkp/blob/73617d0fcc4f51896cce4f9a1a6977a6958297f8/src/modules/commitment/main_impl.h#L267` (accessed on February 23, 2022).

[Grid] Grin. Introduction to MimbleWimble and Grin, 2016. Available online: `https://github.com/mimblewimble/grin/blob/master/doc/intro.md` (accessed on February 23, 2022).

[Grie] Grin. Privacy primer, 2018. Available online: `https://github.com/mimblewimble/docs/wiki/Grin-Privacy-Primer` (accessed on February 23, 2022).

[Grif] Grin Community. Grin: Open research problems, 2020. Available online: `https://grin.mw/open-research-problems` (accessed on February 23, 2022).

[Hir17] Yoichi Hirai. Defining the ethereum virtual machine for interactive theorem provers. In Michael Brenner, K. Rohloff, J. Bonneau, A. Miller, P. Ryan, V. Teague, A. Bracciali, M. Sala, F. Pintore, and M. Jakobsson, editors, *Financial Cryptography and Data Security - FC 2017 International Workshops, WAHC, BITCOIN, VOTING, WTSC, and TA, Sliema, Malta, April 7, 2017, Revised Selected Papers*, volume 10323 of *LNCS*, pages 520–535. Springer, 2017.

[HJC20] Ákos Hajdu, Dejan Jovanovic, and Gabriela F. Ciocarlie. Formal specification and verification of solidity contracts with events (short paper). In Bruno Bernardo and Diego Marmsoler, editors, *2nd Workshop on Formal Methods for Blockchains, FMBC@CAV 2020, July 20-21, 2020, Los Angeles, California, USA (Virtual Conference)*, volume 84 of *OASIcs*, pages 2:1–2:9. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

[IGRS16] F. Idelberger, G. Governatori, R. Riveret, and G. Sartor. Evaluation of logic-based smart contracts for blockchain systems. In J. Alferes, L. Bertossi, G. Governatori, P. Fodor, and D. Roman, editors, *Rule Technologies. Research, Tools, and Applications - 10th International Symposium, RuleML 2016, Stony Brook, NY, USA, July 6-9, 2016. Proceedings*, volume 9718 of *LNCS*, pages 167–183. Springer, 2016.

[Jed] T. Jedusor. Mimblewimble, 2016. Available online: `https://scalingbitcoin.org/papers/mimblewimble.txt` (accessed on February 23, 2022).

[KMS] Eirik Korsell, Philip Mueller, and Yves Schumann. Alias, whitepaper, february 9, 2021. Available online: `https://alias.cash/wp-content/uploads/2021/02/Alias-Whitepaper.pdf` (accessed on February 23, 2022).

[KRDO17] Aggelos Kiayias, Alexander Russell, Bernardo David, and Roman Oliynykov. Ouroboros: A provably secure proof-of-stake blockchain protocol. In Jonathan Katz and Hovav Shacham, editors, *Advances* *in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20-24, 2017, Proceedings, Part I*, volume 10401 of *Lecture Notes in Computer Science*, pages 357–388. Springer, 2017.

[LBL] Len LaPadula, D Elliott Bell, and Leonard J LaPadula. Secure computer systems: Mathematical foundations, draft mtr, the mitre corporation, 1973. Available online: `http://www-personal.umich.edu/~cja/LPS12b/refs/belllapadula1.pdf` (accessed on February 23, 2022).

[LCO$^+$16] L. Luu, D. Chu, H. Olickel, P. Saxena, and A. Hobor. Making smart contracts smarter. In E. Weippl, S. Katzenbeisser, C. Kruegel, A. Myers, and S. Halevi, editors, *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, pages 254–269. ACM, 2016.

[Lim] Wanseob Lim. Ethereum 9 3/4: Send erc20 privately using mimblewimble and zk-snarks, 2019. Available online: `https://ethresear.ch/t/ethereum-9-send-erc20-privately-using-mimblewimble-and-zk-snarks/6217` (accessed on February 23, 2022).

[Maxa] G. Maxwell. Coinjoin: Bitcoin privacy for the real world, 2013. Available online: `https://bitcointalk.org/index.php?topic=279249.0` (accessed on February 23, 2022).

[Maxb] G. Maxwell. Confidential transactions write up, 2020. Available online: `https://web.archive.org/web/20200502151159/https://people.xiph.org/~greg/confidential_values.txt` (accessed on February 23, 2022).

[MD17] Roberto Metere and Changyu Dong. Automated cryptographic analysis of the pedersen commitment scheme. In Jacek Rak, John Bay, Igor V. Kotenko, Leonard J. Popyack, Victor A. Skormin, and Krzysztof Szczypiorski, editors, *Computer Network Security - 7th International Conference on Mathematical Methods, Models, and Architectures for Computer Network Security, MMM-ACNS 2017, Warsaw, Poland, August 28-30, 2017, Proceedings*, volume 10446 of *Lecture Notes in Computer Science*, pages 275–287. Springer, 2017.

[Mie] Ian Miers. Blockchain privacy: Equal parts theory and practice, 2019. Available online: `https://www.zfnd.org/blog/blockchain-privacy/#flashlight` (accessed on February 23, 2022).

[MV] Khomyakova l.I. Mishina V.Yu. Dedollarization and settlements in national currencies: Eurasian and latin american experience. *Voprosy Ekonomiki. 2020;(9):61-79 (In Russian).*

[Nak] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008. Available online: `https://bitcoin.org/bitcoin.pdf` (accessed on February 23, 2022).

[Poe] A. Poelstra. Mimblewimble, 2016. Available online: `https://download.wpsoftware.net/bitcoin/wizardry/mimblewimble.pdf` (accessed on February 23, 2022).

[PS18] George Pîrlea and Ilya Sergey. Mechanising blockchain consensus. In June Andronick and Amy P. Felty, editors, *Proceedings of the 7th ACM SIGPLAN International Conference on Certified Programs and Proofs, CPP 2018, Los Angeles, CA, USA, January 8-9, 2018*, pages 78–90. ACM, 2018.

[RCdS20] J. Santos Reis, P. Crocker, and S. Melo de Sousa. Tezla, an Intermediate Representation for Static Analysis of Michelson Smart Contracts. In Bruno Bernardo and Diego Marmsoler, editors, *2nd Workshop on Formal Methods for Blockchains (FMBC 2020)*, volume 84 of *OpenAccess Series in Informatics (OASIcs)*, pages 4:1–4:12, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum für Informatik.

[RM17] Tim Ruffing and Giulio Malavolta. Switch commitments: A safety switch for confidential transactions. In Michael Brenner, Kurt Rohloff, Joseph Bonneau, Andrew Miller, Peter Y. A. Ryan, Vanessa Teague, Andrea Bracciali, Massimiliano Sala, Federico Pintore, and Markus Jakobsson, editors, *Financial Cryptography and Data Security - FC 2017 International Workshops, WAHC, BITCOIN, VOTING, WTSC, and TA, Sliema, Malta, April 7, 2017, Revised Selected Papers*, volume 10323 of *Lecture Notes in Computer Science*, pages 170–181. Springer, 2017.

[Ros20] Grigore Rosu. Formal Design, Implementation and Verification of Blockchain Languages Using K (Invited Talk). In Bruno Bernardo and Diego Marmsoler, editors, *2nd Workshop on Formal Methods for Blockchains (FMBC 2020)*, volume 84 of *OpenAccess Series in Informatics (OASIcs)*, pages 1:1–1:1, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum für Informatik.

[SBCL21a] Adrián Silveira, Gustavo Betarte, Maximiliano Cristiá, and Carlos Luna. A formal analysis of the mimblewimble cryptocurrency protocol. *Sensors*, 21(17), 2021.

[SBCL21b] Adrián Silveira, Gustavo Betarte, Maximiliano Cristiá, and Carlos Luna. A range proof scheme analysis for the mimblewimble cryptocurrency protocol. In *IEEE - UruCon*, 2021.

[SOASdM$^+$20] Iago Sestrem Ochôa, Luis Augusto Silva, Gabriel de Mello, Nuno M. Garcia, Juan Francisco de Paz Santana, and Valderi Reis Quietinho Leithardt. A cost analysis of implementing a blockchain architecture in a smart grid scenario using sidechains. *Sensors*, 20(3), 2020.

[Tam] Tamarin. Tamarin prover. Available online: `https://tamarin-prover.github.io` (accessed on February 23, 2022).

[The] The Coq Team. The Coq proof assistant reference manual. Available online: `http://coq.inria.fr` (accessed on February 23, 2022).

[TLL$^+$21] Palina Tolmach, Yi Li, Shang-Wei Lin, Yang Liu, and Zengxiang Li. A survey of smart contract formal specification and verification. *ACM Comput. Surv.*, 54(7), July 2021.

[Woo] G. Wood. Ethereum: A secure decentralised generalised transaction ledger eip-150 revision (759dccd - 2017-08-07), 2017. Available online: `https://ethereum.github.io/yellowpaper/paper.pdf` (accessed on February 23, 2022).