NTNU

Norwegian University of
Science and Technology

Anonymity for Decentralized Electronic Cash Systems

Siri Dahle

Master of Science
Submission date: June 2018
Supervisor: Kristian Gjøsteen, IMF

Norwegian University of Science and Technology
Department of Mathematical Sciences

Abstract

In 2008 Bitcoin was introduced as the first decentralized electronic cash system and it has seen widespread adoption since it became fully functional in 2009. This thesis describe the Bitcoin system, anonymity aspects for Bitcoin and how we can use cryptography to improve anonymity by a scheme called Zerocoin. The Bitcoin system will be described with focus on transactions and the blockchain where all transactions are recorded. We look more closely into anonymity in terms of address unlinkability and illustrate how the anonymity provided is insufficient by clustering addresses. Further we describe Zerocoin, a decentralized electronic cash scheme designed to cryptographically improve the anonymity guarantees in Bitcoin by breaking the link between individual Bitcoin transactions. We detail the construction of Zerocoin, provide security analysis and describe how it integrates into Bitcoin.

Sammendrag

Bitcoin ble i 2008 introdusert som første desentraliserte elektroniske betalingssystem, og har blitt svært utbredt siden det ble startet å operere i 2009. I denne masteroppgaven ser vi nærmere på Bitcoin, anonymitet i systemet og Zerocoin, et forslag som viser hvordan kryptografi kan benyttes for å forbedre anonymitet i Bitcoin. Bitcoin presenteres med fokus på transaksjoner og blokkjeden, hvor alle transaksjoner lagres. Vi ser nærmere på anonymitet hvor vi ønsker at addresser ikke skal kunne lenkes sammen, og illustrerer hvordan anonymiteten Bitcoin gir ikke er tilstrekkelig. Videre beskriver vi Zerocoin i detalj. Zerocoin er konstruert for å forbedre anynomiteten Bitcoin gir ved bruk av kryptografi, dette gjennom å bryte den sammenhengende kjeden av Bitcoin-transaksjoner. Vi beskriver Zerocoin-konstruksjonen i detalj, sikkerhetsaspekter og hvordan Zerocoin kan integreres med Bitcoin.

Acknowledgements

This master thesis represents the work of my final semester at NTNU, Trondheim. It feels like forever since I decided that writing acknowledgements in my thesis would by the last thing I did as a student, and all of a sudden the time has come. It has been a lot of hard work and I would like to thank the ones whose help along the way is truly appreciated.

To my supervisor Kristian Gjøsteen, thank you for excellent guidance, for always keeping your door open and for answering all my (stupid) questions. I would also like to thank Ian Miers, Christina Garman and Matthew Green for answering my e-mails.

To my friends at Matteland, these past years would never have been the same without you. Thank you for great laughs, for support and for the best study breaks. I am glad to share this with you. We did it!

To my boyfriend, thank you for listening to my complaints (about every day) this semester. Thank you for encouraging me and for always making me feel better. I swear I will be happier from now on.

And last, to my dearest family. I would never be where I am today if it wasn't for you. Thank you for believing in me, for supporting me and for always being there, unconditionally. I can't wait to come home again. I love you.

Trondheim, 01.06.2018
Siri Dahle

Contents

Chapter 1

Introduction

Bitcoin and cryptocurrencies has caused a lot of excitement since Bitcoin was presented as the first decentralized electronic cash system, a payment system without the requirement of a trusted bank. Electronic cash systems can be either centralized or decentralized, and up to this time all has been centralized. Decentralized systems do not rely on a trusted third party to validate transactions but instead rely on cryptography. In 2008 Bitcoin was presented as the first decentralized digital currency, or *cryptocurrency*, and became fully functional in January 2009. Bitcoin is the first cryptocurrency to see widespread adoption and many marketplaces accept payments in bitcoins.

Digital currencies are currencies that do not have intrinsic value and have meaning only to the extent that a government maintains its value or participants agree that they have meaning [31]. The idea of digital currency is not new to the electronic community and was first presented by Chaum in the 1980's [15]. Chaum obtained an anonymous electronic cash scheme where the main idea is that you can design the withdrawal and spending protocols in such a way that it is impossible to identify how a particular coin was spent even though a central authority, such as a bank, is responsible for giving out electronic coins and validating them. This is done by the use of *blind signatures*.

Since that of Chaum, many papers have been published to improve the efficiency and security of electronic cash systems. The *anonymity* provided by electronic cash systems is often of interest. While anonymity can be defined as the state of not being identifiable within a set of subjects, being pseudonymous is the state of using a pseudonym as identification [43]. Bitcoin provides *pseudonymity*. To validate

transactions and prevent double-spending of money, probability the core problem of digital currencies, the entire history of all Bitcoin transactions are recorded in a public ledger and payments are conducted between pseudonyms to avoid tracking by third parties. These pseudonyms works as anonymous addresses and Bitcoin users can create any number of them [20].

The underlying public transactions still imposes a threat to anonymity. The Bitcoin community generally acknowledges this issue which leads to interest in providing stronger anonymity. One of the main groups of proposals are those who provide stronger anonymity but again require more advanced cryptography and modification to Bitcoin.

In this thesis we look more at one of these proposals. First we examine Bitcoin and anonymity aspects. Next we describe how we can use cryptography to improve the anonymity Bitcoin provides by a scheme called *Zerocoin*. The structure of the thesis is as follows:

Chapter 2 The chapter introduces preliminaries and theory for the thesis especially focusing on zero-knowledge protocols.

Chapter 3 After the theory chapter follows a presentation of the Bitcoin system divided in three main parts; transactions, blockchain and mining and the communication network.

Chapter 4 A review of anonymity in the Bitcoin system follows the presentation of it. We give a definition of anonymity and show how it is attacked by address linking.

Chapter 5 Zerocoin is presented as a proposal to improve Bitcoin anonymity, both construction of the scheme and security analysis.

Chapter 6 The final chapter compare the anonymity of Bitcoin and Zerocoin, consider limitations regarding Zerocoin anonymity and briefly examine further work.

More details of the chapters will be provided at the start of each one.

Chapter 2

Theory

In this chapter we present preliminaries and theory that will be of relevance. We start with preliminaries which briefly introduces facts assumed to be known to the reader and some words regarding notation in the thesis. Further we consider cryptographic hardness assumptions, zero-knowledge protocols and commitment schemes.

2.1 Preliminaries

For an integer n, $\mathbb{Z}_n = \{0, 1, ..., n-1\}$ denotes the ring of integers modulo n. The positive integers smaller than and relatively prime to n is denoted $\mathbb{Z}_n^*$ [11]. We write $x \xleftarrow{r} \mathbb{Z}_n$ to denote that x is drawn at random from $\{0, 1, ..., n-1\}$. For every prime p the integers modulo p is a field $\mathbb{F}_p$. Similarly $\mathbb{F}_p^*$ denotes the elements in $\mathbb{F}_p$ that are smaller than and relatively prime to p. We write $\mathbb{G} \subseteq \mathbb{F}_p^*$ to denote that $\mathbb{G}$ is a subgroup of $\mathbb{F}_p^*$. The arithmetic for group elements is performed modulo the order of the group and will not be stated unless it is found convenient. For exponent arithmetic we will explicitly stated how the modular arithmetic is performed.

We can distinguish between *private key cryptography* and *public key cryptography*. For public key cryptography, also known as *asymmetric* cryptography, each user possesses a private key but also a public key [40]. In this thesis we denote the private key by sk and the public key by pk.

Public key cryptography can be used for applications such as *digital signatures*. Digital signatures are one of the most important cryptographic tools [11]. A signature σ on a message m is issued under a public key pk and one can interpret

that the owner of pk and its corresponding key sk has signed message m. Everyone must be able to verify a signature when given the public key of the signer and the message, but only the signer (the party knowing the private key) must be able to compute the signature [14].

The Bitcoin system is built on public key cryptography and more specifically elliptic curve cryptography. Bitcoin makes use of ECDSA, the *elliptic curve digital signature algorithm*. Users in the Bitcoin system are identified by a public key for this signature scheme. The key pair controls access to bitcoins; the public key pk is used to receive bitcoins and the private key sk is used to sign transactions to spend those bitcoins [2].

An essential part of digital signature schemes are *hash functions*. We let H be a collision-resistant hash function that maps binary strings of arbitrary finite length to binary strings of fixed length l [11]. We write $H : \{0,1\}^* \rightarrow \{0,1\}^l$. It can be mentioned that there are different families of hash functions and later we briefly bring up SHA-256 which is a member of the SHA-2 hash functions.

In this thesis we consider electronic cash systems. When referring to *Bitcoin* (capital B) we mean the payment system and when referring to *bitcoin* (lowercase b) we mean the currency, like "one can transfer bitcoins in the Bitcoin system". The same holds for Zerocoin and other systems to be mentioned.

We will use game-based definitions for the security of Bitcoin and Zerocoin. Informally we can define a game as an interaction between an adversary and a simulator. We let $\mathcal{A}$ and $\mathcal{B}$ denote adversaries and $\mathcal{S}im$ denote a simulator that simulates the environment that the adversaries normally would exist in. Sometimes we assume an oracle $\mathcal{O}$ is connected to the communication. Many security definitions hold assuming the hardness of some cryptographic problem and next we turn to describing two hardness assumptions of relevance.

2.2 Cryptographic Hardness Assumptions

Cryptographic constructions can be proven secure relative to some assumption regarding the hardness of solving some problem. The construction is considered secure as long as the problem is hard to solve, but if we can solve the problem we can break the security of the construction [11]. By hard we often mean that there is no algorithm that solves the problem using a reasonable amount of resources. In this thesis we consider two widely used assumptions, the *Discrete logarithm assumption* and the *Strong RSA assumption*.

The Discrete logarithm assumption. The Discrete logarithm problem forms the basis of numerous cryptographic protocols. The discrete logarithm of y to the base g in a group $\mathbb{G}$ is the smallest non-negative integer x such that $y = g^x$. We write $\log_g y = x$. We can note for later purposes that similarly the double discrete logarithm of $y \in \mathbb{G}$ to the bases g and h is the smallest non-negative integer x such that $g^{(h^x)} = y$. There exists efficient algorithms for finding y given g and x, but finding x given y and g is not easy in general [11].

We can define the Discrete logarithm problem as follows: given a finite cyclic group $\mathbb{G}$ of order q with generator g and an element $y \in \mathbb{G}$, find the smallest integer x, $0 \leq x \leq q-1$, such that $g^x = y$ (in this thesis we normally use q to denote a prime, but discrete logarithms are defined for groups of non-prime order as well). The Discrete logarithm assumption is the assumption that this problem is hard to solve. The utility of the Discrete logarithm problem in a cryptographic setting is that finding discrete logarithms are difficult while the inverse operation of exponentiation can be computed efficiently [50].

The Strong RSA assumption. The Strong RSA assumption was first introduced by Barić and Pfitzmann [4]. The RSA problem can be stated as follows: given an RSA modulus n, an exponent e and a random $u \in \mathbb{Z}_n^*$, find $v \in \mathbb{Z}_n^*$ such that $v^e \equiv u \pmod{n}$. The RSA assumption is the assumption that this problem is hard to solve. The flexible RSA problem can be stated as follows: given an RSA modulus n and a random $u \in \mathbb{Z}_n^*$, find $v \in \mathbb{Z}_n^*$ and $e > 1$ such that $v^e \equiv u \pmod{n}$. The Strong RSA assumption is the assumption that this problem is hard to solve. Note that is called the *Strong* RSA assumption. This differs from the ordinary RSA assumption where the exponent e is given [17].

2.3 Zero-knowledge Protocols

An interactive protocol can be seen as a pair of algorithms for two communicating players. For an interactive proof the players are usually a prover $\mathcal{P}$ and a verifier $\mathcal{V}$. The communicating parties are often assumed to be two interactive algorithms with communication tapes that allows them to send and receive messages from one another [11]. We denote a prover that can cheat by $\mathcal{P}^*$ (he does not necessarily get the secret input as $\mathcal{P}$ does) and a verifier that tries to cheat by $\mathcal{V}^*$ (he does not behave as $\mathcal{V}$ but gets the same public input). We consider an interactive proof system for a language L where $y \in L$ means that a claim is true [18, 49]:

Definition 1. An *interactive proof system* for a set L is a two party game between a prover $\mathcal{P}$ and a verifier $\mathcal{V}$ that on public input y satisfies the following:

- *completeness* meaning that for every $y \in L$, $\mathcal{V}$ accepts after interacting with $\mathcal{P}$ on public input y and potentially auxiliary private input x to $\mathcal{P}$, i.e. the prover can convince the verifier that a true statement is indeed true;;

- *soundness* meaning that if $y \notin L$ then $\mathcal{V}$ for any $\mathcal{P}^*$ accepts with at most negligible probability after interacting with $\mathcal{P}^*$ on public input y, i.e. the prover can not convince the verifier that a false statement is true.

A special flavour of interactive proofs is a *proof of knowledge*. In such a proof the prover convinces the verifier that he knows a certain piece of information he claims to know such as a value, or a *witness*, satisfying a certain predicate [11, 18]. A formal definition can be found by Bellare and Goldreich [5]. Let us say that $\mathcal{P}$ wants to prove that $y \in L$. This means that $\mathcal{P}$ must know a witness $w \in W$ for $y \in L$, but he does not want $\mathcal{V}$ to obtain anything else than the single bit of information that $y \in L$.

We let R be a relation and $(y, w) \in R$. We consider an interactive protocol for a prover $\mathcal{P}$ and a verifier $\mathcal{V}$ who both have public input y and $\mathcal{P}$ knows a witness $w \in W$ for $y \in L$. We say that an interactive protocol between a prover $\mathcal{P}$ and a verifier $\mathcal{V}$ is a *proof of knowledge* if there exists an efficient knowledge extractor that for every $\mathcal{P}^*$ which makes $\mathcal{V}$ accepts with non-negligible probability, the extractor can interact with $\mathcal{P}^*$ and with overwhelming probability output a witness w such that $(y, w) \in R$ [28]. Note that proof of knowledge will imply soundness. A proof of knowledge of a witness implies that a witness exists.

Interactive proofs can be *zero-knowledge*. In a zero-knowledge protocol a prover proves a statement to a verifier without revealing anything about the statement other than that it is true [24]. Goldwasser et al. [26] were the first to introduce the concept of zero-knowledge. If we use protocols where we can control exactly how much sensitive information is being released we can keep private information private, even in presence of adversarial behaviour. A protocol is said to be *zero-knowledge* if it communicates exactly the knowledge that was intended and nothing more, i.e. zero additional knowledge [18].

Before we give a formal definition of zero-knowledge we must say something about *indistinguishability*. We can talk about perfect, statistical or computational zero-knowledge according to three kinds of indistinguishability. We say that two

distributions are *indinstinguishable* if it is hard to distinguish them. In the following we consider two distributions Y_1 and Y_2 and we say that the distributions are [11]:

- *perfectly indistinguishable* if $Y_{1y} = Y_{2y}$ for every y, which means that Y_1 and Y_2 are the same probability space and it is not possible to distinguish them at all,

- *statistically indistinguishable* if the statistical distance between Y_1 and Y_2 is negligible, which means that there is a small advantage over a random guess or

- *computationally indistinguishable* if no efficient algorithm exists than can distinguish them, which means that it requires a lot of computational power to decide which of the distribution produced a given output.

Now we can present a formal definition of zero-knowledge [11]:

Definition 2. A protocol ($\mathcal{P}$, $\mathcal{V}$) is (perfect/statistical/computational) *zero-knowledge* if for any verifier $\mathcal{V}^*$ there exists an efficient simulator $\mathcal{S}im$ such that the output produced by $\mathcal{S}im$ is (perfectly/statistically/ computationally) indistinguishable from the output produced by ($\mathcal{P}$, $\mathcal{V}^*$) when $\mathcal{S}im$ and ($\mathcal{P}$, $\mathcal{V}^*$) have the same public input.

Zero-knowledge protocols are normally viewed in a "general cheating verifier" setting. This means that no matter the strategy of the verifier he learns no additional information. We can also consider an honest verifier that must follow the protocol specifications exactly but maintains the ability to keep a record of the entire interaction. This gives rise to the notion of *honest-verifier zero-knowledge* [11, 25, 27, 49]. Note that zero-knowledge implies honest-verifier zero-knowledge since a verifier for zero-knowledge can be malicious. A definition follows [11]:

Definition 3. A protocol ($\mathcal{P}$, $\mathcal{V}$) is (perfect/statistical/computational) *honest-verifier zero-knowledge* (HVZK) if for any verifier $\mathcal{V}$ there exists an efficient simulator $\mathcal{S}im$ such that the output produced by $\mathcal{S}im$ is (perfectly/statistically/ computationally) indistinguishable from the output produced by ($\mathcal{P}$, $\mathcal{V}$) when $\mathcal{S}im$ and ($\mathcal{P}$, $\mathcal{V}$) have the same public input.

Next we look more at honest-verifier zero-knowledge protocols.

2.3.1 Σ-protocols

Most honest-verifier zero-knowledge protocols are Σ-*protocols*. An interactive Σ-protocol is a special type of three-move protocol. The prover sends a commitment α, the verifier answers with a random challenge β and the prover completes the protocol with a response γ [8]. A standard example of a Σ-protocol is that of Schnorr [48] which allows a prover $\mathcal{P}$ to prove knowledge of a discrete logarithm without revealing any knowledge about it other than that he knows it. The protocol is illustrated in Figure 2.1 where, given a cyclic group $\mathbb{G} \subseteq \mathbb{F}_p^*$ of prime order q where $p = 2q+1$ with generator g and $y = g^x$, the prover proves knowledge of x (the arithmetic for group elements is performed modulo p and the exponent arithmetic modulo q).

Public input: p, q, y and g
Private input to $\mathcal{P}$: x such that $y = g^x$

Prover $\mathcal{P}$		Verifier $\mathcal{V}$
$r \xleftarrow{r} \mathbb{Z}_q$		
$\alpha \leftarrow g^r$	$\xrightarrow{\alpha}$	
	$\xleftarrow{\beta}$	$\beta \xleftarrow{r} \mathbb{Z}_q$
$\gamma \leftarrow x\beta + r \bmod q$	$\xrightarrow{\gamma}$	$g^\gamma \stackrel{?}{=} \alpha y^\beta$

Figure 2.1: Schnorr protocol for proving knowledge of a discrete logarithm.

We consider a Σ-protocol for a prover $\mathcal{P}$ and a verifier $\mathcal{V}$ who both have input y and $\mathcal{P}$ wants to prove that $y \in L$. The prover knows a witness $w \in W$ for $y \in L$ such that $(y, w) \in R$. A Σ-protocol has the following properties [27]:

- *completeness* meaning that if $\mathcal{P}$ and $\mathcal{V}$ follow the protocol specifications on input $(y, w) \in R$ where indeed $y \in L$ and the prover knows a witness $w \in W$, the verifier always accepts;
- *special soundness* meaning that from any y and pair of accepting conversations (α, β, γ) and $(\alpha, \beta', \gamma')$ where $\beta \neq \beta'$, one can efficiently extract the witness $w \in W$ for $y \in L$;
- *special honest-verifier zero-knowledge* (SHVZK) meaning there exists a simulator $\mathcal{S}im$ which on public input y and challenge β outputs an accepting

conversation of the form (α, β, γ) that is indistinguishable from a real conversation produced by ($\mathcal{P}$, $\mathcal{V}$).

Note that special soundness implies that the protocol is a proof of knowledge (and sound). We can show that the Schnorr identification protocol in Figure 2.1 has the desired properties. By computing

$$g^{\gamma} = g^{x\beta+r} = g^{r}g^{x\beta} = g^{r}y^{\beta} = \alpha y^{\beta}$$

we see that completeness is satisfied. For special soundness we assume we are given two conversations (α, β, γ) and (α, β', γ') where $\beta \neq \beta'$. One can compute the secret $x = {}^{(\gamma-\gamma')}/_{(\beta-\beta')}$ since we know that $g^{\gamma}y^{-\beta} = g^{\gamma'}y^{-\beta'}$ (x is the witness in the protocol). For special honest-verifier zero-knowledge we assume the existence of a simulator $\mathcal{S}im$ that:

1. selects γ at random,
2. computes $\alpha = g^{\gamma}y^{-\beta}$ and
3. outputs (α, β, γ).

It is impossible to decide whether α or γ was chosen first, hence the distribution of the real and the simulated conversations are indistinguishable and the protocol is special honest-verifier zero-knowledge.

2.3.2 Rewinding

In this thesis we make use of a technique called *rewinding* [32, 45]. The concept of rewinding is a common proof technique in cryptography that for instance can be used to prove that a protocol is sound. Rewinding require that we are able to save states in the protocol such that we can rewind back to the previously saved states. We commonly have an extractor around the prover to perform the rewinding.

We will show an example of rewinding for the Schnorr protocol in Figure 2.1 to make the concept clear. When rewinding is performed for an interactive proof system ($\mathcal{P}^*$, $\mathcal{V}$) the protocol will start as usual and an accepted conversation (α, β, γ) is outputted. Next we *rewind*, which means going back, to a previous stage in the protocol. We give the prover a new random challenge β' and a new accepted conversation (α, β', γ') is outputted. Rewinding for the Schnorr protocol can be performed as follows:

1. Run $\mathcal{P}^* \longleftrightarrow \mathcal{V}$ until $\mathcal{P}^*$ sends α.

2. Remember the state of $\mathcal{P}^*$ and $\mathcal{V}$.

3. Choose $\mathcal{V}$'s challenge β randomly.

4. Run $\mathcal{P}^* \longleftrightarrow \mathcal{V}$ until $\mathcal{P}^*$ sends γ.

5. Rewind.

6. Choose a new challenge β' randomly.

7. Run $\mathcal{P}^* \longleftrightarrow \mathcal{V}$ until $\mathcal{P}^*$ sends γ'.

The rewinding gives us two accepting conversations (α, β, γ) and $(\alpha, \beta', \gamma')$. As we recently showed (Section 2.3.1) two such conversations can be used to compute the secret input and is hence a way to prove that the protocol has special soundness. Later we make use of the rewinding technique for proofs regarding the Zerocoin construction.

2.3.3 From Interactive to Non-interactive

Any honest-verifier zero-knowledge Σ-protocol can be transformed into a *non-interactive* protocol. For non-interactive proofs the prover sends his proof only and the verifier decides to accept or reject the statement without any further interaction [8].

The Fiat-Shamir transform. The most common and efficient technique for constructing non-interactive zero-knowledge proofs is the *Fiat-Shamir heuristic* [8, 22]. To describe the transformation we need a closer look at the *random oracle model*. The random oracle model was formalized by Bellare and Rogaway [6]. A random function is a function $H\colon \{0,1\}^* \to \{0,1\}^l$ such that the output is selected randomly and independent of all other output. In the random oracle model there exist a random oracle with access to such functions over all l so that two equal queries always have the same response. The oracle is exchanged for a cryptographic hash function in the real application.

The idea behind the transformation from interactive to non-interactive is having the prover compute the message of the verifier (the challenge) as the hash of the message sent by the prover. A message computed this way should look random as in an interactive execution if the hash is modelled as a random oracle. We look at a Σ-protocol between a prover $\mathcal{P}$ and verifier $\mathcal{V}$ and let H be a hash function. The weak Fiat-Shamir transform is as follows [8]:

- $\mathcal{P}(y, w)$ Run $\mathcal{P}$ to obtain a commitment α, compute the challenge $\beta \leftarrow H(\alpha)$, obtain the response γ and output (β, γ).
- $\mathcal{V}(y, \beta, \gamma)$ Run $\mathcal{V}$ to compute α' from (y, β, γ) and verify if $\beta = H(\alpha')$.

Note that the prover might as well output (α, γ). The strong Fiat-Shamir is the same as the weak except that β includes the statement to be proved in the hash; $\beta \leftarrow H(y, \alpha)$. If the interactive proof has special soundness and is special honest-verifier zero-knowledge, the non-interactive proof will be sound and zero-knowledge in the random oracle model [27]. Figure 2.2 shows how the protocol in Figure 2.1 is made non-interactive by the (weak) Fiat-Shamir transform.

Public input: p, q, y and g
Private input to $\mathcal{P}$: x such that $y = g^x$

Prover $\mathcal{P}$		Verifier $\mathcal{V}$
$r \xleftarrow{r} \mathbb{Z}_q$		
$\alpha \leftarrow g^r$		
$\beta \leftarrow H(\alpha)$		
$\gamma \leftarrow x\beta + r \bmod q$	$\xrightarrow{(\beta,\gamma)}$	$\alpha' \leftarrow g^\gamma y^{-\beta}$
		$\beta \stackrel{?}{=} H(\alpha')$

Figure 2.2: The Fiat-Shamir transform applied to the Schnorr protocol in Figure 2.1.

Signatures of knowledge. Later we make use of non-interactive zero-knowledge proofs. A type of non-interactive zero-knowledge proofs are *signatures of knowledge*. These are proofs of knowledge that also serve as signatures by including the message to be signed in the hash [13]. We recall from Section 2.1 that signing a message m resulted in a signature σ. Unless an application has reason to trust the public key pk, this alone is not sufficient to trust the message m. Some proof that pk is trustworthy can therefore be required of such an application in addition to (m, σ, pk). We can consider schemes where one is allowed to issue signatures on behalf of any statement $y \in L$ which we refer to as *signatures of knowledge* [14].

We want to be able to issue signatures using knowledge of a witness $w \in W$ such that $(y, w) \in R$ is accepted. We can think of an algorithm for this as a procedure that decides whether $w \in W$ is a witness for $y \in L$ for the language L. The resulting signature can be referred to as a signature of knowledge of $w \in W$ that is a witness

to $y \in L$ on message m, a signature of knowledge of w on message m or a signature of knowledge on behalf of $y \in L$ on message m [14].

We use the notation of Camenisch and Stadler [13] exampled by Miers et al. [36] when referring to these proof. A non-interactive zero-knowledge proof of knowledge of the elements x_1 and x_2 that satisfy both $y_1 = g^{x_1}$ and $y_2 = g^{x_2}$ is denoted

$$\text{NIZKPoK}\{(x_1, x_2) : y_1 = g^{x_1} \wedge y_2 = g^{x_2}\}$$

where the values not enclosed in () are assumed known to the verifier. Similarly the extension

$$\text{ZKSoK}[m]\{(x_1, x_2) : y_1 = g^{x_1} \wedge y_2 = g^{x_2}\}$$

denotes a zero-knowledge signature of knowledge of x_1 and x_2 on message m. Later in the thesis we will describe the construction of such a signature. The construction also has to do with commitments which we next turn to.

2.4 Commitment Schemes

A commitment scheme is a fundamental primitive in cryptographic protocols and for zero-knowledge proofs in particular. Commitments allow a player in a protocol to choose a value from some set and make a commitment to his choice while keeping in hidden to others. Once the commitment is made the player can no longer change his mind. He can, but is not queried to, reveal his choice at a later point of time. For the understanding of commitments we can look at an informal example [18]:

1. $\mathcal{P}$ wants to commit to a bit b. He does so by writing down b on a piece of paper, puts in a box and locks it with a key.

2. $\mathcal{P}$ gives the box to $\mathcal{V}$.

3. $\mathcal{P}$ can later, if he wants to, open the commitment by giving $\mathcal{V}$ the key to the box.

Once $\mathcal{P}$ has given away the box he can no longer change what is inside and when $\mathcal{V}$ recievies the box he can not tell what is inside before $\mathcal{P}$ gives him the key. We refer to these two properties respectively as the *binding property* and the *hiding property* [18]. They are essential to any commitment scheme and we can generalize them as follows:

- the binding property means that it should be hard for $\mathcal{P}$ to change the value he has committed to at a later stage;
- the hiding property means that it must be hard for $\mathcal{V}$ to extract any additional information about the commitment.

A secure commitment scheme is always depending on randomness, if not a powerful opponent could simply compute any commitment and check for matches. For a set of messages, a set of randomness and a set of commitments, to make a commitment c to a message m with randomness r we write $c = commit(m, r)$. The commitment is opened if the tuple (m, r) is revealed and $\mathcal{V}$ can check that $c = commit(m, r)$. The commitment properties can be *unconditional* or *computational* [18]:

- *Unconditional binding* means that even with infinite computing power $\mathcal{P}$ can not change his mind after committing, which means if $\mathcal{P}$ has committed to m using r there is no pair (m', r') where $m \neq m'$ such that $commit(m, r) = commit(m', r')$.
- *Computational binding* means that the chances of being able to change your mind are very small unless you have "very large" computing resources, which means that for two tuples (m, r) and (m', r') we demand that the probability that $commit(m, r) = commit(m', r')$ is negligible.
- *Unconditional hiding* means that a commitment reveals no information about the commitment even to an infinitely powerful $\mathcal{V}$. For a commitment to be unconditionally hiding we need the distribution of commitments to m to be perfectly indistinguishable from the distribution of commitments to m', i.e. for a valid opening to a commitment there exists another valid opening.
- *Computational hiding* means that a bounded $\mathcal{V}$ will have a hard time guessing what is inside a commitment. For a commitment to be computationally hiding we need the distribution of commitments to m to be computationally indistinguishable from the distribution of commitments to m'.

At most one property can be unconditional at any time. It is impossible for a commitment scheme to be both unconditionally binding and unconditionally hiding. If a commitment scheme is unconditionally binding no collision should exist, but for unconditional hiding any pair should have a collision that opens the commitment to a different value.

2.4.1 Pedersen Commitment

A commitment scheme we consider in this thesis is that of Pedersen [42]. Let g, h be generators of $\mathbb{G}$ such that nobody knows $\log_g h$. To commit to a value $m \in \mathbb{Z}_q$, choose $r \in \mathbb{Z}_q$ at random and compute

$$c \leftarrow g^m h^r$$

where the commitment $c = commit(m, r)$ can be opened by revealing m and r.

The scheme is unconditionally hiding. As long as r is chosen uniformly in $\mathbb{Z}_q$ the commitment is uniformly distributed in $\mathbb{G}$. For a commitment $c = commit(m, r)$ there is randomness r that opens the commitment to that value such that all r are equally likely.

The commitment scheme is computationally binding as long as the Discrete logarithm problem is hard. The one committing can not open a commitment to m for $m \neq m'$ unless he can find $\log_g h$. If there exists to openings (m, r) and (m', r') the discrete logarithm of h can be computed as

$$\log_g h = \frac{m - m'}{r' - r}$$

by setting $g^m h^r = g^{m'} h^{r'}$. This closes our theory chapter. We will make use of the theory described in the thesis and it is especially prominent when it comes to the Zerocoin chapter. Before we consider Zerocoin we present the Bitcoin system.

Chapter 3

The Bitcoin System

In this chapter we will describe Bitcoin, the first decentralized electronic cash system. We want to both provide an overview of the Bitcoin system and consider anonymity and proposals for improvement, so this chapter will also review aspects beside what is necessary for the purpose of anonymity considerations.

We will divide the presentation of the Bitcoin system into three parts. First we consider *transactions*, how they are created and how one can send bitcoins to another user in the Bitcoin system. Second we consider the *blockchain*, the core of the Bitcoin system which is the public ledger where all transactions are recorded, and the process of *mining*. Third we consider the communication network on which Bitcoin operates.

In this chapter we will not discuss the Bitcoin client and setup, challenges for the Bitcoin system, different attacks, changes and modification to the Bitcoin system etc., and refer the reader to for example Antonopoulos [2], Bonneau et al., [9], Conti et al. [16] or Narayanan et al. [38] for more on this. Our focus is neither on social or economic aspects but searches to provide a more technical overview. This chapter is based on the work of Nakamoto who presented the Bitcoin white paper [37] but is also inspired by other papers on Bitcoin [1, 2, 3, 9, 16, 20, 29, 30, 38]. We close the presentation of the Bitcoin system by giving a short summary before we proceed to anonymity considerations.

3.1 Transactions

The state of the world in the Bitcoin system is represented by a series of messages called *transactions*. These are data structures that encode the transfer of value between participants in the Bitcoin system. Transactions are an important part of the system whereas everything else in Bitcoin is designed to give assurance that transactions can be created, broadcast on the network, validated and eventually added to the blockchain.

Keys and addresses. Payments in the Bitcoin system are performed through transactions between Bitcoin users. Identities in Bitcoin are represented by ECDSA public keys [29]. Each Bitcoin user U can generate any number of ECDSA key pairs (pk_U, sk_U). A private key sk is a number, usually picked at random, and pk is derived from sk using elliptic curve multiplication. This is a one-way cryptographic function which means it is easy to derive the public key from the secret key but not to derive the secret key from the public key.

The key pairs are used to authorize the transfer of ownership of bitcoins. To receive bitcoins a user U publishes the public key pk_U or its hash as an address. The private key sk_U is used to sign transactions to spend bitcoins. This means that control of the private key of an address denoted adr provides ownership of the bitcoins sent to that address. We will shortly say more about addresses. The keys of a user are stored in a databased called a *wallet*. Private and public keys are stored together as a key pair in most wallet implementations, but since the public key can be derived from the private key it is possible to store only the private key as well.

Users participate in transactions by using pseudonyms and we refer to these as *bitcoin addresses*. There are different address formats in use in the Bitcoin system which means there are different transaction types. A common type is *pay-to-public-key-hash* transactions [9]. Figure 3.1 shows how an address can be created from the public key. We will not go into the details of the figure but want to illustrate that creating an address can be a process with several steps. In the literature an address if often referred to as a hash of the public key, which indeed is true, but note that an address can be further processed after hashing the public key. Since we use one-way functions the public and private key can not easily be derived from the address.

Note that the way in Figure 3.1 is not the only way to create an address, one can for example create addresses directly from the private key as well [2]. A user in the Bitcoin system can create any number of addresses and it is considered good

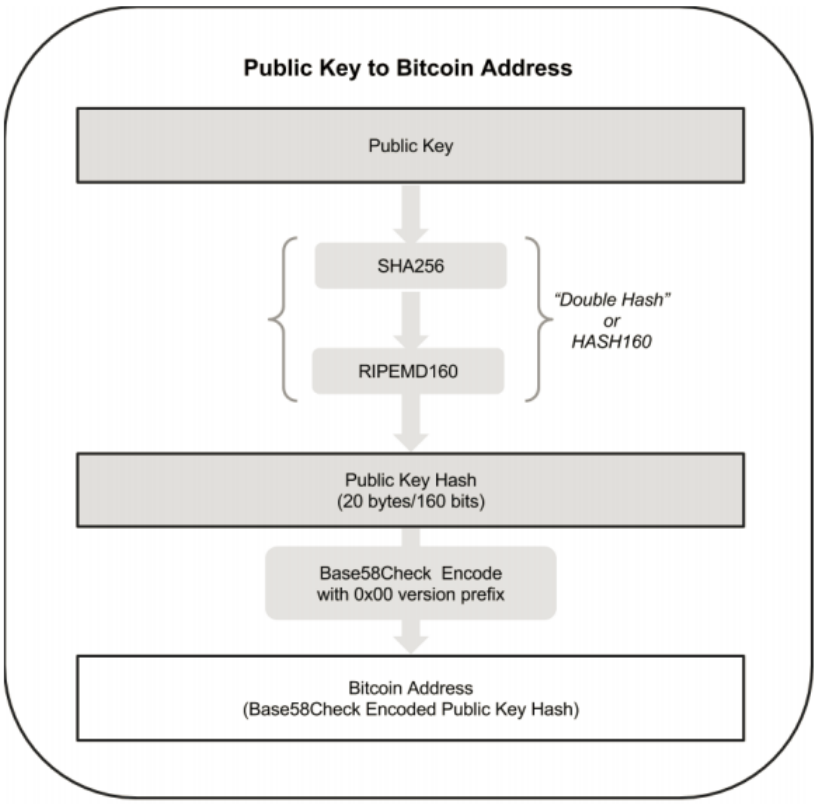

Figure 3.1: A way to create a Bitcoin address from the public key, the figure is copied from Antonopoulos [2].

practice for a payee to generate a new one for every transaction.

Transactions. A Bitcoin transaction indicates a bitcoin movement from source address(es) to destination address(es). A transaction can have one or multiple input and output addresses. We denote a transaction by t. A transaction t_i is specified by a list of input transactions $\{I_j\}_i$ (an exception is a *generation transaction* which does not include a transaction input, more on this later) and a list of output transactions $\{O_j\}_i$. In this context the index j denote the j^{th} input/output of a transaction and the index i denote the i^{th} transaction.

Transactions in Bitcoin are identified by a hash value. Each transaction t_i constituted by the list of input transaction $\{I_j\}_i$ and output transactions $\{O_j\}_i$ is hashed using SHA-256 and the hash will serve as a unique *transaction ID* when published in a block.

Input. Each input I_j of a transaction t_i is a reference to an output from a previous transaction pt_i, which consists of a transaction hash $H(pt_i)$ and an index ind_i into that transaction output, and a script $scriptSig_j$ [9, 16]. An important requirement is that each transaction input I_j has to match a previous transaction output so that the tuple $\big(H(pt_i)$, ind_i, $scriptSig_j\big)$ references pt_i [47]. Note that a previous transaction for a transaction t_i is denoted pt_i and not t_{i-1} because it is not necessarily the first preceding transaction that is referenced.

Output. Each output O_j of a transaction indicates the total amount of bitcoins to be transferred to each output address adr_j. Each output O_j contains an integer value d_j that represents a quantity of Bitcoin currency denoted in *Satoshi* where 1 bitcoin equals 10^8 Satoshi [9]. The total number of bitcoins outputted by a transaction, $\sum_j d_j$, can not exceed the total value of the referenced outputs from previous transactions in $\{I_j\}$ [47]. The output also contains a specification of who is authorized to spend that output called $scriptPubKey_j$. The scripts $scriptSig_j$ and $scriptPubKey_j$ must execute successfully to successfully redeem a previous transaction [9, 16].

The Bitcoin system uses a scripting system for transactions whereas the scripting language include support for cryptographic operations like hashing data and verifying signatures [38]. For pay-to-pub-key-hash transactions the script *scriptPubKey* specifies the hash of a public key and a signature validation routine. This ”locks” the bitcoins to the owner of the private key belonging to that public key. The script *scriptSig* specifies a public key and a signature that ”unlocks” the bitcoins so that they can be spent. The script verifies that the provided public key in *scriptSig* hashes to the hash in *scriptPubKey* and check the signature against

the public key from *scriptSig* (which had to be signed with the private key belonging to the public key) [3, 9]. Recall that control of the private key of an address provides ownership of the bitcoins sent to that address. The scripts are illustrated in Figure 3.2.

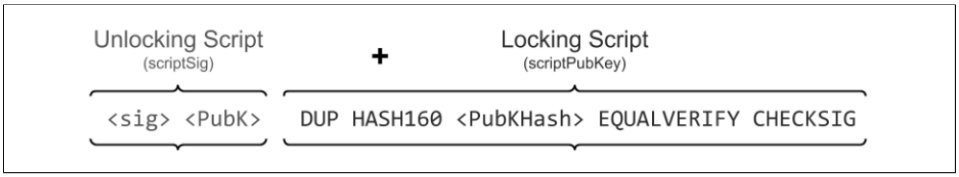

Figure 3.2: The scripts for a pay-to-pub-key-hash transaction, the figure is copied from Antonopoulos [2] and edited.

Figure 3.3 shows an example of a pay-to-pub-key-hash transaction with one input I_1 and one output O_1 (or simply I and O). Let us assume Alice wants to send bitcoins to Bob. Here 50 bitcoins (5 000 000 000 Satoshi) is imported from a previous transaction to Alice by the hash of that transaction (030b593...). In *scriptSig* is a signature σ on the claiming transaction under sk_A and the public key pk_A where Alice unlocks this transaction that was locked to her. These 50 bitcoins are sent and locked to Bob by *scriptPubKey* which contains a hash of Bob's public key pk_B and a signature validation routine.

```
Input:
 Previous tx: 030b5937d9f4aaa1a3133b...
 Index: 0
 scriptSig: 0dcd253cdf8ea11cdc710e5e92af7647...

Output:
 Value: 5000000000
 scriptPubKey: OP_DUP OP_HASH160
 a45f2757f94fd2337ebf7ddd018c11a21fb6c283
 OP_EQUALVERIFY OP_CHECKSIG
```

Figure 3.3: An example of a transaction with one input and one output, the figure is copied from Miers et al. [36].

If Bob at a later time wants to redeem the 50 bitcoins he recieved from Alice that is locked to him, he unlocks it with a transaction where *scriptSig* contains a signature from his private key sk_B, and his public key pk_B. In *scriptPubKey* he presents the hash of the public key belonging the user he wants to send the

bitcoins to and a signature validation routine. The index is 0 which means that Bob references 0 as an input in this transaction where he wants to spend the bitcoins he received from Alice. Note that an address can be further processed after hashing the public key (Figure 3.1) so the hash of the public key we see here is simply the hash of the public key, not the complete address.

There are no such thing as stored balance in Bitcoin. The amount of bitcoins a user receives is stored as an *unspent transaction output* and the only thing that exists are unspent transaction outputs locked to specific users [2]. If the unspent transaction output is greater than the desired value of a transaction it still must be consumed in its entity. This means if you have 10 bitcoins unspent and want to pay someone 5 bitcoins you have to produce two outputs; one paying 5 bitcoins to the desired receiver and one paying 5 bitcoins in change to yourself. It is common practice in the Bitcoin system to return bitcoins to yourself.

Transactions are only valid if they satisfy the constraint that the sum of the values of all transaction outputs is less than or equal to the sum of the values of all inputs (in addition to the requirements that each transaction input matches a previous transaction output and that the scripts execute successfully). Since the total amount of previous transaction outputs must be spent in a transaction it is difficult with transactions with exactly one single input address and one single output address.

Inputs

Previous output (index)	Amount	From address	Type	ScriptSig
e631567f352f....:1	3.02887912	1CGVyAgAx9gg1va5pGNVJtF6gdKpPUVTSf	Address	304402201700305a3d79a[....]2b985b15daa0ab9c50cd61449ca037dc9f0
c284ec14325f....:0	3.04042789	1GY84QPLfM9d4KqTjTbbHsb9BX9FF1kYQx	Address	3045022100e724004f2d3[....]91d95b56ad29f817f3e3259daffbd72f2a98
0fbec1d29b8e....:0	2.99934316	1CGVyAgAx9gg1va5pGNVJtF6gdKpPUVTSf	Address	304402200f6e9b4281cb0[....]2b985b15daa0ab9c50cd61449ca037dc9f0
232715b3c51a....:1	3.00515088	17ALqzZFPbSqXz9aQhzgK6ts9htZfV8Mwu	Address	3044022073114954 78c1d[....]8d46656bf7613d47dd4e6a5b062d9fb6a34

Outputs

Index	Amount	To address	Type	ScriptPubKey
0	0.51682435	1LUHXNTsHPUGVJJeefPdb2rpdxtWoHrcKv	Address	OP_DUP OP_HASH160 d5936a017660c48be2adaa9a77153eccfdb8b0b8 OP_EQUALVERIFY OP_CHECKSIG
1	11.5569767	1HzAb4E1kZH4pDKoxML4KXBLPPyUootw4s	Address	OP_DUP OP_HASH160 ba51b9aee7595c72a2cbc1d4e3e90e356f77804 OP_EQUALVERIFY OP_CHECKSIG

Figure 3.4: An example of a Bitcoin transaction with four input addresses and two output addresses, the figure is copied from Herrera-Joancomartí [29].

Figure 3.4 shows an example of a transaction with several inputs and outputs. As just mentioned, users can collect change of a payment in one of his own addresses, often reffered to as a *shadow address* [29]. This address belongs to the same user that performs the payment. Bitcoins not accounted for will serve as

transaction fees (more on this later). Transactions are often compared to double-entry book-keeping where one page is the inputs of a transaction and the other is the outputs. A bitcoin can be consider a page in the ledger rather than a physical coin which has no story of where it has been.

Signatures. A chain of ownership is created as outputs from one transaction can be used as inputs in new transactions and proof of ownership is therefore required in each transaction. To transfer bitcoins the sender digitally signs a hash of a previous transaction and of the public key belonging to the receiver (as for the example with Alice and Bob). This is done with the sender's private key to prove he is the real owner. The receiver can verify this signature to verify the chain of ownership since he knows the sender's public key. Figure 3.5 illustrates this process. Bitcoin can at its core be considered as a chain of signatures that reflects a coin's path through the Bitcoin system. To verify the validity of a bitcoin a user can check the validity of each signature in the chain.

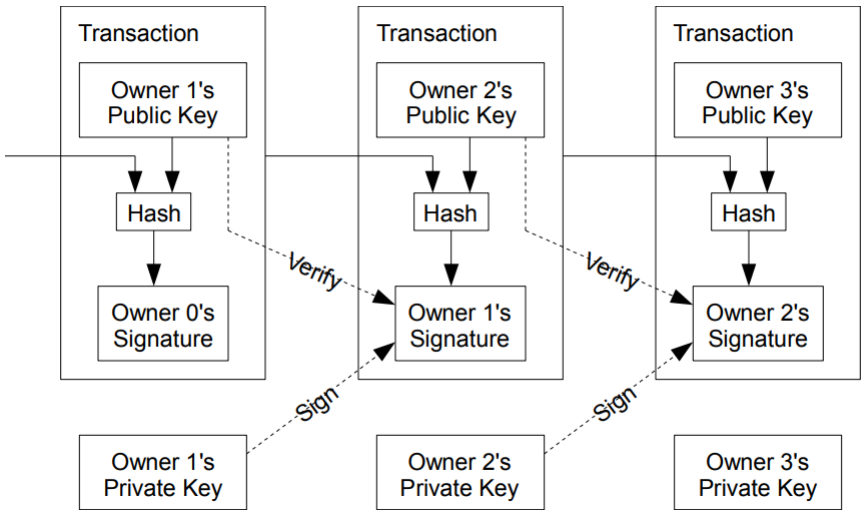

Figure 3.5: How transactions are hashed, signed and verified in the Bitcoin system, the figure is copied from Nakamoto [37].

Lets us say that Alice is owner 1 and Bob is owner 2 in Figure 3.5. Alice creates the transaction in the middle and she needs to prove she has the bitcoins she claims to have. To do so she signs a hash of the transaction she got the bitcoins from (the transaction to the left) and a hash of Bob's public key pk_B with her private key sk_A. We call the signature σ_A. Bob will verify that Alice is the real owner of the

bitcoins by validating σ_A, which implies checking the correspondence between σ_A and pk_A (that corresponds to sk_A). Note that it is crucial that private keys that are used to create signatures are kept secret at all times, otherwise someone else could spend your bitcoins. In addition to validate that the digital signature is correct Bob should also validate that the bitcoins in the input addresses are not previously spent before accepting a payment. The last validation prevents double-spending.

3.2 Blockchain and Mining

To solve the double-spending attack without a trusted party all transactions must be publicly announced. They are published in a global, permanent transaction log, denoted *pubLog*, which is not modifiable except from adding new transactions. Note that this means that payments in the Bitcoin system are non-reversible. We call this public ledger, which contains all bitcoin transactions performed since the system started to operated, the *blockchain*. The blockchain is stored in different nodes of the Bitcoin network which makes Bitcoin a completely distributed system. Any Bitcoin user can maintain a copy of the blockchain. Transactions are collected in *blocks* and the process of adding new blocks to the blockchain is called *mining*. In this chapter we consider the blockchain and the process of mining in more detail.

The Blockchain. When Alice wants to send a transaction she broadcasts it on the Bitcoin network. The transaction has to be verified and included in a block before it becomes a part of the blockchain. Any party can attempt to add blocks to the blockchain (to *mine*) by collecting a set of valid pending transactions and compiling them into a block (more on validity of transactions in the next section).

A block is a data structure that mainly contains a set of transactions in the Bitcoin system. An example of a block is shown in Figure 3.6. This block includes four transactions and we see that the output amount does not exceed the input amount for any transaction. The bitcoins not accounted for serve as fees as we can see in the third transaction. The first transaction is a so-called *generation transaction* (more on this shortly).

In Figure 3.6 we can see that each block has its own hash and contains the hash of the previous block. A system for participants to agree on a single history of the order in which the transactions were received is needed. We say that *global consensus* on the content of the blockchain is required. A part of the solution originally proposed by Nakamoto [37] is a so-called timestamp server which takes the hash of a block of items to be timestamped. Actually transactions are hashed

Block 125552

Hash: 00000000000000001e8d6829a8a21adc5d38d0a473b144b6765798e61f98bd1d
Previous block: 00000000000008a3a41b85b8b29ad444def299fee21793cd8b9e567eab02cd81
Time: 2011-05-21 17:26:31
Difficulty: 244 112.487774
Transactions: 4
Total BTC: 84.52
Size: 1.496 kilobytes
Merkle root: 2b12fcf1b09288fcaff797d71e950e71ae42b91e8bdb2304758dfcffc2b620e3
Nonce: 2504433986

Transactions

Transaction	Fee	Size (kB)	From (amount)	To (amount)
51d37bdd87...	0	0.135	Generation: 50 + 0.01 total fees	15nNvBTUdMaiZ6d3GWCeXFu2MagXL3XM1q: 50.01
60c25dda8d...	0	0.259	1HuppjXz7dPrt2a67LqacDW5T4VanFrpqC: 29.5	1B8vkT58i8KUPVJvvyQfrbc8Wjwu3vEarQ: 0.5 1BQbxzgRSLEsmv1JNc8MG76wdUgMwbsaww: 29
01f314cdd8...	0.01	0.617	1NdzSE6sHubscXJrv7jJn2gd4fL9L3ai6E: 0.03 1Jjv9m5VrRUE7VoktCsj18KUSqkqchhbum: 0.02 1HsYJJPqTn34DEjMnTb3VfKckX7ZcWPibm: 4.82	175FNxcLc1YrTwwG6TcsywcsHYdVqyhbwC: 0.01 1MucNMRJmcqVQeqE7v4dqogpNbhyxqq8R6: 4.85
b519286a10...	0	0.404	12DCoCVvDCkQShZ5RTh9bysgCkmkRMNQbT: 0.14 13CJwnnXJPwkzY4Xnaoqf8dnyNBwrHG9fe: 0.01	1Mos7p8fqJKBcYNRG1TdT5hBRxdMP6YHPy: 0.15

Figure 3.6: An example of a Bitcoin block, the figure is copied from Herrera-Joancomartí [29].

using a Merkle tree where only the root is included in the block's hash, but this will only be noted here and not explored any further. The hash is widely published and the timestamp proves that the data must have existed at the time in order to get into the hash. A chain is formed by each timestamped block including the previous timestamped block in its hash. This is illustrated in Figure 3.7.

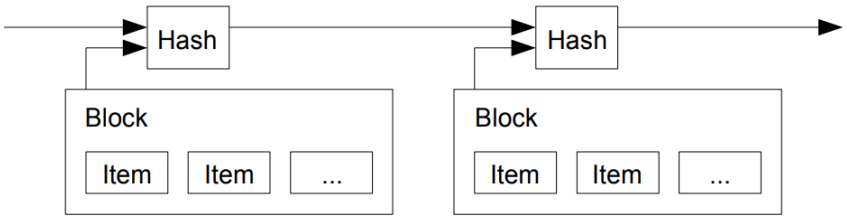

Figure 3.7: How a chain is formed by including the hash of the previous timestamped block in a new timestamped block, the figure is copied from Nakamoto [37].

The process of mining. Mining is the process of adding new blocks to the blockchain. To decide which block that will be added the mining process uses a proof-of-work system to implement a distributed timestamp server. The mining process starts by *miners* (which simply are the ones mining) collecting pending

transactions into a block. Which block will be considered the next in the chain depends on the proof-of-work, which basically is a challenging computational puzzle. The validity of the solution can be verified easily even though the solution itself is hard to find.

The puzzle is to find a block whose hash begins with a number of zero bits. The miners are searching for a block bc such that $\mathrm{SHA256}\big(\mathrm{SHA256}(bc)\big) = 0^l||\{0,1\}^{256-l}$ [9, 36]. A periodic network vote selects the value l to adjust the difficulty to ensure that a block is created every tenth minute on average [3, 9]. The average work for the puzzle is exponential in the number of zero bits required and the difficulty increases if blocks are generated too fast. Note that mining consumes a great amount of energy and computing resources.

Each block has a *nonce* (see Figure 3.6) which is an arbitrary number that can be used only once. The standard strategy to solve the puzzle is brute force by trying random nonces and hashing the block until the desired value is obtained [9, 29]. The first announced valid block containing a solution to the puzzle is considered correct. This block then becomes the top block of the blockchain, miners discard their work on that block and move on to the next block. If all transactions in the block validate and a valid proof of work links the block to the chain thus far, we say that the block is valid. A block is accepted by using that block's hash in the next block. To ensure blockchain integrity a block is not fully trusted until it has a certain number of confirmations, typically six, meaning that there are six blocks on top of that block in the blockchain [24].

The blockchain can have several branches. The consensus blockchain is the longest version at any given time, which means the one with the greatest proof-of-work effort invested in it. Typically this is the branch with the most blocks, but note that the longest chain is defined as the one with the greatest excepted difficulty to produce. This is because mining difficulty can vary. Thereby the Bitcoin protocol depends on the assumption that an adversary can not assess more then 50 % of the computational power of the Bitcoin network (it can be noted that this number is questioned [21]). Two chains of equal length can occur if two valid solutions are found approximately at the same time and miners can then choose either fork [3]. Transactions on the discarded branch will be collected into blocks on the prevailing branch eventually.

Mining implies hard work and miners in the Bitcoin system are rewarded. This happens by two separate incentives. The first one is *transaction fees*. This is the net difference in value between all input and all output transactions in a block. The

second is newly generated bitcoins. Every new block includes a *generation transaction* without any input address and an output address belonging to the miner who created the block (see the first transaction in Figure 3.6). This transaction is also known as a *coinbase transaction* [3, 9]. Miners receive all new currency initially and this is the only allowed mechanism for creation of money in the Bitcoin system. Each block contained 50 new bitcoins for the first four years of operation of the network, then decreased to 25 bitcoins and will half approximately every four years until roughly 2140.

3.3 Communication Network

Transactions and blocks are data that are generated in a distributed way, both addressed by a hash of the data. To transmit such information over the Internet the Bitcoin system uses a distributed peer-to peer (P2P) network consisting of nodes. In this section we will present an overview of the network.

The Bitcoin network consists of nodes where a Bitcoin node is a collection of functions. The four functions are network routing, blockchain database, wallet services and mining [2]. The nodes both provide and consume services at the same time. The nodes can support different functionality and there is no hierarchy within the network. When we say ”the Bitcoin network” we talk about the collection of nodes running the Bitcoin peer-to-peer protocol. Note there are also other protocols for example for mining [2], but this is out of scope for this thesis.

The distributed network is decentralized and created by Bitcoin users in a dynamic way where nodes of the Bitcoin peer-to-peer network are computers that run the Bitcoin node software [20]. The computers that participate in the network are peers to each other, but each peer is not connected to all others. The steps to run the network are as follows stated by Nakamoto [37]:

1. New transactions are broadcast to all nodes.
2. Each node collects new transactions into a block.
3. Each node works on finding a difficult proof-of-work for its block.
4. When a node finds a proof-of-work, it broadcasts the block to all nodes.
5. Nodes accept the block only if all transactions in it are valid and not already spent.

6. Nodes express their acceptance of the block by working on creating the next block in the chain, using the hash of the accepted block as the previous hash.

Let us look more closely at the first point about transactions being broadcast. A user sends a transaction directly to his connected peers when creating a transaction and these peers assess whether it is valid. We recall that the fee for a transaction is set by the each user when constructing the transaction and miners can also exclude transactions with insufficient fees or prioritize transaction with the highest fees. Other reasons for a transaction to be ignored by peers are that [30]:

- the transaction has already been relayed recently (repeated);
- the transaction is already in the main block chain (old);
- the transactions attempts to claim an output already claimed by a previous transactions (double-spend);
- the input signature(s) can not be verified, e.g. attempt to spend some other user's coins (bad signature);
- one or more of the outputs claimed by the inputs can not be found (orphan).

If the transaction is considered valid the connected peers relay it to their peers and the transaction gets propagated through the rest of the network until it reaches a miner who collects it in a block. Both new transaction broadcasts and block broadcasts do not necessarily need to reach all nodes. Transactions will get into a block as long as they reach several nodes. A node will request a block when it receives the next and realizes that one was missing.

3.4 Summary

Before we move on to the next chapter where we consider Bitcoin anonymity, we summarize the basic operation of Bitcoin [34]. First at least one public-private key pair (pk_U, sk_U) is generated for each user U. The public key, or hashing and processing the public key, serve as an address adr_U for the user to receive bitcoins. When a user wants to spend his bitcoins he creates a transaction t which is broadcast to his peers. The transaction should prove that the user has the bitcoins he claims to have by a signature σ_U. The user's peers in turn broadcast the transaction to their peers if the transaction is valid and eventually the transaction reaches

a miner. The miner collects all pending transactions in a block and works on solving a computational puzzle until he hits the target hash. A generation transaction with the miners address as receiver is included in the block as a reward. When the miner solves the puzzle he broadcasts it to his peers who again broadcast it to their peers and all other miners discard their work on that block. The miner gets the newly generated bitcoins, which is the only allowed mechanism for generating new coins in the Bitcoin system, and all the fees for the included transactions as a reward. The block is a part of the blockchain once this block is referenced as a previous block.

Chapter 4

Anonymity in the Bitcoin System

In this chapter we consider anonymity aspects in the Bitcoin system. We recall from Chapter 3 that all Bitcoin transactions are public, that the sender and receiver of bitcoins are identified through public keys and that transactions can have multiple inputs and outputs. These are features of particular interest in this chapter.

Anonymity can be defined in terms on unlinkability and we will start this chapter by doing so for Bitcoin. The first section gives a definition in terms of *address unlinkability* and briefly consider anonymity measures. In the next section we describe address linking in Bitcoin by presenting a heuristic, show how linking can be performed and briefly consider related work on this.

4.1 Defining Anonymity

Anonymity for Bitcoin will in this thesis be defined in terms of unlinkability and base this section mainly on the work of Androulaki et al. [1]. They introduce a notion called *activity unlinkability*. This refers to the fact that an adversary should not be able to link two different addresses or transactions related to a user of his choice. They are referred to as *address unlinkability* and *transaction unlinkability* respectively. Our focus will be on address unlinkability. If two Bitcoin addresses can be linked to the same user by an adversary, he can also link all transactions that these addresses participate in [1, 10, 39].

We will now define an adversary $\mathcal{A}$ against the anonymity of Bitcoin. We assume $\mathcal{A}$ is motivated to acquire information about the addresses related to all or a subset of Bitcoin users. He has access to the public log of Bitcoin, *pubLog*, and

is also a part of the Bitcoin system himself. Further we assume $\mathcal{A}$ is computationally bounded and cannot construct ill-formed Bitcoin blocks, forge signatures, double-spend transactions that are confirmed etc. We consider the following game `B-Anonymity` for defining Bitcoin anonymity:

`B-Anonymity`. The game `B-Anonymity` goes as follows:

- The adversary $\mathcal{A}$ specifies a transaction generator τ that generates transactions in such a way that it is not obvious that $\mathcal{A}$ wins the game.
- The simulator $\mathcal{S}im$ outputs $pubLog$.
- The adversary $\mathcal{A}$ chooses an address adr_0 among the addresses that appear in the public log of Bitcoin, $adr_0 \in pubLog$, and sends it to $\mathcal{S}im$.
- The simulator $\mathcal{S}im$ outputs an address adr_1 such that $adr_1 \neq adr_0$ if possible.
  - If $adr_1 \neq adr_0$ then adr_1 is so that $\mathcal{S}im$ randomly chooses a bit $b \in \{0, 1\}$ and if $b = 0$, $\mathcal{S}im$ chooses an address $adr_1 \in pubLog$ such that adr_0, adr_1 belong to the same user, otherwise $\mathcal{S}im$ randomly chooses adr_1 such that adr_0, adr_1 are owned by different users.
- The adversary $\mathcal{A}$ outputs b' as a guess on whether the two addresses belong to the same user or not and wins the game if $b = b'$.

The game can be illustrated as in Figure 4.1:

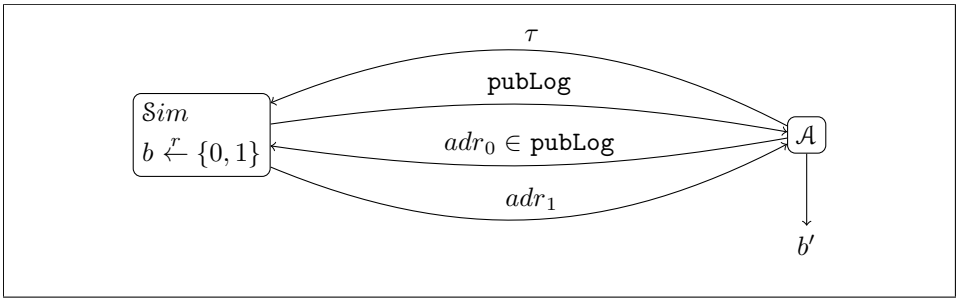

Figure 4.1: An illustration of the game `B-Anonymity` whereby we want to ensure that an adversary can not link addresses to a user.

We have two ways of defining the advantage of $\mathcal{A}$. First we can assess the advantage of $\mathcal{A}$ winning the game in relation to an adversary $\mathcal{A}'$ which plays the same game as $\mathcal{A}$ but does not have access to $pubLog$. We assume that they both

have gathered the same a-priori knowledge κ about whether these addresses belong to the same user or not. We can define the advantage of $\mathcal{A}$ the following way:

$$Adv_{\mathcal{A}} = \left|\Pr[b' \leftarrow \mathcal{A}(pubLog, \kappa) : b = b'] - \Pr[b' \leftarrow \mathcal{A}'(\kappa) : b = b']\right| \tag{4.1}$$

because this equation tells us how much information one can obtain from *pubLog*. We want the probability that $\mathcal{A}$ wins and $\mathcal{A}'$ wins to be only negligibly different as this must mean that *pubLog* does not reveal much information.

Second we can assess the advantage of $\mathcal{A}$ winning the game in relation to an adversary who plays the same game as $\mathcal{A}$ but responds to all game challenges with random guesses. An adversary who responds with random guesses will guess the correct b with probability ½. We can define the advantage of $\mathcal{A}$ the following way:

$$Adv_{\mathcal{A}} = \left|\Pr[b = b'] - \frac{1}{2}\right| \tag{4.2}$$

and based on our definition of address unlinkability and advantage of the adversary we can more formally say something of what we require of anonymity for the Bitcoin system in this thesis.

Definition 4. The Bitcoin system satisfies the requirement of anonymity if every adversary $\mathcal{A}$ has negligible advantage in `B-Anonymity`.

By negligible advantage we mean that the public log should not reveal much information to the adversary for him to base his guess on (4.1) and that the adversary can not do better than to simply guess randomly whether the addresses belong to the same user or not (4.2). Let us say that ϵ is the advantage of $\mathcal{A}$. Since the transaction log is public it is not possible to make the definition hold for any negligible ϵ. So instead of proving that the Bitcoin system does satisfy the requirement of anonymity, we will actually look at how we can attack the definition by showing how an adversary has considerable advantage regarding address linking. Before we do so we briefly mention some anonymity measures in Bitcoin.

It is claimed that public keys are the mechanism Bitcoin use to ensure anonymity and is based on the fact that users can create any number of anonymous Bitcoin addresses from these keys to use in their transactions [20, 29]. Nakamato stated in the Bitcoin white paper that privacy can be maintained as long as the public keys are kept anonymous and further; *the public can see that someone is sending an amount to someone else, but without information linking the transactions to anyone* [37].

An adversary who wants to de-anonymize users will attempt to construct a one-to-many mapping between users and public keys and associate information external to the system with the users. To prevent such an attack Bitcoin stores the mapping of a user to his of her public keys on that user's node only and also by allowing each user to generate many public keys. To better protect their identity a user can also avoid revealing any identifying information in connection with these keys, repeatedly send varying fractions of their own bitcoins to themselves by using multiple newly generated public keys and/or use a trusted third-party mixer or laundry [44].

4.2 Attacking Anonymity

By allowing users to use different addresses (derived from their public keys) in every transaction the Bitcoin system partially addresses the anonymity issue. Nevertheless, the underlying Internet infrastructure that is non-anonymous together with the fact that all Bitcoin transactions are publicly available in the blockchain, has proven to be a threat to anonymity [3]. The original Bitcoin paper actually cautioned that some linking of addresses is unavoidable [37].

There are several papers published on Bitcoin anonymity and one of the categories are papers that perform blockchain analysis [29]. Recall that the blockchain is public and includes all transactions ever performed by the Bitcoin system. Data obtained from the blockchain can be used to derive information from users and properties like usage patterns. Such data includes which Bitcoin addresses money comes to, where it goes, when, how many bitcoins is held within the addresses etc.

To attack the anonymity we defined in Section 4.1 the goal is use data obtained from the blockchain (*pubLog*) to cluster the addresses in the blockchain that belongs to the same user as a user can generate any number of keys (and addresses from them). Another step can then be taken to link address clusters to identities in the real-world. As mentioned we will start by describing a heuristic for linking addresses before we show how the linking can be performed. We close the chapter by giving a short overview of related work. We mainly base the presentation on the work of Reid and Harrigan [44].

4.2.1 Heuristic for Linking Addresses

We recall that transactions can have several input addresses from Section 3.1. Multi-input transactions occur when a user wishes to perform a payment and the

amount exceeds the value of each of the available bitcoins in this users wallet [1]. By assuming that all input addresses of a transaction belong to the same user, addresses of the same user can be clustered. More precise, if two (or more) public keys are used as inputs in the same transaction we say that the same user controls them. This heuristic can be defined as follows [34]: *If two (or more) addresses are inputs to the same transaction, they are controlled by the same user; i.e. for any transaction t, all pk* $\in I(t)$ *are controlled by the same user.*

The sender of a transaction must know the private key belonging to each public key used as an input in order to spend the bitcoins (described in Section 3.1). It is therefore unlikely that the collection of public keys are controlled by multiple parties which if so would have to reveal their private keys to each other. This would imply that others could use their bitcoins. This makes the heuristic quite safe. The heuristic also extends beyond the inputs to a single transaction. If one transaction has input addresses adr_0 and adr_1 and another has adr_1 and adr_2, we can conclude that adr_0, adr_1 and adr_2 belong to the same user [34].

Androulaki et al. [1] showed that the advantage of an adversary who wants to say something about whether addresses belong to the same user or not is considerable given our heuristic. They also argue that the heuristic can not be easily evaded in any future implementation of Bitcoin without compromising the basic operation of Bitcoin. Next we show that the advantage of $\mathcal{A}$ in `B-Anonymity` is considerable by showing how addresses can be linked using the heuristic regarding multi-input transactions. By considerable advantage we mean, if we look at (4.1) and (4.2), that the public log reveals information to the adversary and that he can do better than simply guessing randomly whether addresses belong to the same user or not.

4.2.2 Linking Addresses

First, a note on public keys and addresses is found convenient. We defined address unlinkability and described that we want to link *addresses*, but here we show how we can link *public keys*. Recall that the address is derived from the public key. One can not derive the public key from the address, but we will look at *clustering* and if you cluster public keys you also cluster the addresses derived from them. This means that when we show how to link public keys we can also link the addresses derived from these keys.

Reid and Harrigan [44] define two distinct network structures for the Bitcoin system, the *transaction network* and the *user network*. The basis for them is

provided by three features of the Bitcoin system; the public availability of Bitcoin transactions, the use of public keys and the fact that a transaction can have multiple inputs and multiple outputs. Both networks represents the flow of bitcoins over time, but the transaction network describe the flow between *transactions* and the user network between *users*.

The goal is to obtain the (pseudonymous) user network from the transaction network. The transaction network describe how bitcoins move from transaction to transaction and if we can cluster all transactions belonging to a user in this network, we obtain the user network that describe how bitcoins move from user to user. To do so we will move from a transaction network through an incomplete user network to a complete user network. An overview of this process is shown in Figure 4.2 and details of the figure will follow.

The networks can be described in graph terminology. For the transaction network each vertex represents a transaction t. Each directed edge represents a flow of bitcoins from an output of one transaction to an input of another. Further we can consider an *incomplete* user network in the sense that each (diamond) vertex represents a single public key pk rather than a user U. Each directed edge represents an input-output pair of a single transaction where the public key of the input belongs to the user corresponding to the source and the public key of the output belongs to the user corresponding to the target. Each subset of vertices whose corresponding public keys belongs to a single user needs to be contracted in order to perfect the network. For the user network each (circular) vertex represents a user.

To completely perfect the network using a dataset of transactions alone is impossible because a user can create a new public-private key pair for every transaction. However, for transactions with multiple inputs, public keys can be related to one user by the heuristic in Section 4.2.1. Some address linking is therefore possible and subsets of vertices in the incomplete network can be contracted. We will describe how addresses can be linked by considering these networks and go into the details of Figure 4.2. We follow the work of Reid and Harrigan [44]. They downloaded the public transaction ledger and clustered Bitcoin addresses into users by using the described heuristic. We will use the notation introduced in Section 3.1.

Figure 4.3 shows a sub-network of the transaction network. Public keys will be used in our explanation related to the figure but are not shown. Note that each directed edge also includes a value in bitcoins and a timestamp that is not shown in Figure 4.2. The timestamp tells when this transaction was added to the blockchain. Let us look at transaction t_1. This transaction has one input $(I_1)_1$ and

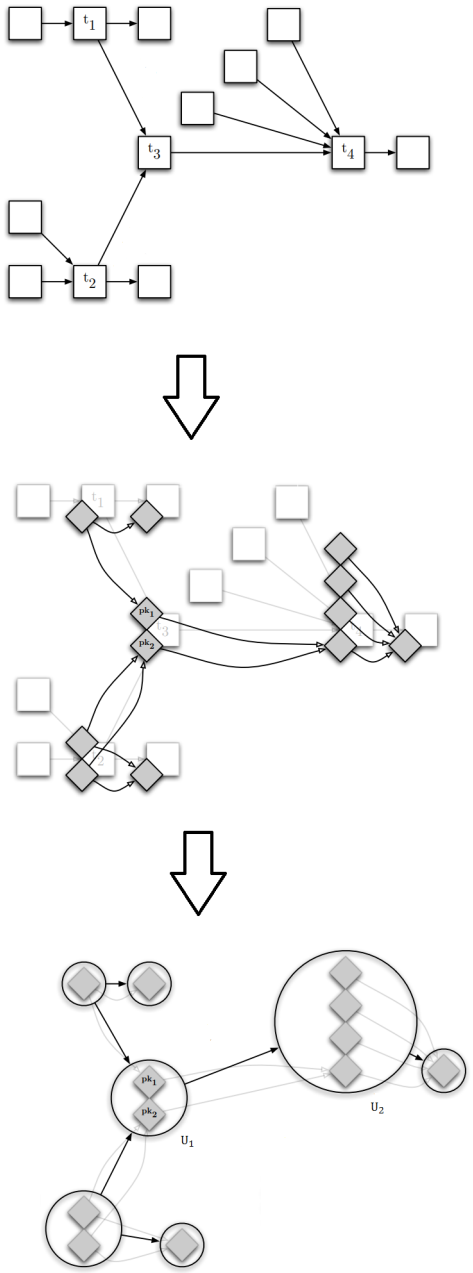

Figure 4.2: An overview of how we move from the transaction network (through an incomplete user network) to obtain the user network. Regular vertices illustrates transactions, diamond vertices public keys and the circles illustrates users. The figure components are copied and edited from Reid and Harrigan [44].

two outputs $(O_1)_1$ and $(O_2)_1$. In $(O_2)_1$ 1.2 bitcoins was sent to a user identified by the public key pk_1. One of the two outputs of t_2, $(O_2)_2$, sent 0.12 bitcoins to a user identified by pk_2. The inputs of t_3, $(I_j)_3$, are connected to $(O_2)_1$ and $(O_2)_2$. The only output of t_3 is to t_4 which is a transaction with sixteen inputs (only four in Figure 4.3) and only one output.

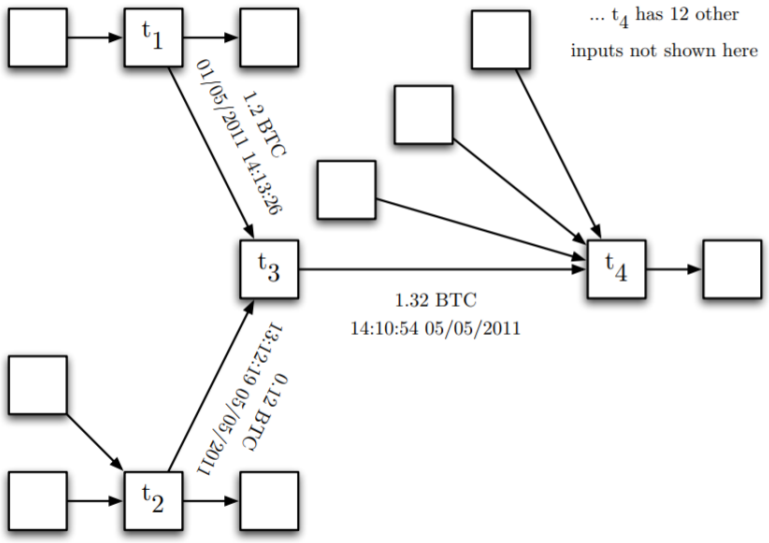

Figure 4.3: An example of a sub-network for the transaction network, the figure is copied from Reid and Harrigan [44].

Figure 4.4 is overlaid onto the network in Figure 4.3 and shows an example of a sub-network from an incomplete user network. Here each diamond vertex represents a public key and each directed edge between diamond vertices represents a flow of bitcoins from one public key to another. For each transaction, as can be seen behind the diamond vertices, each diamond vertex represents the public keys corresponding to the inputs of that transaction. We remember that an output from t_1 was sent to a user with public key pk_1 and an output from t_2 to a user with public key pk_2. These are the inputs to transaction t_3 and this transaction is therefore marked with pk_1 and pk_2.

Figure 4.5 is overlaid onto the network in Figure 4.4 and shows an example of a sub-network from the user network. Here each circular vertex represents a user and not a public key (therefore not incomplete) and each directed edge between circular vertices represents a flow of bitcoins from one user to another. By our heuristic, an adversary $\mathcal{A}$ who wants to link addresses and more specifically say

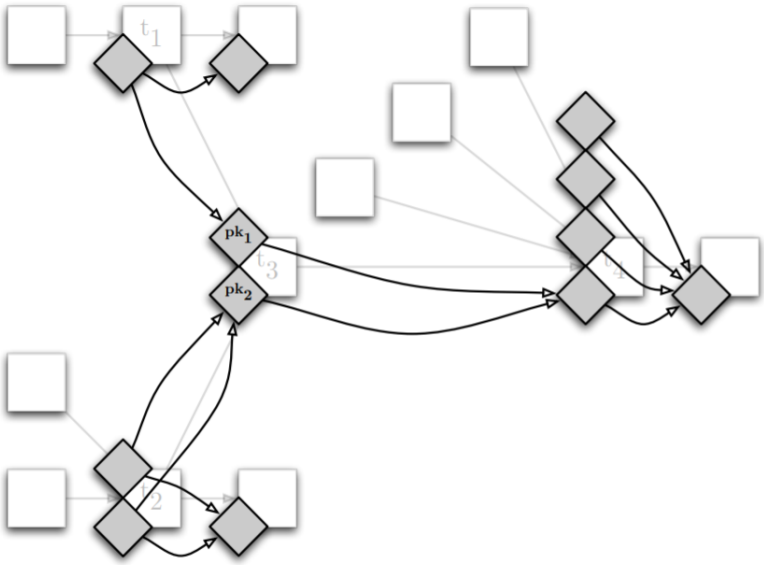

Figure 4.4: An example of a sub-network from the incomplete user network overlaid to the network from Figure 4.3, the figure is copied from Reid and Harrigan [44].

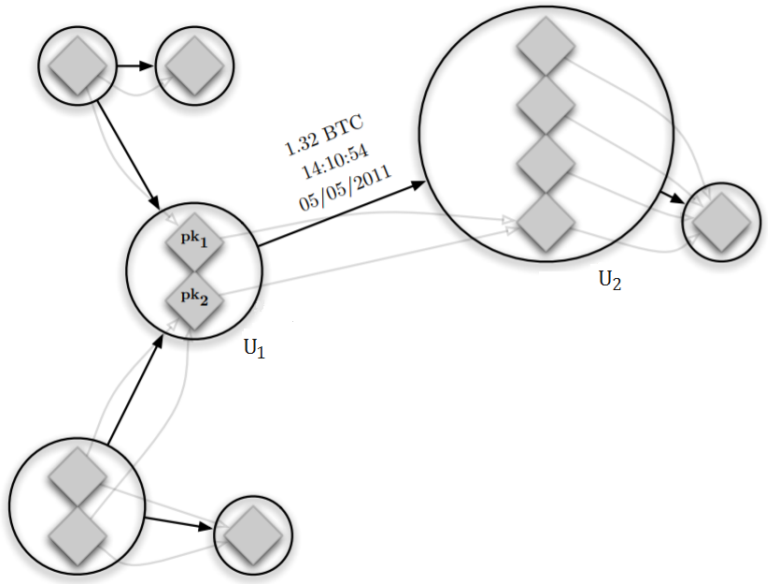

Figure 4.5: An example of sub-network from the user network overlaid to the network from Figure 4.4, the figure is edited from that of the original paper [44].

something about whether addresses belong to the same user or not, can see that an address derived from pk_1, say adr_1, and an address derived from pk_2, say adr_2, must belong to a single user U_1. This is because they correspond to an input pair of a single transaction. The same reasoning holds for user U_2. The sixteen (four on the figure) inputs to transaction t_4 results in a contraction of sixteen public keys into a single circular vertex, a single user U_2.

By this technique addresses of the same user can be linked together by the adversary. Further he can take another step and link address clusters to identities in the real-world. External information on Bitcoin addresses can be obtained from different resources on the Internet to help the clustering process and to identify the users behind these clusters. Reid and Harrigan, for example, used forum posts [44]. In such a way the anonymity of identities and organizations can be attacked. Reid and Harrigan [44] concluded that it is possible to associate many public keys with each other by using an appropriate network representation and showed how this could be used to track a thief in a case study. They note that it is not difficult for others to replicate their work.

4.2.3 Related Work

Reid and Harrigan published the first research article on Bitcoin anonymity [44]. To analyze the anonymity offered by the Bitcoin system others have also exploited data obtained from the blockchain and in this section we briefly present some of these articles. The presentation is based on a selection of previous work [1, 29, 30, 34, 46].

Ron and Shamir [46] performed a similar analysis to that of Reid and Harrigan [44]. They performed an analysis of typical behavior of entities on the Bitcoin network from the blockchain data. They did not have as their goal to de-anonymize users but answered questions like how users acquire bitcoins and how they spend them, how they move bitcoins between their various accounts in order to better protect their privacy etc.

Androulaki et al. [1] took another step into clustering addresses by taking into account the same idea of Reid and Harrigan [44] with multi-input addresses. They also added another heuristic regarding shadow addresses. In the case where a Bitcoin transaction has two output addresses where one them is an address that has appeared in the public ledger before and the other one never has, they assumed that this new address is a shadow address. This address can therefore be clustered with the input addresses. Note that they assumed that most transaction do not have more than two output addresses. They also evaluated privacy by behavior-

based clustering techniques and by simulating the use of Bitcoin in a university, and found that the profiles of almost 40% of users can be recovered.

Meiklejohn et al. [34] built on past efforts to cluster addresses. They performed an active analysis in contrast to that of Androulaki et al. [1], Reid and Harrigan [44] and Ron and Shamir [46] by interacting with parties on the Bitcoin network to create a list of known Bitcoin addresses. This was done by performing payments from their own Bitcoin addresses to known services like mining pools, exchanges sites and so on. To obtain user identification of other addresses they also browsed the Internet. Meiklejoin et al. also used the heuristic that all input addresses belong to the same user and a heuristic for identifying shadow addresses. Their approach was similar to that of Androulaki et al. [1] but was not limited to two outputs only in the transactions. The authors concluded that it is possible to trace the movements for large Bitcoin transactions and that the anonymity provided by the Bitcoin network is not sufficient.

Note that Bitcoin is a dynamic system. This implies that some of the hypotheses assumed for some blockchain analysis may not completely hold. Above it is assumed that multi-input transactions only can be created by one user which was the case at the time the first paper on Bitcoin anonymity was published. Bitcoin did not provide support for different users to participate in a single transaction. Later proposals where multiple users can join a single transaction, like *CoinJoin*, break this assumption but again have practical complications and may provide smaller anonymity sets [10, 29, 33].

As we can see, Bitcoin has limitations regarding anonymity. There are three main classes of proposals to improve anonymity [9]; *i*) through the peer-to-peer network, *ii*) through mix services that mixes the coins of multiple users or *iii*) with *altcoins* with integrated unlinkability. For the last class of proposals we consider the possibility to exchange bitcoins for some alternate currency (altcoins) and later change the altcoins back for bitcoins. Figure 4.6 shows a comparative evaluation of some of the proposals for improving the anonymity of Bitcoin. We refer the reader to the work of Conti et al. [16] for a greater comparison of techniques for improving anonymity and of altcoins where both distinct features and properties, advantages and disadvantages for each is presented.

One of the altcoin proposals is that of Miers et al. [36]; a new system designed to cryptograhically amplify the anonymity guarantees in Bitcoin called *Zerocoin*. Zerocoin propose to add decentralized anonymous electronic cash to Bitcoin and in the next chapter we examine Zerocoin. In Chapter 6 we make a comparison of the

Proposal	*Class*	*Security*			*Deploy.*	
		Internal Unlinkability	Theft Resistance	DoS Resistance	Bitcoin-compatible	# Transactions
CoinJoin [79]	P2P	○	●	○	●	1
Shuffle Net [35]	P2P	○	●	○	●	1
Fair Exchange [13]	P2P	○	●	○	●	4
CoinShuffle [104]	P2P	●	●	◐	●	1
Mixcoin [26]	distr.	◐	◐	●	●	2
Blindcoin [118]	distr.	●	◐	●	●	4
CryptoNote [119]	altcoin	●	●	●	○	0
Zerocoin [81]	altcoin	●	●	●	○	2
Zerocash [16]	altcoin	●	●	●	○	0

Figure 4.6: A comparative evaluation of proposals for improving anonymity of Bitcoin, the figure is copied from Bonneau et al. [9].

anonymity discussed for Bitcoin in this chapter and the anonymity considerations to be presented for Zerocoin.

Chapter 5

Improving Bitcoin Anonymity: Zerocoin

Zerocoin is a cryptographic extension to Bitcoin proposed by Miers et al. [36]. It is a decentralized electronic cash system that breaks the link between individual Bitcoin transactions using standard cryptographic assumptions. Zerocoin does not introduce any new trusted parties or change the security model of Bitcoin. It extends Bitcoin with a new form of anonymous electronic cash and is essentially a decentralized laundry that allows individual Bitcoin clients to generate new coins (if they have sufficient bitcoins to do so). The idea is that you can take real bitcoins, transfer them to zerocoins and pull back bitcoins with the exact same property.

Zerocoin is a construction that relies on digital commitments and zero-knowledge proofs. Each coin is a commitment. Zerocoin does not rely on digital signatures to validate coins but authenticates coins by proving in zero-knowledge that a coin belongs to a public list of valid coins maintained on the blockchain. Note that Zerocoin could be integrated into any cryptocurrency although originally proposed for Bitcoin.

The Zerocoin construction is referred to as a decentralized electronic cash scheme as it does not require a central coin issuer. In this chapter we first describe the algorithms that make up such a scheme. Next we define the security required of a decentralized electronic cash scheme and we do so by two game-based definitions. We further describe a concrete instantiation and look at how the described construction integrates into Bitcoin. We close the chapter by giving proofs

of security for the construction in terms of the two games defined.

This chapter is based on the original Zerocoin paper [36] and one of the authors master thesis [35] to varying degrees. The description of the algorithms of the scheme and the concrete construction is pretty straightforward with some modifications where it is found convenient (especially regarding notation) while security analysis and the construction of the signature of knowledge that Zerocoin takes use of have greater modifications and are written in more detail than where it is introduced [35, 36].

5.1 Algorithms

In this section we define the algorithms that make up a decentralized electronic cash scheme and define the completeness required. In the following we let $\mathbf{C}$ denote the set of allowable coin values and λ represent an adjustable security parameter. A decentralized electronic cash scheme is a tuple of algorithms (`Setup`, `Mint`, `Spend`, `Verify`) defined as follows:

System setup. The algorithm `Setup` generates a set of public parameters and a description of the set $\mathbf{C}$:

- Input:
  - a security parameter λ
- Output:
  - public parameters *params*
  - a description of the set $\mathbf{C}$

Minting coins. The algorithm `Mint` generates new coins:

- Input:
  - parameters *params*
- Output:
  - a coin $c \in \mathbf{C}$
  - a trapdoor tr for c

Spending coins. The algorithm `Spend` allows coins to be spent and generates coin spend transactions:

- Input:
  - parameters $params$
  - a coin c
  - the trapdoor tr for c
  - some transaction string $T \in \{0,1\}^*$ that stores transaction-specific information
  - an arbitrary set of coins C
- Output:
  - a proof π and a serial number sn if $c \in C \subseteq \mathbf{C}$
  - otherwise $\perp$

Verifying transactions. The algorithm `Verify` checks the validity of a transaction:

- Input:
  - parameters $params$
  - a proof π
  - a serial number sn
  - transaction information T
  - a set of coins C
- Output:
  - 1 if $C \subseteq \mathbf{C}$ and (π, sn, T) is valid
  - 0 otherwise

It is worth noting that the serial number sn that is released during the spending of a coin is a unique value. This is designed to prevent any user from spending the same coin twice. We call (π, sn) a coin spend transaction or simply a spend transaction. The transaction string T is an arbitrary string intended to store transaction-specific information, as the identity of the receiver. Before we formalize the completeness of our decentralized electronic cash scheme we summarize the algorithms:

- $params \leftarrow \texttt{Setup}(\lambda)$
- $(c,\, tr) \leftarrow \texttt{Mint}(params)$
- $(\pi,\, sn) \leftarrow \texttt{Spend}(params,\, c,\, tr,\, T,\, C)$
- $\{0,1\} \leftarrow \texttt{Verify}(params,\, \pi,\, sn,\, T,\, C)$

We want every decentralized electronic cash scheme to satisfy the following requirement for completeness:

Definition 5. Let $params \leftarrow \texttt{Setup}(\lambda)$, $(c, tr) \leftarrow \texttt{Mint}(params)$ and $C \subseteq \mathbf{C}$ be any valid set of coins where $|C|$ is at most N. Let $(\pi,\, sn) \leftarrow \texttt{Spend}(params,\, c,\, tr,\, T,\, C)$. A decentralized electronic cash scheme $\Pi = (\texttt{Setup}, \texttt{Mint}, \texttt{Spend}, \texttt{Verify})$ is *complete* if $\forall$ C, T and randoms coins used in the above algorithms the following equality holds with all but negligible probability:

$$\texttt{Verify}\big(params, \pi, sn, T, C \cup \{c\}\big) = 1.$$

If `Setup`, `Mint` and `Spend` are executed correctly, we want the transaction to be verified in `Verify`. For completeness we want a prover to be able to convince a verifier that a true statement is indeed true. This means that an honest user producing a spend transaction by following the scheme should be able to convince others that he did so and his transaction should be accepted with all but negligible probability.

5.2 Security

The security of a decentralized electronic cash system will be defined by two games. We first describe the game `Z-Anonymity` and then the game `Balance`. The first game ensures that an adversary can not link a given coin spend transaction $(\pi,\, sn)$ to the coin associated with it and the latter that an adversary can not spend more coins than he mints, i.e. that a party can not forge coins. Proof of security is provided in Section 5.4.

5.2.1 Anonymity

We continue to define anonymity in terms of unlinkability as we did for Bitcoin. Recall that in the algorithm `Spend` a coin is spent and the output is a (coin) spend transaction $(\pi,\, sn)$. We want to ensure that an adversary can not link a given spend

transaction to the coin associated with it, which implies linking the address used to mint the original zerocoin to the address used to redeem the zerocoin. We let the adversary provide many of the coins used in generating the spend transaction and still want the linking to be unfeasible. We consider the following game for defining anonymity in Zerocoin:

`Z-Anonymity`. The game $\texttt{Z-Anonymity}\big(\Pi, \mathcal{A}, \lambda\big)$ goes as follows:

- The simulator $\mathcal{S}im$ outputs $params \leftarrow \texttt{Setup}(\lambda)$.
- For $i \in \{0, 1\}$ the simulator runs $(c_i, tr_i) \leftarrow \texttt{Mint}(params)$, outputs (c_0, c_1) and stores the associated trapdoors (tr_0, tr_1).
- The adversary $\mathcal{A}$ outputs $(C, T) \leftarrow \mathcal{A}(params, c_0, c_1)$ using any strategy he wishes.
- The simulator $\mathcal{S}im$ samples $b \xleftarrow{r} \{0, 1\}$ and outputs $(\pi, sn) \leftarrow \texttt{Spend}\big(params, c_b, tr_b, T, C \cup \{c_0, c_1\}\big)$.
- The adversary $\mathcal{A}$ outputs $b' \leftarrow \mathcal{A}(\pi, sn)$ as a guess of which coin was spent.

The game can be illustrated as in Figure 5.1:

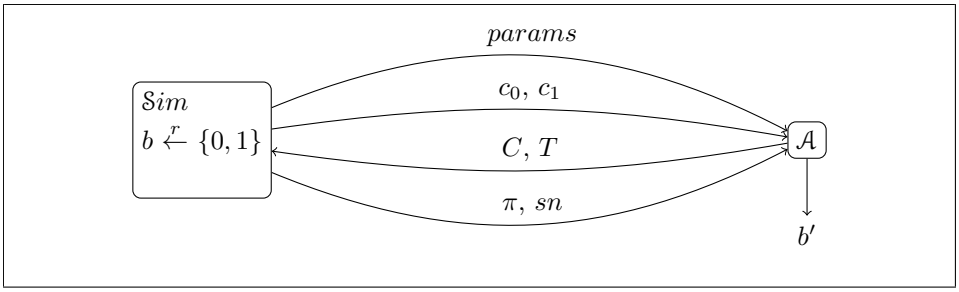

Figure 5.1: An illustration of the game `Z-Anonymity` whereby we want to ensure that an adversary can not link a given spend transaction the coin associated with it.

The adversary $\mathcal{A}$ wins if $b = b'$. We want the information provided to $\mathcal{A}$ not to reveal anything about the bit b so that $\mathcal{A}$ can not do any better than an adversary who simply guesses b randomly, i.e. who guesses the correct b with probability ½. We define the advantage of $\mathcal{A}$ as follows:

$$Adv_{\mathcal{A}} = \left| \Pr[b = b'] - \frac{1}{2} \right| \tag{5.1}$$

and close this section by giving a more formal definition of the what we require of anonymity for a decentralized electronic cash system like Zerocoin:

Definition 6. The requirement for anonymity is satisfied for a decentralized electronic cash scheme Π = (`Setup`, `Mint`, `Spend`, `Verify`) if every adversary $\mathcal{A}$ has negligible advantage in `Z-Anonymity`.

5.2.2 Balance

As mentioned we would like to ensure that an adversary can not spend more coins than he mints. We give the adversary a collection of valid coins and an honest simulator that he can use to spend any of these coins. If $\mathcal{A}$ produces M coins and $M+1$ valid spends, no transaction will duplicate a serial number or modify a transaction produced by the simulator. We consider the following game for defining balance in Zerocoin where the adversary wants to spend more coins than he mints:

`Balance`. The game `Balance`$(\Pi, \mathcal{A}, N, \lambda)$ goes as follows:

- The simulator $\mathcal{S}im$ outputs $params \leftarrow$ `Setup`(λ).
- For $i = 1$ to N the simulator runs $(c_i, tr_i) \leftarrow$ `Mint`$(params)$ and outputs $\{c_1, ..., c_N\}$
- The adversary $\mathcal{A}$ sends a query (c_i, C_i, T_i) to $\mathcal{S}im$
- The simulator $\mathcal{S}im$ does the following:
  - If $c_i \notin \{c_1, ..., c_N\}$ output $\perp$.
  - Otherwise output $(\pi_i, sn_i) \leftarrow$ `Spend`$(params_i, c_i, tr_i, T_i, C_i)$ and record (sn_i, T_i) in the set S.
- The adversary $\mathcal{A}$ outputs $\left(c_1^*, ..., c_M^*, V_1^*, ..., V_M^*, V_{M+1}^*\right)$.

Note that the raised star denotes that something is produced by $\mathcal{A}$. Also note that the number of queries is not necessarily the same as the number of honest minted coins. The adversary can send as many queries as he likes and he can also query coins not among the honest minted coins (results in $\perp$). However, we use the index i for both honest minted coins by $\mathcal{S}im$ and queries from $\mathcal{A}$ to make it clear that $\mathcal{S}im$ only spends coins he has minted (and has the trapdoor to). The simulator records (sn_i, T_i) for the i^{th} spend, meaning that a coin with serial number sn_i is spent in a transaction with transaction string T_i. This coin that the

simulator spends is one of his minted coins c_i for $i = 1$ to N, but the i^{th} spend is not necessarily for the i^{th} minted coin (but only minted coins are spent by the simulator).

When $\mathcal{S}im$ outputs sn_i as response to the i^{th} query this is a serial number sn_i for one of his minted coins c_i, but the first minted coin is not necessarily spent in the first query (or at all). The game can be illustrated as in Figure 5.2:

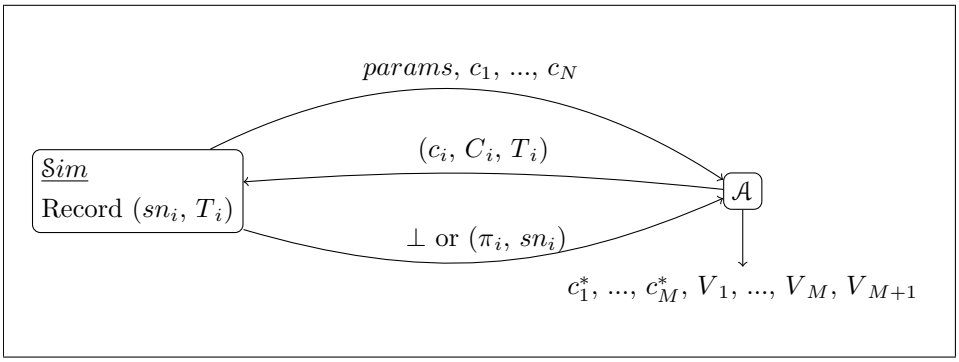

Figure 5.2: An illustration of the game `Balance` whereby we want to ensure that an adversary can not spend more coins than he mints.

We give the adversary access to coins and spend transactions produced by an honest party. The adversary can alter valid coins, for example by modifying the corresponding transaction information string T. He can also specify arbitrary input for the value C. We say that $\mathcal{A}$ wins if $\forall\ V^* = (\pi^*_{,}\ sn^*_{,}\ T^*_{,}\ C^*) \in \{V^*_1, ..., V^*_M, V^*_{M+1}\}$ the following holds:

- `Verify`$(params, \pi^*_{,}\ sn^*_{,}\ T^*_{,}\ C^*) = 1$,
- $C^* \subseteq \{c_1, ..., c_N, c^*_1, ..., c^*_M\}$,
- $(sn^*_{,}\ T^*) \notin S$ and
- sn^* appears only in one tuple from $\{V^*_1, ..., V^*_M, V^*_{M+1}\}$.

This means that we require that the spends $\mathcal{A}$ produces must be verified (recall that this means that $C^* \subseteq \mathbf{C}$ and $(\pi^*_{,}\ sn^*_{,}\ T^*)$ is valid), the coins in C^* must be among the the coins produced in `Mint` by an honest party or by $\mathcal{A}$ through the simulator $\mathcal{S}im$, the pair $(sn^*_{,}\ T^*)$ for a given transaction should not be recorded in the set of records S by $\mathcal{S}im$ and a serial number sn^* should only appear once among the tuples of valid spend transactions.

If these conditions hold the adversary $\mathcal{A}$ can produce more spends than minted coins and he wins the game. We note the event that $\mathcal{A}$ wins by E. We define the advantage of $\mathcal{A}$ as the probability that $\mathcal{A}$ wins `Balance`;

$$Adv_{\mathcal{A}} = \Pr[E] \tag{5.2}$$

and close this section by giving a more formal definition of the what we require of balance for a decentralized electronic cash system like Zerocoin:

Definition 7. The requirement of balance is satisfied for a decentralized electronic cash scheme $\Pi = (\texttt{Setup}, \texttt{Mint}, \texttt{Spend}, \texttt{Verify})$ having at most N coins if $\forall$ N every adversary $\mathcal{A}$ has negligible advantage in `Balance`.

5.3 Instantiation

In this section we describe an instantiation of the decentralized electronic cash scheme from Section 5.1. This is the Zerocoin construction. Before we describe the construction we need to describe an accumulator is made use of. Further we describe the construction of a signature of knowledge before we close the chapter by describing how the Zerocoin construction integrates with Bitcoin.

5.3.1 Accumulation

The construction to be described uses an *accumulator*. An accumulator scheme is an algorithm that allows one to combine a set of values into one short accumulator. There exists a short witness w that a given value was incorporated into the accumulator and at the same time it is infeasible to find a witness for a value that was not accumulated. The accumulator is used to prove that you know a coin c in the set of all minted zerocoins C without revealing which coin it is. This accumulator is based on the Strong RSA assumption. For details we refer the reader to Benaloh and de Mare [7] who introduced the concept and Barić and Pfitzmann [4] and Camenisch and Lysyanskaya [12] who improved it. The following algorithms define the accumulator used in the construction:

Accumulation setup. The algorithm `AccumSetup` generates a set of public parameters:

- Input:

- a security parameter λ

- Output:
  - $n = p'q'$ where p' and q' are primes sampled with polynomial dependence on the security parameter
  - a seed value $u \in QR_n$ [4, 36] where $u \neq 1$ and QR_n denote the quadratic residues modulo n

Accumulating. The algorithm `Accumulate` computes an accumulator Λ of prime numbers:

- Input:
  - parameters *params* (n, u)
  - a set of prime numbers $C = \{c_1, ..., c_N\} \mid c \neq p', q' \wedge\ c \in \left[X, X^2\right)$ for some fixed X

- Output:
  - an accumulator Λ computed as $u^{c_1 c_2 \cdots c_N} \pmod{n}$

We will say a bit more about the range $\left[X, X^2\right)$ in the next section.

Generating a witness. The algorithm `GenWitness` generates a witness w that a given value c is accumulated:

- Input:
  - parameters *params* (n, u)
  - a set of prime numbers C as in `Accumulate`
  - a value $c \in C$

- Output:
  - a witness w which is the accumulation of all the values in C besides c; $w =$ `Accumulate`$\big(params,\ C \setminus \{c\}\big)$

Verifying. The algorithm `AccVerify` checks whether a presented witness for a given value in fact is a valid witness:

- Input:

  - parameters $params$ $(n,\ u)$
  - an accumulator Λ
  - an element c
  - a witness w

- Output:
  - 1 if $\Lambda = \Lambda' \equiv w^c \pmod{n}$, c is prime and $c \in \left[X, X^2\right)$
  - 0 otherwise

The accumulator can be summarized as follows:

- $params \leftarrow \texttt{AccumSetup}(\lambda)$
- $\Lambda \leftarrow \texttt{Accumulate}(params,\ C)$
- $w \leftarrow \texttt{GenWitness}(params,\ c,\ C)$
- $\{0, 1\} \leftarrow \texttt{AccVerify}(params,\ \Lambda,\ c,\ w)$

A trusted setup process is assumed for generating the accumulator parameters [36, 38]. Note that the accumulator trapdoor (p', q') can be deleted immediately after the parameters are generated as they are not used after `AccumSetup`. Also note that the accumulator can be incrementally updated as pointed out by Miers et al. [36], but we will not discuss this any further.

If the Strong RSA assumption holds the accumulator satisfies a strong collision-resistance property [12]. This ensures that no adversary running in feasible time can produce a pair $(c,\ w)$ where $c \notin C$ that is verified in `AccVerify`.

An efficient zero-knowledge proof of knowledge that a (committed) value is accumulated is presented by Camenisch and Lysyanskaya [12]. This is converted into a non-interactive zero-knowledge proof of knowledge (NIZKPoK) by the Fiat-Shamir transform. The resulting proof is referred to the following way:

$$\text{NIZKPoK}\{(c, w) : \texttt{AccVerify}\big((n, u), \Lambda, c, w\big) = 1\}.$$

Recall that this notation denotes a proof of knowledge of the elements c and w that satisfies $\texttt{AccVerify}\big((n,\ u),\ \Lambda,\ c,\ w\big) = 1$, i.e. a proof of knowledge of a witness w for a value c that verifies in `AccVerify`. This means a valid witness w that c is accumulated. The proof is one of two parts in a signature of knowledge used in the construction to be described.

5.3.2 Construction

In this section we describe an instantiation of a decentralized electronic cash scheme which is the Zerocoin construction. We assume the hardness of the Strong RSA problem and the Discrete logarithm problem. We also assume the existence of a zero-knowledge proof system. The Zerocoin system can be proven secure under these assumptions, something we do in Section 5.4. The construction is based on the algorithms defined in Section 5.1:

- $params \leftarrow \texttt{Setup}(\lambda)$
- $(c, tr) \leftarrow \texttt{Mint}(params)$
- $(\pi, sn) \leftarrow \texttt{Spend}(params, c, tr, T, C)$
- $\{0, 1\} \leftarrow \texttt{Verify}(params, \pi, sn, T, C)$

Next we describe the construction in detail.

System setup. The details of $params \leftarrow \texttt{Setup}(\lambda)$ are as follows:

- Run $\texttt{AccumSetup}(\lambda)$ on input λ to obtain (n, u).
- Generate primes p, q such that $p = 2^j q + 1$ for $j \geq 1$ according to Miers et al. [36].
- Select random generators g, h for $\mathbb{G}$ where $\mathbb{G} \subseteq \mathbb{F}_p^*$.
- Output $params = (n, u, p, q, g, h)$.

Minting coins. The details of $(c, tr) \leftarrow \texttt{Mint}(params)$ are as follows:

- On input $params$ select sn, $r \xleftarrow{r} \mathbb{Z}_q$.
- Compute the commitment $c = commit(sn, r)$ where $c \leftarrow g^{sn} h^r \pmod p$ such that c is prime in $\left[X, X^2\right)$ for a fixed X.
  - If c is not a prime, start `Mint` from the beginning and try again. We want c to be prime as the accumulator accumulates prime numbers in `Accumulate`.
- Set $tr = (sn, r)$.
- Output (c, tr).

Note that the set we commit to is the set of serial numbers randomly chosen from $\mathbb{Z}_q$, the set of randomness is numbers randomly chosen from $\mathbb{Z}_q$ and the set of commitments C will exist in $\mathbb{F}_p^*$. The commitments are Pedersen commitments as described in Section 2.4.1. The range $\big[X, X^2\big)$ is defined as so to guarantee that the product of two commitments is outside the interval. This is described in an article based on Zerocoin [19].

Spending coins. The details of $(\pi,\ sn) \leftarrow$ `Spend`$(params,\ c,\ tr,\ T,\ C)$ are as follows:

- (Output $\perp$ if $c \notin C$.)
- Compute $\Lambda \leftarrow$ `Accumulate`$\big((n,\ u),\ C\big)$.
- Compute $w \leftarrow$ `GenWitness`$\big((n,\ u),\ c,\ C\big)$.
- Output $(\pi,\ sn)$.

The proof π compromises the following signature of knowledge on the transaction string T:

$$\pi = \text{ZKSoK}[T]\{(c, w, r) : \texttt{AccVerify}\big((n, u), \Lambda, c, w\big) = 1 \wedge c = g^{sn}h^r\}.$$

This signature of knowledge proves knowledge of a coin c, a witness w and a random number r such that *i*) the coin c was minted somewhere in the past by showing that the committed value is in an accumulator and that *ii*) the coin c can be opened to a serial number sn with commitment randomness r. We examine the signature of knowledge further after the last part of the construction:

Verifying transactions. The details of $\{0, 1\} \leftarrow$ `Verify`$(params,\ \pi,\ sn,\ T,\ C)$ are as follows:

- Compute $\Lambda \leftarrow$ `Accumulate`$\big((n,\ u),\ C\big)$, recall that Λ is computed as $u^{c_1 c_2 \cdots c_N} \pmod n$ when $C = \{c_1, ..., c_N\}$.
- Verify that π is the signature of knowledge on T by using the public values (recall from `AccVerify` that you compute Λ' $\equiv w^c \pmod n$ and check if Λ' $= \Lambda$.
- Output 1 if the proof verifies successfully, output 0 otherwise.

We want to note as Miers et al. [36] that we specify security in terms of a single adjustable security parameter λ, but in reality there are three distinct security choices for the Zerocoin construction; the size of the RSA modulus used in the accumulator, the size of the group used in the coin commitments and the security of the zero-knowledge proofs. We will not explore this any further and move on to the signature of knowledge which is a central part of the Zerocoin construction.

5.3.3 Construction of the Signature of Knowledge

In this section we consider the signature of knowledge π which proves that the spending party can open one of the commitments in C to a presented serial number sn. The signature is composed of two zero-knowledge proofs; first that a committed value c is accumulated and second that c is a commitment to sn (which means that $c = g^{sn}h^r$). Recall that

$$\pi = \text{ZKSoK}[T]\{(c, w, r) : \texttt{AccVerify}\big((n, u), \Lambda, c, w\big) = 1 \wedge c = g^{sn}h^r\}.$$

The first part (that we remember from Section 5.3.1), a proof of knowledge of c and w, is given by Camenisch and Lysyanskaya [12] and the second part, a proof of knowledge of c and r, is briefly presented by Miers [35] and Miers et al. [36]. We refer the reader to Camenisch and Lysyanskaya for first part part of the proof as we focus on the construction of the second part. Note that this presentation is more detailed than that of where it is briefly introduced [35, 36]. The proof we consider is a double discrete logarithm proof. First we look at an interactive protocol which we later make non-interactive.

Interactive protocol. We consider, given a cyclic group $\mathbb{G} \subseteq \mathbb{F}_p^*$ of order q where $p = 2^j q + 1$ with generator g_1, g_2, a cyclic group $\mathbb{H} \subseteq \mathbb{F}_q^*$ of order ℓ where $q = 2\ell+1$ with generator h_1 and $y_1 = {g_1}^{{h_1}^{x_1}} {g_2}^{x_2}$ and $y_2 = {h_1}^{x_1}$, how to prove knowledge of x_1 and x_2. A prover $\mathcal{P}$ does so by executing the protocol in Figure 5.3 with a verifier $\mathcal{V}$ for $i = 1$ to k where k is a security parameter.

The proof in Figure 5.3 proves knowledge of x_1 and x_2 such that $y_1 = {g_1}^{{h_1}^{x_1}} {g_2}^{x_2}$ and $y_2 = {h_1}^{x_1}$. We recall the commitment $c = g^{sn}h^r$ for Zerocoin. We have chosen to use a general notation and not the Zerocoin notation for convenience, but if we consider a case where $g = g_1$, $h = g_2$, $sn = {h_1}^{x_1}$ and $r = x_2$, we prove knowledge of c and r such that c is a commitment to sn by proving we know sn with commitment randomness r such that $c = g^{sn}h^r$. Next we prove that the proof is complete, has special soundness and is special honest-verifier zero-knowledge.

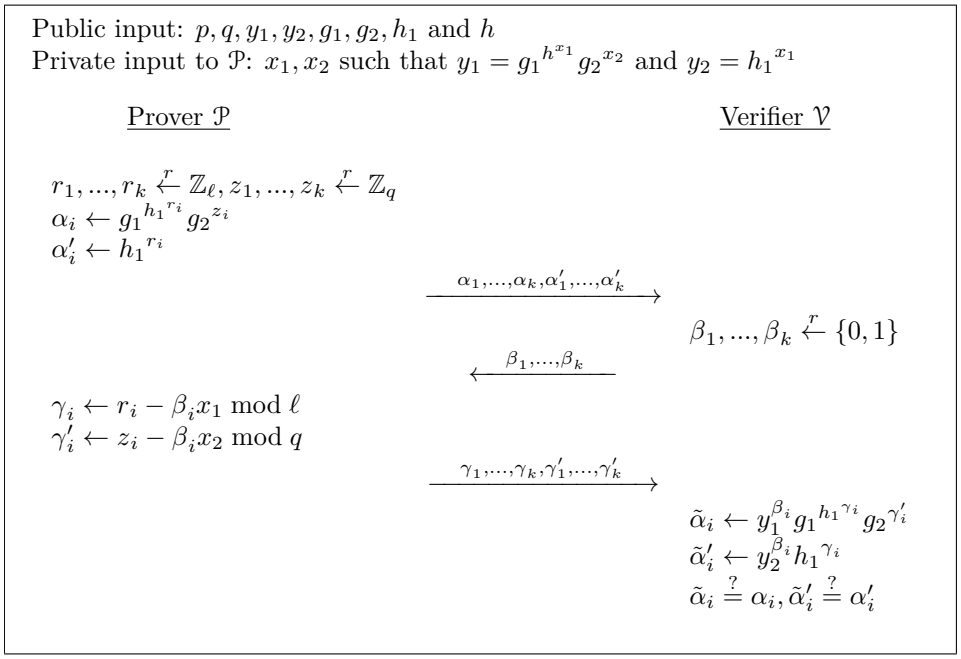

Figure 5.3: An interactive protocol for proving knowledge of a double discrete logarithm.

Theorem 1. *The proof in Figure 5.3 is complete.*

Proof. We show that the proof is complete by showing that the verifier $\mathcal{V}$ will accept on public input (p, q, y_1, y_2, g_1, g_2 and h_1) if the prover $\mathcal{P}$ is honest.

$$
\begin{aligned}
&\beta_i = 0 : \\
&\tilde{\alpha}_i = y_1^{\beta_i} {g_1}^{{h_1}^{\gamma_i}} {g_2}^{\gamma'_i} = y_1^0 {g_1}^{{h_1}^{r_i}} {g_2}^{z_i} = {g_1}^{{h_1}^{r_i}} {g_2}^{z_i} = \alpha_i \\
&\tilde{\alpha}'_i = y_2^{\beta_i} {h_1}^{\gamma_i} = y_2^0 h_1 r_i = {h_1}^{r_i} = \alpha'_i
\end{aligned}
$$

$$
\begin{aligned}
&\beta_i = 1 : \\
&\tilde{\alpha}_i = y_1 {g_1}^{{h_1}^{(r_i - x_1)}} {g_2}^{(z_i - x_2)} = {g_1}^{{h_1}^{x_1}} {g_2}^{x_2} {g_1}^{{h_1}^{(r_i - x_1)}} {g_2}^{(z_i - x_2)} \\
&\quad = {g_1}^{{h_1}^{(x_1 + r_i - x_1)}} {g_2}^{(x_2 + z_i - x_2)} = {g_1}^{{h_1}^{r_i}} {g_2}^{z_i} = \alpha_i \\
&\tilde{\alpha}'_i = y_2^{\beta_i} {h_1}^{\gamma_i} = y_2 {h_1}^{(r_i - x_1)} = {h_1}^{x_1} {h_1}^{(r_i - x)} = {h_1}^{(x_1 - x_1)} {h_1}^{r_i} = {h_1}^{r_i} = \alpha'_i
\end{aligned}
$$

Since $\tilde{\alpha}_i = \alpha_i$ and $\tilde{\alpha}'_i = \alpha'_i$ the proof is complete. □

Theorem 2. *The proof in Figure 5.3 has special soundness.*

Proof. We will show special soundness for the proof by showing that it is possible to find the secret (x_1 and x_2) if we have two accepting conversations between the prover and the verifier from rewinding. We perform the rewinding for the protocol in Figure 5.3 as follows:

1. Run $\mathcal{P}^* \longleftrightarrow \mathcal{V}$ until $\mathcal{P}^*$ sends (α_i, α'_i).
2. Remember the state of $\mathcal{P}^*$ and $\mathcal{V}$.
3. Choose $\mathcal{V}$'s challenge (β_i) randomly.
4. Run $\mathcal{P}^* \longleftrightarrow \mathcal{V}$ until $\mathcal{P}^*$ sends (γ_i, γ'_i).
5. Rewind.
6. Choose a new challenge $(\hat{\beta}_i)$ randomly.
7. Run $\mathcal{P}^* \longleftrightarrow \mathcal{V}$ until $\mathcal{P}^*$ sends $(\hat{\gamma}_i, \hat{\gamma}'_i)$.

This gives us two accepting conversations between $\mathcal{P}^*$ and $\mathcal{V}$:

$$\big((\alpha_i, \alpha'_i), (\beta_i), (\gamma_i, \gamma'_i)\big) \text{ and } \big((\alpha_i, \alpha'_i), (\hat{\beta}_i), (\hat{\gamma}_i, \hat{\gamma}'_i)\big).$$

We assume without loss of generality that for some i we have $\beta_i \neq \hat{\beta}_i$. Since the conversations are accepting we know that the following must hold:

$$\tilde{\alpha}_i = y_1^0 {g_1}^{{h_1}^{\gamma_i}} {g_2}^{\gamma'_i} = y_1^1 {g_1}^{{h_1}^{\hat{\gamma}_i}} {g_2}^{\hat{\gamma}'_i} \implies {g_1}^{{h_1}^{\gamma_i}} {g_2}^{\gamma'_i} = y_1 {g_1}^{{h_1}^{\hat{\gamma}_i}} {g_2}^{\hat{\gamma}'_i}.$$

By manipulation of this expression we see that

$${g_1}^{{h_1}^{\gamma_i}} {g_2}^{\gamma'_i} = y_1 {g_1}^{{h_1}^{\hat{\gamma}_i}} {g_2}^{\hat{\gamma}'_i} \implies y_1 = {g_1}^{{h_1}^{\gamma_i}} {g_2}^{\gamma'_i} ({g_1}^{{h_1}^{\hat{\gamma}_i}} {g_2}^{\hat{\gamma}'_i})^{-1} = {g_1}^{{h_1}^{(\gamma_i - \hat{\gamma}_i)}} {g_2}^{(\gamma'_i - \hat{\gamma}'_i)}.$$

We recall that $y_1 = {g_1}^{{h_1}^{x_1}} {g_2}^{x_2}$ so we have

$${g_1}^{h^{(\gamma_i - \hat{\gamma}_i)}} {g_2}^{(\gamma'_i - \hat{\gamma}'_i)} \implies x_1 \equiv \gamma_i - \hat{\gamma}_i \bmod \ell, x_2 \equiv \gamma'_i - \hat{\gamma}'_i \bmod q.$$

From the rewinding, which gave us the two accepting conversations between the prover and the verifier, we managed to compute the secret input of the prover. This means that the protocol has special soundness (Section 2.3.1) and we have shown what we wanted. □

Theorem 3. *The proof in Figure 5.3 is special honest-verifier zero-knowledge.*

Proof. To show that the proof is special honest-verifier zero-knowledge we need to show that there exists a simulator $\mathcal{S}im$ that on the same public input as in a real conversation produced by a prover $\mathcal{P}$ and verifier $\mathcal{V}$ can produce a conversation that is indistinguishable from that between from $\mathcal{P}$ and $\mathcal{V}$. More specifically, we want to show that $\mathcal{S}im$ on public input $(p, q, y_1, y_2, g_1, g_2, h_1)$ and challenge (β_i) outputs an accepting conversation $\big((\alpha_i, \alpha'_i), (\beta_i), (\gamma_i, \gamma'_i)\big)$ that is indistinguishable from a real conversation $\big((\alpha_i, \alpha'_i), (\beta_i), (\gamma_i, \gamma'_i)\big)$ produced by $\mathcal{P}$ and $\mathcal{V}$ with public input $(p, q, y_1, y_2, g_1, g_2, h_1)$, challenge (β_i) and private input x_1 and x_2.

In the real conversation between $\mathcal{P}$ and $\mathcal{V}$ we replace $\mathcal{V}$ with (β_i). The prover first selects (α_i, α'_i) at random and computes (γ_i, γ'_i). In the simulated conversation the simulator $\mathcal{S}im$ is given (β_i), selects (γ_i, γ'_i) at random and computes (α_i, α'_i). We want to show that one can not determine in which of the two ways an accepting conversation $\big((\alpha_i, \alpha'_i), (\beta_i), (\gamma_i, \gamma'_i)\big)$ was created. This means we want to show that

$$\begin{aligned} &\Pr[\big(\mathcal{P} \longleftarrow (\beta_i)\big) \rightsquigarrow \big((\alpha_i, \alpha'_i), (\beta_i), (\gamma_i, \gamma'_i)\big)] \\ &= \Pr[\big(\mathcal{S}im \longleftarrow (\beta_i)\big) \rightsquigarrow \big((\alpha_i, \alpha'_i), (\beta_i), (\gamma_i, \gamma'_i)\big)]. \end{aligned} \tag{5.3}$$

where the arrow $\rightsquigarrow$ can be read "outputs". We need to show that given $\big((\alpha_i, \alpha'_i), (\beta_i)\big)$ there is only one choice for (γ_i, γ'_i) and given $\big((\beta_i), (\gamma_i, \gamma'_i)\big)$ there is only one choice

for (α_i, α_i'). We start by considering the first of the two probabilities in (5.3) and observe that

$$\begin{aligned}
&\Pr[\big(\mathcal{P} \longleftarrow (\beta_i)\big) \rightsquigarrow \big((\alpha_i, \alpha_i'), (\beta_i), (\gamma_i, \gamma_i')\big)] \\
&= \Pr[\alpha_i = {g_1}^{{h_1}^{r_i}} {g_2}^{z_i} \wedge \alpha_i' = {h_1}^{r_i} | r_i \xleftarrow{r} \mathbb{Z}_\ell, z_i \xleftarrow{r} \mathbb{Z}_q] \\
&= \Pr[{g_2}^{z_i} = \alpha_i {g_1}^{-{h_1}^{r_i}} \wedge {h_1}^{r_i} = \alpha_i' | r_i \xleftarrow{r} \mathbb{Z}_\ell, z_i \xleftarrow{r} \mathbb{Z}_q] \\
&= \Pr[z_i \equiv \log_{g_2} \alpha_i - {h_1}^{r_i} \log_{g_2} g_1 \wedge r_i \equiv \log_{h_1} \alpha_i' | r_i \xleftarrow{r} \mathbb{Z}_\ell, z_i \xleftarrow{r} \mathbb{Z}_q].
\end{aligned} \tag{5.4}$$

First we show that there exists a unique pair (r_i, z_i) such that ${g_1}^{{h_1}^{r_i}} {g_2}^{z_i} = \alpha_i$ and ${h_1}^{r_i} = \alpha_i'$. From $r_i \equiv \log_{h_1} \alpha_i'$ we know that there exists a unique r_i such that $\log_{h_1} \alpha_i' \equiv r_i$. This implies that there exists a unique z_i such that $z_i \equiv \log_{g_2} \alpha_i - {h_1}^{r_i} \log_{g_2} g_1$. We therefore know that there exists one and only one pair (r_i, z_i) such that ${g_1}^{{h_1}^{r_i}} {g_2}^{z_i} = \alpha_i$ and ${h_1}^{r_i} = \alpha_i'$. Next we show that given $\big((\alpha_i, \alpha_i'), (\beta_i)\big)$, where we know that (α_i, α_i') is uniquely specified for (r_i, z_i), there is only one choice for (γ_i, γ_i'). We assume we have two accepting conversations $\big((\alpha_i, \alpha_i'), (\beta_i), (\gamma_i, \gamma_i')\big)$ and $\big((\alpha_i, \alpha_i'), (\beta_i), (\hat{\gamma}_i, \hat{\gamma}_i')\big)$. We know that the following must hold:

$$\begin{aligned}
(*) \quad & \alpha_i = {y_1}^{\beta_i} {g_1}^{{h_1}^{\gamma_i}} {g_2}^{\gamma_i'} = {y_1}^{\beta_i} {g_1}^{{h_1}^{\hat{\gamma}_i}} {g_2}^{\hat{\gamma}_i'} \\
(**) \quad & \alpha_i' = {y_2}^{\beta_i} {h_1}^{\gamma_i} = {y_2}^{\beta_i} {h_1}^{\hat{\gamma}_i}
\end{aligned}$$

where the second equation (**) gives us that $\gamma_i = \hat{\gamma}_i$. It then follows from the first equation (*) that $\gamma_i' = \hat{\gamma}_i'$. This means that there is only one choice for (γ_i, γ_i') given $\big((\alpha_i, \alpha_i'), (\beta_i)\big)$. Due to the injectivity it is sufficient to consider where r_i and z_i is drawn from in (5.4). Since r_i is drawn at random from $\mathbb{Z}_\ell$, z_i is drawn at random from $\mathbb{Z}_q$ and there are k pairs we have

$$\Pr[\big(\mathcal{P} \longleftarrow (\beta_i)\big) \rightsquigarrow \big((\alpha_i, \alpha_i'), (\beta_i), (\gamma_i, \gamma_i')\big)] = \left(\frac{1}{\ell} \cdot \frac{1}{q}\right)^k = \frac{1}{(\ell q)^k} \tag{5.5}$$

for the first of the two probabilities in (5.3). Next we consider the simulated conversation and the following probability:

$$\begin{aligned}
&\Pr[\big(\mathcal{S}im \longleftarrow (\beta_i)\big) \rightsquigarrow \big((\alpha_i, \alpha_i'), (\beta_i), (\gamma_i, \gamma_i')\big)] \\
&= \Pr[\hat{\gamma}_i = \gamma_i \wedge \hat{\gamma}_i' = \gamma_i' | \hat{\gamma}_i \xleftarrow{r} \mathbb{Z}_\ell, \hat{\gamma}_i' \xleftarrow{r} \mathbb{Z}_q].
\end{aligned}$$

Note that we write $\hat{\gamma}_i \xleftarrow{r} \mathbb{Z}_\ell$ since $r_i \xleftarrow{r} \mathbb{Z}_\ell$ and $\gamma_i \leftarrow r_i - \beta_i x_1 \pmod{\ell}$, and $\hat{\gamma}_i' \xleftarrow{r} \mathbb{Z}_q$ since $z_i \xleftarrow{r} \mathbb{Z}_q$ and $\gamma_i' \leftarrow z_i - \beta_i x_2 \pmod{q}$. It is clear from the computation of

(α_i, α_i') given $\big((\beta_i), (\gamma_i, \gamma_i')\big)$ that there is only one choice for (α_i, α_i'), therefore we can say that

$$\begin{aligned} &\Pr[\big(\mathcal{S}im \longleftarrow (\beta_i)\big) \rightsquigarrow \big((\alpha_i, \alpha_i'), (\beta_i), (\gamma_i, \gamma_i')\big)] \\ &= \Pr[\hat{\gamma}_i = \gamma_i \wedge \hat{\gamma}_i' = \gamma_i' | \hat{\gamma}_i \xleftarrow{r} \mathbb{Z}_\ell, \hat{\gamma}_i' \xleftarrow{r} \mathbb{Z}_q] = \left(\frac{1}{\ell} \cdot \frac{1}{q}\right)^k = \frac{1}{(\ell q)^k} \end{aligned} \tag{5.6}$$

and conclude from (5.5) and (5.6) that

$$\begin{aligned} &\Pr[\big(\mathcal{P} \longleftarrow (\beta_i)\big) \rightsquigarrow \big((\alpha_i, \alpha_i'), (\beta_i), (\gamma_i, \gamma_i')\big)] \\ &= \Pr[\big(\mathcal{S}im \longleftarrow (\beta_i)\big) \rightsquigarrow \big((\alpha_i, \alpha_i'), (\beta_i), (\gamma_i, \gamma_i')\big)] \end{aligned}$$

which is what we wanted to show. This shows that our proof is special honest-verifier zero-knowledge and more specifically *perfect* special honest-verifier zero-knowledge since the probability spaces are perfectly indistinguishable (they are the same). □

We have shown that our interactive proof is complete, has special soundness and is special honest-verifier zero-knowledge. This is a Σ-protocol. As we can recall such protocols can be made non-interactive.

Non-interactive protocol. We can take the interactive Σ-protocol that has special soundness and is special honest-verifier zero-knowledge and make the protocol non-interactive by the Fiat-Shamir transform. The resulting proof will be sound and zero-knowledge in the random oracle model (reffering to Section 2.3.3). By including the transaction string T in the hash computed by the prover, the proof will also serve as a signature. We refer to the resulting non-interactive proof as a zero-knowledge signature of knowledge (ZKSoK).

We let $k \leq l$ be two security parameters and $H : \{0,1\}^* \to \{0,1\}^l$ be a collision-resistant hash function. As in the interactive version $y_1 = {g_1}^{{h_1}^{x_1}} {g_2}^{x_2}$ and $y_2 = {h_1}^{x_1}$ is given and we want to prove knowledge of x_1 and x_2. The symbol $||$ denotes the concatenation of two binary strings or of binary representations of integers and group elements and by $\beta[i]$ we mean the i^{th} bit of the string β. To construct the signature of knowledge first generate $2k$ random numbers $r_1, ..., r_k$ and $z_1, ..., z_k$. For $i = 1$ to k compute $\alpha_i \leftarrow {g_1}^{{h_1}^{r_i}} {g_2}^{z_i}$ and $\alpha_i' \leftarrow {h_1}^{r_i}$. Compute

$$\beta \leftarrow H\big(T||y_1||y_2||g_1||g_2||h_1||\alpha_1||...||\alpha_k||\alpha_1'||...||\alpha_k'\big)$$

and (where $\beta[i]$ is either 0 or 1)

$$\gamma_i \leftarrow r_i - \beta[i]x_1,$$
$$\gamma'_i \leftarrow z_i - \beta[i]x_2.$$

The signature of knowledge on the transaction string T is $\big(\beta, (\gamma_i, \gamma'_i)\big)$. To verify the signature compute:

$$\tilde{\alpha}_i \leftarrow {y_1}^{\beta[i]} {g_1}^{{h_1}^{\gamma_i}} {g_2}^{\gamma'_i},$$
$$\tilde{\alpha}'_i \leftarrow {y_2}^{\beta[i]} {h_1}^{\gamma_i}$$

and

$$\beta' \leftarrow H\big(T||y_1||y_2||g_1||g_2||h_1||\tilde{\alpha}_1||...||\tilde{\alpha}_k||\tilde{\alpha}'_1||...||\tilde{\alpha}'_k\big)$$

and check whether $\beta = \beta'$. The protocol below illustrates the non-interactive proof (for $i = 1$ to k):

Public input: p, q, y_1, y_2, g_1, g_2 and h_1
Private input to $\mathcal{P}$: x_1, x_2 such that $y_1 = {g_1}^{{h_1}^{x_1}} {g_2}^{x_2}$ and $y_2 = {h_1}^{x_1}$

Prover $\mathcal{P}$		Verifier $\mathcal{V}$
$r_1, ..., r_k \xleftarrow{r} \mathbb{Z}_\ell, z_1, ..., z_k \xleftarrow{r} \mathbb{Z}_q$		
$\alpha_i \leftarrow {g_1}^{{h_1}^{r_i}} {g_2}^{z_i}$		
$\alpha'_i \leftarrow {h_1}^{r_i}$		
$\beta \leftarrow H\big(T\|\|y_1\|\|y_2\|\|g_1\|\|g_2\|\|h_1\|\|\alpha_1\|\|...\|\|\alpha_k\|\|\alpha'_1\|\|...\|\|\alpha'_k\big)$		
$\gamma_i \leftarrow r_i - \beta[i]x_1 \bmod \ell$		
$\gamma'_i \leftarrow z_i - \beta[i]x_2 \bmod q$		
	$\xrightarrow{\beta,\gamma_1,...,\gamma_k,\gamma'_1,...,\gamma'_k}$	
		$\tilde{\alpha}_i \leftarrow {y_1}^{\beta[i]} {g_1}^{{h_1}^{\gamma_i}} {g_2}^{\gamma'_i}$
		$\tilde{\alpha}'_i \leftarrow {y_2}^{\beta[i]} {h_1}^{\gamma_i}$
		$\beta' \leftarrow H\big(T\|\|y_1\|\|y_2\|\|g_1\|\|g_2\|\|h_1\|\|\tilde{\alpha}_1\|\|...\|\|\tilde{\alpha}_k\|\|\tilde{\alpha}'_1\|\|...\|\|\tilde{\alpha}'_k\big)$
		$\beta \stackrel{?}{=} \beta'$

Figure 5.4: A non-interactive protocol from the interactive in Figure 5.3 for proving knowledge of a double discrete logarithm.

Since we showed that the interactive proof has special soundness (Theorem 2)

and is special honest-verifier zero-knowledge (Theorem 3) we have that the non-interactive proof illustrated in Figure 5.4 will be sound and zero-knowledge in the random oracle model. These are properties we will make use of in our security analysis in Section 5.4.

5.3.4 Integrating into Bitcoin

Before we move on to prove security for the Zerocoin construction we describe how it integrates into Bitcoin. As mentioned Zerocoin is essentially a decentralized laundry where the idea is that you can take bitcoins from your wallet, transfer them to zerocoins (where they get mixed with other user's coins) and pull back bitcoins with the exact same property. You temporarily pool bitcoins together in exchange for this currency called zerocoin. Nobody should be able to link your new coins to the old ones.

Minting coins. Let us assume Alice wants to mint a zerocoin of denomination d. In the described construction individual coins all have the same value, but it is noted by Miers et al. [36] that multiple values can be supported by running distinct Zerocoin instantiations simultaneously. Alice will run $\texttt{Mint}(params)$ to obtain (c, tr). Zerocoins are just numbers and each is a digital commitment to a random serial number. What Alice keep secret is the trapdoor tr, which is the serial number sn and the randomness r used to mint c. This is her trapdoor to use that zerocoin again.

What gives the zerocoins value is that they are put on the blockchain; one zerocoin on the blockchain costs one bitcoin. Once a mint transaction has been accepted on the blockchain the zerocoin is included in a global accumulator by miners. The currency (bitcoin) is then essentially placed in escrow and can not be accessed except through a Zerocoin spend.

The scripting language Bitcoin uses can not be used for sophisticated calculations such as verifying zero-knowledge proofs so the Zerocoin system extends Bitcoin by adding a new instruction, $\text{ZEROCOIN}_{\text{MINT}}$ [36]. Minting a zerocoin constructs a transaction with an output where $scriptPubKey$ contains this instruction and a coin c.

Spending coins. A Zerocoin spend transaction allows you to claim the bitcoins left by some other Zerocoin user. Let us assume Alice wants to spend c with Bob. She constructs a partial transaction t'. This transaction references an unclaimed mint transaction in the input I and includes Bob's address in the output

O. Alice then goes through all valid mint transactions in the block chain, assembles the set of minted coins C and outputs $(\pi, sn) \leftarrow \texttt{Spend}(params, c, tr, T, C)$ where $T = H(t')$. Recall that the serial number for the coin is released during the spending of it so that Alice can not spend the same coin twice.

When Alice gives a valid proof she can go through the blockchain, find some other zerocoin left by another user and claim the bitcoin belonging to that zerocoin back. The spend transaction (π, sn) Alice creates to spend a zerocoin claims as input some Zerocoin mint transaction. The transaction has a *scriptSig*-field containing (π, sn) and a reference to the block containing the accumulator used in π. This is the input of t'. Recall that in the proof Alice proves knowledge of a zerocoin that is on the blockchain and the randomness that causes that zerocoin commitment to open to the serial number she presents. Since the proof is zero-knowledge there is no actual way to link the address that was used to mint the original zerocoin to the address used to redeem the zerocoin. Any given coin spend is hence anonymous since it can not be traced to its withdrawal.

For an adversary not being able to simply change who the transaction is payed to, Alice needs to sign the transaction. In a normal Bitcoin transaction this is done by an ECDSA signature by the key specified in $scriptPubKey$ of the referenced input, but there is no such ECDSA for a spend transaction for an arbitrary zerocoin [36]. Alice therefore uses the signature of knowledge π to sign the transaction hash that normally would be signed using ECDSA. The proof is not just used for spending a coin but as a signature of knowledge also signing the Bitcoin address where the withdrawn bitcoins should be sent.

Verifying transactions. The proof is non-interactive so that the withdrawal of bitcoins from the escrow pool can be verified at any time. Zerocoin spend transactions must first be verified by a miner to make sure invalid transactions are not included in a block and then again by the distributed network running Bitcoin to make sure that an invalid block is not included in the blockchain. A new block should be created once every 10 minutes according to specifications of the Bitcoin protocol and if verification for blocks with a reasonable number of zerocoins takes longer than 10 minutes the network can not function.

Verification means checking that $\texttt{Verify}(params, \pi, sn, T, C) = 1$. By doing so it is verified that π is a valid signature of knowledge. It is also checked that the serial number sn does not appear in any previous transaction to prevent double-spending. The spend transaction is considered valid if these conditions hold and the referenced mint transaction is not claimed as input to a different transaction.

Alice is then allowed to redeem d bitcoins left by some other Zerocoin user and embeds c in the output of a Bitcoin transaction t that spends d (+ fees) classical bitcoins.

5.4 Proof of Security

In this section we prove security for the Zerocoin construction in Section 5.3.2 based on the security definitions with rise in the games `Z-Anonymity` and `Balance` from Section 5.2. When it comes to security our focus in this thesis has been on anonymity, but we also want to prove that the balance property is fulfilled. An electronic cash system where one could spend more coins than minted would of course be inexpedient. We are inspired by the work of Miers [35] and Miers et al. [36], but the proofs are modified and written in more detail.

Note that the proofs assume the existence of an efficient simulator and an extractor for the signature of knowledge that can extract the witness used by an adversary to construct a proof, even when the adversary is supplied with simulated proofs.

5.4.1 Anonymity

Recall that we defined the game `Z-Anonymity` whereby we wanted to ensure that an adversary $\mathcal{A}$ could not link a given coin spend transaction the the coin associated with it. We start by presenting a theorem that claims under what conditions the requirement of anonymity is satisfied. Next we give a proof of security by proving the theorem.

Theorem 4. *Assume we have a decentralized electronic cash system* Π *where the signature of knowledge* π *is computationally zero-knowledge in the random oracle model, then the adversary* $\mathcal{A}$ *in* `Z-Anonymity` *has negligible advantage.*

Recall that in `Z-Anonymity` the adversary $\mathcal{A}$ outputs a bit b' as a guess on which coin is spent and wins if $b = b'$. We want to prove that $\mathcal{A}$'s view is independent of the bit b so that the advantage of $\mathcal{A}$ must be negligible. We will perform the proof in two steps and argue in each step that $\mathcal{A}$'s view is independent of the bit b:

- Replace the signature of knowledge π provided to $\mathcal{A}$ with a simulated signature of knowledge π'.

- Replace the coins (c_0, c_1) and serial number sn provided to $\mathcal{A}$ with random values from the appropriate distributions.

Proof. We let $\mathcal{A}$ be an adversary that has advantage ϵ in `Z-Anonymity`.

Replacing the signature of knowlegde. We denote the game `Z-Anonymity` by $\mathtt{G}_0$. Further we consider a game identical to $\mathtt{G}_0$ except that the proof π provided to $\mathcal{A}$ is simulated by a ZKSoK-simulator instead of generated in `Spend`. We denote this game by $\mathtt{G}_1$. We want to argue that π can be replaced by a simulated signature π' without substantially affecting the advantage of $\mathcal{A}$ if π is computationally zero-knowledge. If the system is secure if π is computationally zero-knowledge, the system is also secure if π is statistically or perfect zero-knowledge. We know that π is zero-knowledge according to Section 5.3.3.

To argue we will show that if $\mathcal{A}$ has advantage different than ϵ when π is replaced with π', we can construct a distinguisher that determines if the proof is as in the real game ($\mathtt{G}_0$) or simulated ($\mathtt{G}_1$). We construct the game $\mathtt{G}_2$ where we assume we have an adversary $\mathcal{A}'$, a simulator $\mathcal{S}im'$ and an oracle $\mathcal{O}_{SoK}$. On input a trapdoor $\mathcal{O}_{SoK}$ samples b'' and if $b'' = 0$ outputs π as in the real game ($\mathtt{G}_0$) and a simulated proof if $b'' = 1$. The game $\mathtt{G}_2$ goes as follows (illustrated in Figure 5.5):

- The simulator $\mathcal{S}im'$ outputs $params \leftarrow \mathtt{Setup}(\lambda)$ as in the real game.
- For $i \in \{0, 1\}$ the simulator $\mathcal{S}im'$ outputs $(c_i, tr_i) \leftarrow \mathtt{Mint}(params)$ and stores the associated trapdoors (tr_0, tr_1).
- The adversary $\mathcal{A}'$ outputs $(C, T) \leftarrow \mathcal{A}'(params, c_0, c_1)$ using any strategy he wishes.
- The simulator $\mathcal{S}im'$ samples $b \xleftarrow{r} \{0, 1\}$ and runs $\mathtt{Spend}\big(params, c_b, tr_b, T, C \cup \{c_0, c_1\}\big)$ as in the real game but send queries to $\mathcal{O}_{SoK}$ to obtain π.
- The simulator $\mathcal{S}im'$ outputs (π, sn).
- The adversary $\mathcal{A}'$ outputs $b' \leftarrow \mathcal{A}'(\pi, sn)$ as a guess of which coin was spent.

If $b'' = 0$ then $\mathtt{G}_2$ is run as $\mathtt{G}_0$ and if $b'' = 1$ then $\mathtt{G}_2$ is run as $\mathtt{G}_1$. This means that if $b'' = 0$ the input to the adversary is distributed identically to the real game and the advantage is ϵ. What about when $b'' = 1$ and a simulated proof is outputted?

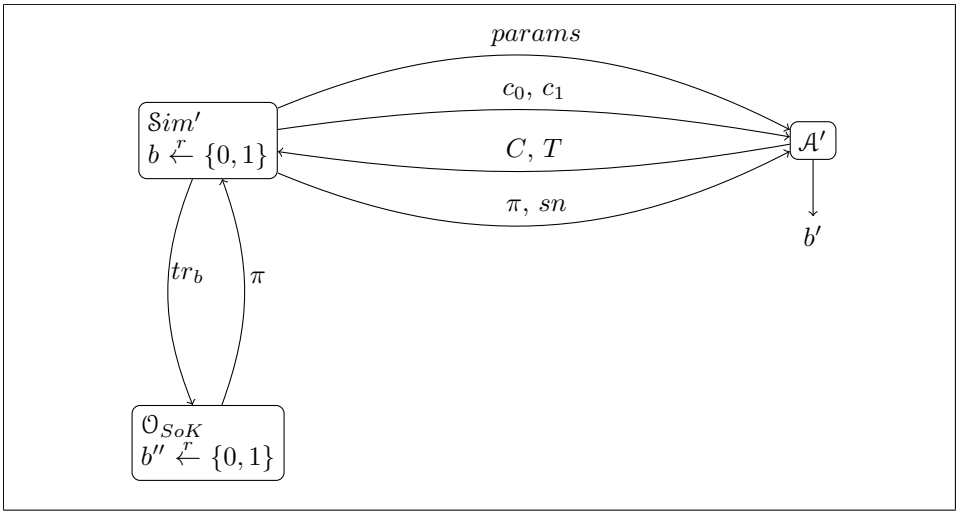

Figure 5.5: A game identical to `Z-Anonymity` in Figure 5.1 except that the simulator sends queries to an oracle to obtain the proof π.

We know according to (5.1) that

$$\begin{aligned}
\epsilon &= \left|\Pr[b = b' | \mathtt{G}_0] - \frac{1}{2}\right| \\
&= \left|\Pr[b = b' | \mathtt{G}_0] - \Pr[b = b' | \mathtt{G}_1] + \Pr[b = b' | \mathtt{G}_1] - \frac{1}{2}\right| \\
&\leq \left|\Pr[b = b' | \mathtt{G}_0] - \Pr[b = b' | \mathtt{G}_1]\right| + \left|\Pr[b = b' | \mathtt{G}_1] - \frac{1}{2}\right|.
\end{aligned}$$

By the notation $\Pr[b = b' | \mathtt{G}_0]$ we mean the probability that $b = b'$ given that we are in game $\mathtt{G}_0$, i.e. the probability that $b = b'$ in $\mathtt{G}_0$. We have that $\left|\Pr[b = b' | \mathtt{G}_1] - 1/2\right|$ is the advantage of $\mathcal{A}$ in $\mathtt{G}_1$. Since π here is simulated and does not depend on the bit b, the probability that $b = b'$ is $1/2$ and we can look away from the advantage of $\mathcal{A}$ in $\mathtt{G}_1$. This leaves us with:

$$\epsilon \leq \left|\Pr[b = b' | \mathtt{G}_0] - \Pr[b = b' | \mathtt{G}_1]\right|.$$

We observe that $\left|\Pr[b = b' | \mathtt{G}_0] - \Pr[b = b' | \mathtt{G}_1]\right|$ is the advantage of $\mathcal{A}'$ in $\mathtt{G}_2$ to guess the correct b'', i.e. the advantage against the signature of knowledge π. This tells how much information one gets from π to say something about b. Since π is computationally zero-knowledge we know that this advantage is negligible. This

means that ϵ must be negligible and moreover that $\mathcal{A}$'s view is independent of the bit b.

Replacing the coins and serial number. In this step we modify `Z-Anonymity` the following ways:

- Instead of generating the coins c_0 and c_1 by `Mint` we sample two values c_0 and c_1 at random from the set of prime numbers in $\left[X, X^2\right)$.
- We sample the serial number sn at random from $\mathbb{Z}_q$.
- Finally we simulate the signature of knowledge π.

We want to argue that a simulation with these values will have the same distribution as in the real game which will mean that $\mathcal{A}$'s view is independent of the bit b. We know from the argumentation above that a proof π generated from `Spend` as in `Z-Anonymity` can not be distinguished from that of a simulated proof. Further since the Pedersen commitment is unconditionally hiding an adversary can not distinguish whether sn is sampled and then $c = g^{sn}h^r$ computed or if c and sn are both simply sampled at random. Since the values are sampled from $\left[X, X^2\right)$ and $\mathbb{Z}_q$ the distribution of the values will be identical to that of the real game.

The argumentation of the two steps shows that $\mathcal{A}$'s view is independent of the bit b which means that $\mathcal{A}$ can not do much better than guessing which of the two minted coins was spent given two Zerocoin mints and one spend. We recall that $Adv_{\mathcal{A}} = |\Pr[b = b'] - 1/2|$ (5.1). The above argumentation gives us that ϵ, which is the advantage of $\mathcal{A}$ in `Z-Anonymity`, must be negligible as well and we have shown what we wanted. □

In Section 5.2.1 we stated that a decentralized electronic cash system like Zerocoin satisfies the requirement of anonymity if every adversary has negligible advantage in `Z-Anonymity`. By this proof we have shown that an adversary in `Z-Anonymity` has negligible advantage under the assumption that the signature of knowledge is computationally zero-knowledge in the random oracle model. This means that the requirement of anonymity for Zerocoin is satisfied under the given assumption.

5.4.2 Balance

Recall that we defined the game `Balance` whereby we wanted to ensure that an adversary $\mathcal{A}$ could not spend more coin than he mints. An adversary can spend

more coins than he mints either by stealing another user's zerocoin which entails spending a coin with the same serial number or by double-spending one of his own coins which entails assigning the coin two different serial numbers. We start by presenting a theorem that claims under what conditions the requirement of balance is satisfied. Next we give a proof of security by proving the theorem.

Theorem 5. *Assume we have a decentralized electronic cash system* Π *where the signature of knowledge* π *is sound in the random oracle model, the Strong RSA problem is hard and the Discrete Logarithm problem is hard in* $\mathbb{G}$*, then the adversary* $\mathcal{A}$ *in* `Balance` *has negligible advantage.*

Before proving Theorem 5 we provide an outline. We want to show that an adversary that wins `Balance` can be used to either find a collision in the commitment scheme, which allows us to solve the Discrete logarithm problem, or to find a collision in the accumulator, which allows us to solve the Strong RSA problem. We do so in four steps summarized as follows and detailed right below:

- We construct an adversary $\mathcal{B}$.
- We consider the cases where the simulation does not abort.
- We consider the cases where the simulation aborts.
- Finally we conclude based on the previous steps.

The proof starts by assuming we have an adversary $\mathcal{A}$ in `Balance`. We construct an adversary $\mathcal{B}$ that operates using $\mathcal{A}$ to produce coins and valid tuples for coin spend transactions. The adversary $\mathcal{B}$ uses an extractor to extract values from the spend proofs produced by $\mathcal{A}$. This gives six possible outcomes. If the run of $\mathcal{B}$ does not stop it means $\mathcal{A}$ has succeeded. Recall that $\mathcal{A}$ succeeds if he can spend another user's coin or double-spend one of his own. We show that this allows us to solve the Discrete logarithm problem. Four of the resulting outcomes result in the run of $\mathcal{B}$ stopping. We look more closely at the probability for each of these. Next we consider the probabilities all together in order to say something about the advantage of $\mathcal{B}$ in relation to the advantage of $\mathcal{A}$. We end up seeing that if we have an adversary $\mathcal{A}$ that wins `Balance` we have an adversary that solves the Discrete logarithm problem or the Strong RSA problem, so such an adversary $\mathcal{A}$ can not exist under the assumption that these problems are hard.

Proof. We assume we have an adversary $\mathcal{A}$ in `Balance`.

Constructing the adversary $\mathcal{B}$. We construct an adversary $\mathcal{B}$ that takes input (p, q, g, h) where g, h are generators for $\mathbb{G} \subseteq \mathbb{F}_p^*$ with order q and outputs $x \in \mathbb{Z}_q$ such that $g^x \equiv h \pmod{p}$. This is how $\mathcal{B}$ works:

- On input (p, q, g, h) generate the accumulator parameters (n, u) as in `AccumSetup`.
- Set $params \leftarrow (n, u, p, q, g, h)$.
- For $i = 1$ to N compute $(c_i, tr_i) \leftarrow \texttt{Mint}(params)$ where $c_i \leftarrow g^{sn_i} h^{r_i} \pmod{p}$ and $tr_i = (sn_i, r_i)$.
- Run $\mathcal{A}$ as in `Balance`.
  - The adversary $\mathcal{A}$ queries (c_i, C_i, T_i) and the simulator $\mathcal{S}im$ outputs $(\pi_i, sn_i) \leftarrow \texttt{Spend}(params_i, c_i, tr_i, T_i, C_i)$ for $i = 1$ to K.
  - Each of $\mathcal{A}$'s queries to $\mathcal{S}im$ is answered with the appropriate trapdoor information.
  - The simulator $\mathcal{S}im$ records $\big((sn_1, T_1), ..., (sn_K, T_K)\big)$.
  - A set of M coins $(c_1^*, ..., c_M^*)$ and a corresponding set of valid tuples $V_j^* = (\pi_j^*, sn_j^*, T_j^*, C_j^*)$ for $j = 1$ to $M+1$ is outputted by $\mathcal{A}$ at the end of the game.
- Apply an extractor to the j^{th} zero-knowledge proof π_j^* for $j = 1$ to $M + 1$ to extract the values $(\bar{c_j}, \bar{r_j})$, recall that the proof is

$$\pi = \text{ZKSoK}[T]\{(c, w, r) : \texttt{AccVerify}((n, u), \Lambda, c, w) = 1 \wedge c = g^{sn} h^r\}.$$

Note that in this context we use the bar to denote a value that is extracted from the proof. When the extractor is applied we get the following outcomes:

1. If the extractor fails on input π_j^*, abort and signal `Exit`.
2. If $\bar{c_j} \notin C_j^*$, abort and signal `Acc`.
3. If $\bar{c_j} \in \{c_1, ..., c_N\}$:
   (a) If $(sn_j^*, \bar{r_j}) = (sn_i, r_i)$ for some i and $T_j^* \neq T_i$, abort and signal `Forge`.
   (b) Otherwise if for some i $(sn_j^*, \bar{r_j}) = (sn_i, r_i)$, abort and signal `Col`.
   (c) Otherwise set $(sn_\delta, r_\delta) = (sn_i, r_i)$.

4. If $\bar{c_j} = \bar{c_k}$ for some k, set $(sn_\delta, r_\delta) = (sn_k^*, \bar{r_k})$.

Next these cases will be described in detail and we start with the latter two (3c and 4) where the simulation does not abort. Note in the following that the index i is used related to honest produced mints and spends from the simulator and the indices j, k related to what is produced by the adversary.

The simulation does not abort. We know by the win conditions of `Balance` that when the simulation does not abort we are able to extract $\bar{c_j}$ for $j = 1$ to $M + 1$ where $\forall\ j,\ \bar{c_j} \in C_j^* \subseteq \{c_1, ..., c_N, c_1^*, ..., c_M^*\}$ and each sn_j^* is distinct and does not match any serial number output sn_i by the simulator $\mathcal{S}im$.

The third case where $\bar{c_j} \in \{c_1, ..., c_N\}$ means that $\mathcal{A}$ tries to spend of the honest minted coins. For some i (for $i = 1$ to N) we have $\bar{c_j} = c_i$ where $c_i \in \{c_1, ..., c_N\}$. In the last opportunity for the third case (3c), if neither a forgery or collision has accrued, it means that $\mathcal{A}$ has spent one of the honest minted coins but provided a new serial number for it. The adversary $\mathcal{A}$ has produced a pair $(sn_j^*, \bar{r_j}) \neq (sn_i, r_i)$ for a coin $\bar{c_j} = c_i$. Since $\bar{c_j} = c_i$ and $c_i = g^{sn_i} h^{r_i}$ we have $(sn_\delta, r_\delta) = (sn_i, r_i)$ where $(sn_\delta, r_\delta) \neq (sn_j^*, \bar{r_j})$ and $\bar{c_j} = g^{sn_j^*} h^{\tilde{r}_j} \equiv g^{sn_\delta} h^{r_\delta} \pmod{p}$.

In fourth case (4) where $\bar{c_j} = \bar{c_k}$ for some k (for $k = 1$ to $M + 1$) we look at the case where $\mathcal{A}$ has spent a coin twice. This means that the serial number sn_k^* for $\bar{c_k}$ is not the same as the serial number sn_j^* for $\bar{c_j}$, so $(sn_j^*, \bar{r_j}) \neq (sn_k^*, \bar{r_k})$ even though $\bar{c_j} = \bar{c_k}$. We set $(sn_\delta, r_\delta) = (sn_k^*, \bar{r_k})$ which satisfies $\bar{c_j} = g^{sn_j^*} h^{\tilde{r}_j} \equiv g^{sn_\delta} h^{r_\delta} \pmod{p}$ and $(sn_\delta, r_\delta) \neq (sn_j^*, \bar{r_j})$.

We have by the soundness of π (for the cases that the simulation does not abort) the values $(\bar{c_j}, sn_j^*, \bar{r_j}, sn_\delta, r_\delta)$ where we know that $\bar{c_j} = g^{sn_j^*} h^{\tilde{r}_j} \equiv g^{sn_\delta} h^{r_\delta} \pmod{p}$ and $(sn_\delta, r_\delta) \neq (sn_j^*, \bar{r_j})$. We recall that $\mathcal{B}$ outputs $x \in \mathbb{Z}_q$ such that $g^x \equiv h \pmod{p}$. Any pair (sn_δ, r_δ) as described can be used to solve for $x = \log_g h$ as in the following:

$$\begin{aligned}
g^{sn_j^*} h^{\tilde{r}_j} &\equiv g^{sn_\delta} h^{r_\delta} \\
g^{sn_j^*} g^{-sn_\delta} &\equiv h^{r_\delta} h^{-\tilde{r}_j} \\
g^{sn_j^* - sn_\delta} &\equiv h^{r_\delta - \tilde{r}_j} \\
(g^{sn_j^* - sn_\delta})^{(r_\delta - \tilde{r}_j)^{-1}} &\equiv (h^{r_\delta - \tilde{r}_j})^{(r_\delta - \tilde{r}_j)^{-1}} \\
g^{(sn_j^* - sn_\delta)(r_\delta - \tilde{r}_j)^{-1}} &\equiv h
\end{aligned}$$

and to solve for $\log_g h$ output $(sn_j^* - sn_\delta) \cdot (r_\delta - \tilde{r}_j)^{-1} \pmod{q}$. Note that the exponent arithmetic is done modulo q and not modulo p as for the group elements.

This tells us that if we have an adversary $\mathcal{A}$ that wins `Balance` we can construct an adversary $\mathcal{B}$ that solves the Discrete logarithm problem.

We have considered the cases where $\mathcal{A}$ succeeds. Recall that $\mathcal{A}$ is the adversary in `Balance` that wants to spend more coins that he mints and he succeds if he spends another user's coin or spends the same coin twice. Next we consider the probability that the simulation aborts.

The simulation aborts. Recall that case 1, 2, 3a and 3b make the simulation abort.

1. If the extractor fails on input π_j^*, abort and signal `Exit`.

   This is simply that the extractor fails on extracting the desired values. The extractor wants to extract the values $(\bar{c_j}, \bar{r_j})$ from π_j^* for $j = 1$ to $M + 1$ tuples produced by $\mathcal{A}$. For every π_j^* the probability that the extractor fails on input π is $\mu(\lambda)$ where μ is a negligible function. By summation we have

   $$\Pr[\texttt{Exit}] \leq (M + 1)\mu(\lambda). \tag{5.7}$$

2. If $\bar{c_j} \notin C_j^*$, abort and signal `Acc`.

   This means that the extracted coin $\bar{c_j}$ is not in $C_j^* \subseteq \{c_1, ..., c_N, c_1^*, ..., c_M^*\}$ which means that $\bar{c_j}$ is not among the honest minted coins or the coins produced by $\mathcal{A}$. This will give us a collision in the accumulator. We assume we have an adversary $\mathcal{A}_2$ that induces this event, uses him to construct a Strong RSA-solver $\mathcal{B}_2$ and say that

   $$\Pr[\texttt{Acc}] = Adv_{\mathcal{B}_2}. \tag{5.8}$$

   The idea is that if you can find a witness for a non-member in a given accumulator you can solve the Strong RSA problem [12, 36]. Adversary $\mathcal{B}_2$ is given as input a Strong RSA instance (n, u), selects (p, q, g, h) as described in `Setup` in Section 5.3.2 and sets $params = (n, u, p, q, g, h)$. He generates $(c_1, ..., c_N)$ as in the run of $\mathcal{B}$ and runs $\mathcal{A}_2$ who produces a valid output $(\pi_,^* C^*)$. A $\bar{c} \notin C^*$ is extracted and the event `Acc` is induced by $\mathcal{A}_2$. Now $\mathcal{B}_2$ extracts $\bar{w}$ from π^* and uses this value, a witness for a value not accumulated, to compute a solution to the Strong RSA instance (n, u).

   We know that $\Lambda \equiv u^{c_1 c_2 \cdots c_N} \pmod{n}$. We get a witness w that $\bar{c} = c_{N+1}$ is in the accumulator Λ which means that $w^{c_{N+1}} \equiv \Lambda \pmod{n}$. Since every c

is a prime number we know that c_{N+1} and $\prod_{i=1}^{N} c_i$ are relatively prime. This means that there exists s_1, s_2 such that $s_1 c_{N+1} + s_2 \prod_{i=1}^{N} c_i = 1$. We have (where the math is done modulo n)

$$\begin{aligned} w^{c_{N+1}} &\equiv \Lambda \\ w^{c_{N+1}} &\equiv u^{\prod c_i} \\ (w^{c_{N+1}})^{s_2} &\equiv (u^{\prod c_i})^{s_2} \\ w^{s_2 c_{N+1}} &\equiv u^{s_2 \prod c_i} \\ u^{s_1 c_{N+1}} \cdot w^{s_2 c_{N+1}} &\equiv u^{s_1 c_{N+1}} \cdot u^{s_2 \prod c_i} \\ (u^{s_1} w^{s_2})^{c_{N+1}} &\equiv u^{s_1 c_{N+1} + s_2 \prod c_i} \\ (u^{s_1} w^{s_2})^{c_{N+1}} &\equiv u \end{aligned}$$

and if we set $u^{s_1} w^{s_2} = v$ and $c_{N+1} = e$ we have found v, e such that $v^e \equiv u \pmod{n}$, which means we have a solution to the Strong RSA problem. This shows that an adversary $\mathcal{A}_2$ that induces the event `Acc` can be used to construct an adversary $\mathcal{B}_2$ that solves the Strong RSA problem.

3. If $\bar{c_j} \in \{c_1, ..., c_N\}$:

   (a) If $(sn_j^*, \bar{r_j}) = (sn_i, r_i)$ for some i and $T_j^* \neq T_i$, abort and signal `Forge`.

   As mentioned the third case means that $\mathcal{A}$ tries to spend a coin that is the among the honest minted coins and for some i (for $i = 1$ to N) we have $\bar{c_j} = c_i$ where $c_i \in \{c_1, ..., c_N\}$. This first possibility (3a) where $(sn_j^*, \bar{r_j}) = (sn_i, r_i)$ and $T_j^* \neq T_i$ means that $\mathcal{A}$ has produced a coin $\bar{c_j}$ with serial number sn_j^* (with corresponding $\bar{r_j}$) that is not output and recorded by the simulator. The pair $(sn_j^*, \bar{r_j})$ is equal a pair (sn_i, r_i) for an honest minted coin for a different transaction. A coin with serial numbers sn_i exists and suddenly the serial number sn_j^* where $(sn_j^*, \bar{r_j}) = (sn_i, r_i)$ shows up for a different transaction but the simulator has not output and recorded sn_j^*. This means that $\mathcal{A}$ tries to forge $\bar{c_j}$. We assume we have an adversary $\mathcal{A}_3$ that induces this event, uses him to construct an adversary $\mathcal{B}_3$ that solves the Discrete logarithm problem and say that:

$$\Pr[\texttt{Forge}] = Adv_{\mathcal{B}_3}. \tag{5.9}$$

We construct a game as follows: adversary $\mathcal{B}_3$ is given a discrete logarithm instance $(p,\ q,\ g,\ h)$ as input. He wins if he can output $x \in \mathbb{Z}_q$ such that $g^x \equiv h \pmod{p}$. He runs $\mathcal{A}_3$ just as $\mathcal{B}$ runs $\mathcal{A}$ in the main simulation except that trapdoor information is not used to answer the queries of $\mathcal{A}_3$ to $\mathcal{S}im$. Recall that $\mathcal{S}im$ outputs $(\pi_i,\ sn_i)$ by running `Spend` with appropriate trapdoor information tr_i to $\mathcal{A}$, but to $\mathcal{A}_3$ we select random serial numbers from $\mathbb{Z}_q$ and simulate the ZKSoK responses by programming the random oracle.

We can compare $\mathcal{B}$ and $\mathcal{B}_3$ where they differ to get a more clear view of how $\mathcal{B}_3$ works. Note that $\mathcal{B}$ and $\mathcal{B}_3$ takes input $(p,\ q,\ g,\ h)$, generates $(n,\ u)$ and mint coins by $c_i \leftarrow g^{sn_i} h^{r_i}$. First we look at when $\mathcal{B}$ runs $\mathcal{A}$ with $\mathcal{S}im$:

- $\mathcal{A}$ queries $(c_i,\ T_i,\ C_i)$ for $i = 1$ to K.
- $\mathcal{S}im$ outputs $(\pi_i,\ sn_i) \leftarrow \texttt{Spend}(params_i,\ c_i,\ tr_i,\ T_i,\ C_i)$.
- $\mathcal{S}im$ records $\big((sn_1,\ T_1),\ ...,\ (sn_1,\ T_K)\big)$.
- $\mathcal{A}$ outputs a set of M coins $(c_1^*,\ ...,\ c_M^*)$ and a corresponding set of valid tuples $(\pi_j^*,\ sn_j^*,\ T_j^*,\ C_j^*)$ for $j = 1$ to $M + 1$,

and next how $\mathcal{B}_3$ runs $\mathcal{A}_3$ with $\mathcal{S}im'$:

- $\mathcal{A}_3$ queries $(c_i,\ T_i,\ C_i)$ for $i = 1$ to K.
- $\mathcal{S}im'$ outputs $(\hat{\pi}_i,\ \hat{sn}_i)$ where $\hat{\pi}_i$ is simulated and $\hat{sn}_i$ is sampled random from $\mathbb{Z}_q$.
- $\mathcal{S}im'$ records $\big((\hat{sn}_1,\ T_1),\ ...,\ (\hat{sn}_1,\ T_K)\big)$.
- $\mathcal{A}_3$ outputs a set of M coins $(c_1^*,\ ...,\ c_M^*)$ and a corresponding set of valid tuples $(\pi_j^*,\ sn_j^*,\ T_j^*,\ C_j^*)$ for $j = 1$ to $M + 1$.

Since $\mathcal{A}_3$ induces `Forge` we know for some i we have, for a coin c_j^* with belonging proof π_j^*, that $c_j^* = c_i$, $sn_j^* = \hat{sn}_i$ and $T_j^* \neq T_i$. When $\mathcal{A}_3$ outputs a forgery on a repeated serial number but for a different string we rewind $\mathcal{A}_3$ back to where the proof for this serial number sn_j^* is produced. A new proof π_j^{**} is outputted and from this proof we extract $\bar{r_j}$ to obtain the pair $(sn_j^*,\ \bar{r_j})$. This pair equals $(\hat{sn}_i,\ r_i)$. We obtain a commitment that can be opened to two different serial numbers. This makes $\mathcal{B}_3$ able to solve the Discrete logarithm problem. He does so by setting

$$c_j^* = g^{sn_j^*} h^{\bar{r}_j} \equiv g^{s\hat{n}_i} h^{r_i} \pmod{p}$$

and solves for the discrete logarithm as shown above. This shows that an adversary $\mathcal{A}_3$ that induces the event `Forge` can be used to construct an adversary $\mathcal{B}_3$ that solves the Discrete logarithm problem.

(b) Otherwise if for some i $(sn_j^*, \bar{r_j}) = (sn_i, r_i)$, abort and signal `Col`.

We are still on the third case where $\mathcal{A}$ tries to spend one of honest minted coins. This other possibility is that $(sn_j^*, \bar{r_j}) = (sn_i, r_i)$, the same as above, but without the condition $T_j^* \neq T_i$. This means that $\mathcal{A}$ has produced a pair $(sn_j^*, \bar{r_j})$ which is equal a pair (sn_i, r_i) where $\mathcal{S}im$ has output and recorded sn_i. However, $\mathcal{S}im$ has not produced sn_j^* which is equal sn_i so we have a collision in the commitment scheme.

We remember that the simulator records (sn_i, T_i) for the i^{th} query for $i = 1$ to K. There are K pairs (sn, r) that satisfies $\bar{c_j} = g^{sn}h^r$ and these are distinct. For some i the adversary $\mathcal{A}$ has produced a pair $(sn_j^*, \bar{r_j}) = (sn_i, r_i)$ where sn_j^* is not output and recorded by the simulator $\mathcal{S}im$. It is a coincidence which of the coins (more specifically the serial numbers) outputted by $\mathcal{S}im$ that $\mathcal{A}$'s produced pair collide with. Thereby we have

$$\Pr[\texttt{Col}] \leq \frac{1}{K} \,. \tag{5.10}$$

We will make use of the probability that the simulation aborts in our argumentation in the next step.

Conclusion. In (5.2) we defined the probability that $\mathcal{A}$ wins as $\Pr[E]$. We denote the event that the simulation does not abort by F and have the following result:

$$\begin{aligned} \Pr[E] &= \Pr[E|F] \cdot \Pr[F] + \Pr[E|\neg F] \cdot \Pr[\neg F] \\ &\leq \Pr[E|F] + \Pr[\neg F] \end{aligned} \tag{5.11}$$

since both $\Pr[F] \leq 1$ and $\Pr[E|\neg F] \leq 1$ (obviously). We hope the probability that the simulation does not abort is close to 1 and we are not interested in the probability that $\mathcal{A}$ wins when the simulation stops. The adversary $\mathcal{B}$ will win if the simulation does not abort and $\mathcal{A}$ wins. We denote the probability that $\mathcal{B}$ wins by D and we have $\Pr[D] = \Pr[E|F]$. We know by (5.11) that

$$\Pr[D] \geq \Pr[E] - \Pr[\neg F]. \tag{5.12}$$

We say that the advantage of $\mathcal{B}$ is the probability that $\mathcal{B}$ wins and by (5.7)-(5.10)

we can write (5.12) as

$$Adv_{\mathcal{B}} \geq Adv_{\mathcal{A}} - \left(M+1)\mu(\lambda) + Adv_{\mathcal{B}_2} + Adv_{\mathcal{B}_3} + \frac{1}{K} \right)$$
$$\implies Adv_{\mathcal{B}} + Adv_{\mathcal{B}_2} + Adv_{\mathcal{B}_3} \geq Adv_{\mathcal{A}} - \left((M+1)\mu(\lambda) + \frac{1}{K} \right). \tag{5.13}$$

In Equation 5.13 we can look away from the negligible probabilities (we assume K is large). These leaves us with

$$Adv_{\mathcal{B}} + Adv_{\mathcal{B}_2} + Adv_{\mathcal{B}_3} \geq Adv_{\mathcal{A}}. \tag{5.14}$$

By considering (5.14) we see that if $Adv_{\mathcal{A}}$ is great then the sum of $Adv_{\mathcal{B}}$, $Adv_{\mathcal{B}_2}$ and $Adv_{\mathcal{B}_3}$ must be great. This means that at least one of these advantages must be great. We recall that $Adv_{\mathcal{B}}$ and $Adv_{\mathcal{B}_3}$ are advantages for solving the Discrete logarithm problem and $Adv_{\mathcal{B}_2}$ for solving the Strong RSA problem. This means that if we have an adversary $\mathcal{A}$ that wins `Balance` we have an adversary that solves the Discrete logarithm problem or the Strong RSA problem. Under the assumption that these problems are hard the advantage of $\mathcal{A}$ must be negligible and we have shown what we wanted. □

In Section 5.2.2 we stated that a decentralized electronic cash system like Zerocoin satisfies the requirement of balance if every adversary has negligible advantage in `Balance`. By this proof we have shown that an adversary in `Balance` has negligible advantage under the assumption that the signature of knowledge is sound in the random oracle model, the Strong RSA problem is hard and the Discrete Logarithm problem is hard. This means that the requirement of balance for Zerocoin is satisfied under these assumptions.

Before we close this section we want to note, as pointed out by Miers [35], that the definition of balance is incomplete in the sense that it does not ensure that a party can actually spend the money they are paid. It is also noted that even if Zerocoin is cryptographically secure against double-spending and forgery of zerocoins, Bitcoin upon which it depends is not, so that double-spends and forgery of zerocoins can be accomplished by breaking Bitcoin and without touching the underlying cryptographic primitives of Zerocoin [24]. We will not discuss this any further and close our description of Zerocoin. We review the system further in the next chapter.

Chapter 6

Discussion

The final chapter of this thesis starts with a comparison between anonymity for Bitcoin and Zerocoin, and describe how linking of addresses is harder for Zerocoin than it is for Bitcoin. However, Zerocoin also imposes limitations regarding anonymity and some aspects on this will be described. Finally we briefly consider work presented to improve the Zerocoin system. The first section is mainly independent and is based on what is previously presented in the thesis. The second section is mainly based on the original Zerocoin paper [36] while the last section presents several papers.

6.1 A Review of Anonymity

To make a comparison between anonymity in Bitcoin and Zerocoin we take a look back at our games `B-Anonymity` and `Z-Anonymity`. In `B-Anonymity` for Bitcoin we wanted to ensure that an adversary could not link given addresses to a user of his choice. In `Z-Anonymity` we wanted to ensure that an adversary could not link a given coin spend transaction to the coin associated with it which implies linking the address used to mint the original zerocoin to the address used to redeem the zerocoin. Both games enhances the fact that we want addresses to be unlinkable, which is the kind of anonymity we have required from Bitcoin and Zerocoin in this thesis.

We showed how an adversary in `B-Anonymity` could link addresses by clustering them together based on the heuristic regarding multi-input transactions (and further link address clusters to real-world identities). For Zerocoin we proved that an

adversary in `Z-Anonymity` had only negligible advantage. Based on these results one can say that Zerocoin provides stronger anonymity guarantees when it comes to address unlinkability than Bitcoin.

In a normal Bitcoin transaction history each transaction is linked to a preceding transaction and we showed how data obtained from the Bitcoin blockchain can support linking of addresses (referring to Section 4.2). Remember that when you mint a zerocoin your bitcoins go into an escrow pool and there is nothing that makes anyone else able to spend it unless it gets taken out by a Zerocoin spend transaction with a proof. When you give a valid proof you can go through the blockchain, find some other zerocoin and claim the bitcoin belonging to that zerocoin back.

Since the proof is zero-knowledge, user anonymity is achieved. There is no actual way to link the address that was used to mint the original zerocoin to the address used to redeem the zerocoin. Nobody knows which serial number corresponds to which zerocoin. If the serial number for a minted coin was to be linked to the serial number released in a withdrawal one must either know the randomness that causes the commitment to open or directly which coin is proven knowledge of, but non of these are revealed by the proof. All an adversary could deduce from your Zerocoin mint and Zerocoin spend is that you are one of the (many) users who did a Zerocoin mint but not who you are. Figure 6.1 shows a Bitcoin blockchain and a Zerocoin blockchain and illustrates how Zerocoin breaks the linkage between coins.

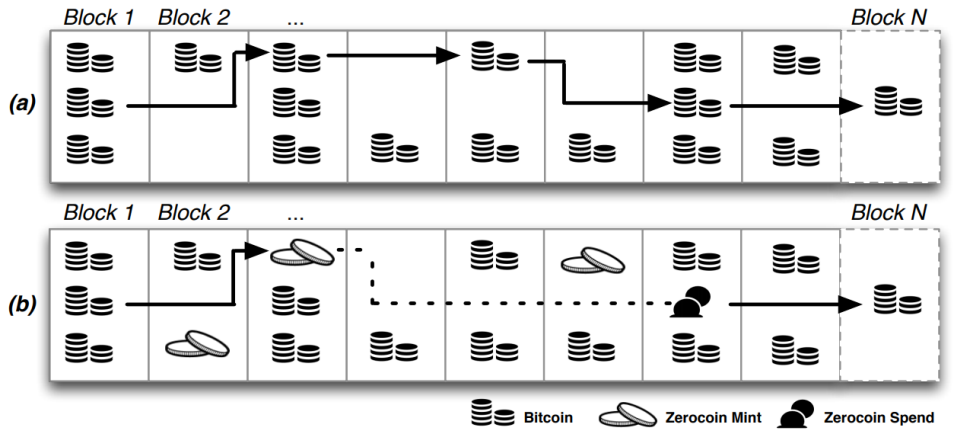

Figure 6.1: The figure illustrates two chains; a normal Bitcoin transaction chain (a) and a Zerocoin chain (b) and shows how in a Zerocoin chain the linkage between mint and spend can not be determined from the block chain data. The figure is copied from Miers et al. [36].

The upper chain in Figure 6.1 is a Bitcoin blockchain. Each transaction is linked to a preceding transaction as the bitcoins in the output of a transaction in a block is used as input for a transaction in another block (described in Chapter 3). Through this linkage one can follow bitcoins in the chain and cluster addresses based on the movement of bitcoins as we showed earlier.

The chain below the Bitcoin blockchain on Figure 6.1 is a Zerocoin blockchain. As one can see bitcoins are exchanged for zerocoins through a Zerocoin mint and bitcoins taken out again by a Zerocoin spend. This is why Zerocoin can be refereed to as a decentralized laundry or mix. Assume that you put in one bitcoin, exchange it for a zerocoin and exchange your zerocoin for another bitcoin in return, this breaks the movement of that bitcoin in the chain as the dotted line in Figure 6.1 represents. It is impossible to connect the two trades.

The reasoning for clustering addresses from Section 4.2 will not hold in the case where Zerocoin is used as the transaction history (for example between t_1, t_2, t_3 and t_4 in Figure 4.4) is broken. We can not construct such transaction and user networks to cluster addresses since we can not represent the flow of bitcoins between transactions or users in the same way. The link is broken through the use of Zerocoin as Figure 6.1 shows. This makes linkage of addresses harder with the use of Zerocoin than it is for Bitcoin and we can say that Zerocoin amplify the anonymity guarantees in Bitcoin. However, the Zerocoin system also imposes limitations.

6.2 Limitations of Zerocoin Anonymity

Zerocoin reveals the destination and denomination of transactions, which is potential source for adversaries who want to attack the anonymity of the system. Miers et al. [36] claim that this may be an advantage rather than a loss since the bank can be considered an adversarial party. They further claim that the problem with denomination can be avoided by using Zerocoin to anonymize bitcoins or to fix one or a small set of coin denominations and exchange coins until one has those denominations.

In previous electronic cash systems information about the number of minted and spent coins are revealed primarily to merchants and the bank, but in a decentralized electronic cash system like Zerocoin this information is revealed to all users of the system. Let us assume N coins are minted in `Mint` and subsequently spent in `Spend`. This gives us an anonymity set of size n. If another coin is minted after the

last spending of the N coins the size of the anonymity set for the next spend is not $n + 1$ but 1 because all observers know that the previous coins have been spent.

Another aspect of anonymity is the minted zerocoins put on the blockchain. If Alice is the only one putting zerocoins on the blockchain she (obviously) gets no anonymity. A sufficient number of independent users is required to obtain anonymity. We can consider an attack scenario where someone maliciously tries to put a lot of zerocoins on the blockchain and try to de-anonymize, but an adversary can not do much better than that. Since an adversary can mint a large fraction of the coins, a lower bound on the provided anonymity is the number of coins minted by honest parties before a coin is spent and an upper bound is the total set of minted coins.

The loss of wallets is a serious concern in the Bitcoin community. This can also threaten anonymity in the setting of Zerocoin. We recall that the trapdoor for a minted zerocoin is stored. By using the stored trapdoor an adversary can de-anonymize Zerocoin transactions. Miers et al. [36] presents two proposals for this problem. The first is to securely delete the trapdoor immediately after the spending of a coin, but if the trapdoor is stolen at some earlier point this provides no protection. The second proposal is to generate the spend transaction immediately after a coin is minted. Such a solution makes an adversary who gets a hold of the wallet unable to link any zerocoins in it to their mint transactions, but reduces the anonymity of the user by decreasing the number of coins in the set of all minted coins. In addition, spending a coin right after its minting will leak some information about when a coin was minted and may impair the anonymity provided.

6.3 Further Work

In this section we briefly present proposals that are presented to improve Zerocoin. In the original Zerocoin paper [36] it is presented several experiments conducted to test the performance of Zerocoin. The authors note that the need for double discrete logarithm proofs of knowledge to redeem zerocoins leads to large proof size and verification time. They further note that a scheme with both smaller proofs and greater speed is preferable.

Zerocoin also has drawbacks when it comes to functionality. We have already noted that Zerocoin reveals the amount and other metadata about transactions even though a payment transaction is unlinked from its origin address. There are also other regards on funtionality [47]:

- Zerocoin uses coins of fixed denominations; does not support payments of exact values or provide a means to divide coins (make change following a transaction).
- Zerocoin has no mechanism for a user to pay another user directly in zerocoins, the payment is conducted through bitcoins.

The authors of the original Zerocoin paper [36] present in a new paper [24] how to modify the Zerocoin protocol to create divisible coins. This means that every zerocoin can contain an arbitrary individual denomination. This denomination can again can be divided into new coins of arbitrary value. They propose to commit to both a serial number and a balance for a coin that can be divided into new coins instead of committing to a serial number only. They also consider how to improve the anonymity in Zerocoin. We refer the reader to the paper [24] for more on this.

As to reduce the proof size and quicker the verification in Zerocoin, Danezis et al. [19] presents *Pinocchio coin*. The system takes use of Pinocchio [41] which is a proof system for efficiently verifying general computations while relying on cryptographic assumptions. The authors note that the proof size for Pinocchio zerocoins (344 bytes) is comparable with existing Bitcoin transactions contrary to the proof size of Strong RSA zerocoins (50 kb). However, the protocol of Pinocchio coin only supports coins of fixed denomination and is best viewed as a decentralized mix like Zerocoin.

Pinocchio coin takes use of what is called *zK-SNARKs* to reduce proof size and verification time in Zerocoin. A system called *Zerocash* [47] also takes use of zK-SNARKs to reduce the proof size and quicker the verification of Zerocoin and provides anonymous payments of any amount. A zk-SNARK (zero-knowledge succinct non-interactive argument of knowledge) is an efficient variant of a zero-knowledge proof of knowledge. The proof is *succinct* meaning that the proofs are very short and easy to verify. In addition to provide anonymous payments of any amounts, Zerocash improves performance and functionality compared to Zerocoin by [47]:

- reducing the size of transactions spending a coin by 97.7 %;
- reducing the verification time of spend transactions by 98.6 %;
- allowing for payments to be made directly to a user's fixed address without user interaction;
- hiding transaction amounts and the values of coins held by a user.

While Zerocoin is not a full-fledged anonymous currency and uses Bitcoin as baking currency, Zerocash makes it possible to entirely replace traditional Bitcoin payments with anonymous alternatives. Zerocash is implemented into the cryptocurrency *Zcash* (the cryptocurrency *Zcoin* uses the Zerocoin protocol) [51]. However, there are also drawbacks to Zerocash, for example that it requires protocol modification and relies on a trusted setup for generating parameters to the implementation of zk-SNARKs. Some of the authors of Zerocash propose to use a modified and separate instance of Zerocash to create coins that allows search for accountable user tracing [23], but we will not explore this any further. We refer the reader to the Zerocash paper [47] for details of the Zerocash system.

There are many proposals that seek to improve the anonymity of Bitcoin, not just the ones mentioned here, and they all come with advantages and disadvantages [16]. In this thesis we have focused on Zerocoin. Zerocoin provides stronger anonymity but require substantial modifications to Bitcoin and derive its anonymity (and security against counterfeiting) from cryptographic assumptions at the cost of increased computational complexity and size. This again results in proposals to improve Zerocoin, so development is not standstill. As stated in the Zerocoin paper [36] it is reasonable to believe that further research will lead to different trade-offs between security, accountability and anonymity, and it will be exciting to see how research in the field of cryptocurrencies and anonymity evolves in the future.

Bibliography

[1] Elli Androulaki, Ghassan O Karame, Marc Roeschlin, Tobias Scherer, and Srdjan Capkun. Evaluating user privacy in Bitcoin. In *International Conference on Financial Cryptography and Data Security*, pages 34–51. Springer, 2013.

[2] Andreas M Antonopoulos. *Mastering Bitcoin: unlocking digital cryptocurrencies*. O'Reilly Media, Inc., 2014.

[3] Simon Barber, Xavier Boyen, Elaine Shi, and Ersin Uzun. Bitter to better — how to make Bitcoin a better currency. In *International Conference on Financial Cryptography and Data Security*, pages 399–414. Springer, 2012.

[4] Niko Barić and Birgit Pfitzmann. Collision-free accumulators and fail-stop signature schemes without trees. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 480–494. Springer, 1997.

[5] Mihir Bellare and Oded Goldreich. On defining proofs of knowledge. In *Annual International Cryptology Conference*, pages 390–420. Springer, 1992.

[6] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *Proceedings of the 1st ACM conference on Computer and communications security*, pages 62–73. ACM, 1993.

[7] Josh Benaloh and Michael de Mare. One-way accumulators: A decentralized alternative to digital signatures. In *Workshop on the Theory and Application of of Cryptographic Techniques*, pages 274–285. Springer, 1993.

[8] David Bernhard, Olivier Pereira, and Bogdan Warinschi. How not to prove yourself: Pitfalls of the Fiat-Shamir heuristic and applications to Helios. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 626–643. Springer, 2012.

[9] Joseph Bonneau, Andrew Miller, Jeremy Clark, Arvind Narayanan, Joshua A Kroll, and Edward W Felten. Sok: Research perspectives and challenges for Bitcoin and cryptocurrencies. In *Security and Privacy (SP), 2015 IEEE Symposium on*, pages 104–121. IEEE, 2015.

[10] Joseph Bonneau, Arvind Narayanan, Andrew Miller, Jeremy Clark, Joshua A Kroll, and Edward W Felten. Mixcoin: Anonymity for Bitcoin with accountable mixes. In *International Conference on Financial Cryptography and Data Security*, pages 486–504. Springer, 2014.

[11] Jan Camenisch. *Group signature schemes and payment systems based on the discrete logarithm problem.* PhD thesis, ETH Zurich, 1998.

[12] Jan Camenisch and Anna Lysyanskaya. Dynamic accumulators and application to efficient revocation of anonymous credentials. In *Annual International Cryptology Conference*, pages 61–76. Springer, 2002.

[13] Jan Camenisch and Markus Stadler. Efficient group signature schemes for large groups. In *Annual International Cryptology Conference*, pages 410–424. Springer, 1997.

[14] Melissa Chase and Anna Lysyanskaya. On signatures of knowledge. In *Annual International Cryptology Conference*, pages 78–96. Springer, 2006.

[15] David Chaum. Blind signatures for untraceable payments. In *Advances in cryptology*, pages 199–203. Springer, 1983.

[16] Mauro Conti, Chhagan Lal, Sushmita Ruj, et al. A survey on security and privacy issues of Bitcoin. *arXiv preprint arXiv:1706.00916*, 2017.

[17] Ronald Cramer and Victor Shoup. Signature schemes based on the strong RSA assumption. *ACM Transactions on Information and System Security (TISSEC)*, 3(3):161–185, 2000.

[18] Ivan Damgård. Commitment schemes and zero-knowledge protocols. In *School organized by the European Educational Forum*, pages 63–86. Springer, 1998.

[19] George Danezis, Cedric Fournet, Markulf Kohlweiss, and Bryan Parno. Pinocchio coin: building Zerocoin from a succinct pairing-based proof system. In *Proceedings of the First ACM workshop on Language support for privacy-enhancing technologies*, pages 27–30. ACM, 2013.

[20] Joan Antoni Donet Donet, Cristina Pérez-Sola, and Jordi Herrera-Joancomartí. The Bitcoin P2P network. In *International Conference on Financial Cryptography and Data Security*, pages 87–102. Springer, 2014.

[21] Ittay Eyal and Emin Gün Sirer. Majority is not enough: Bitcoin mining is vulnerable. In *International conference on financial cryptography and data security*, pages 436–454. Springer, 2014.

[22] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *Conference on the Theory and Application of Cryptographic Techniques*, pages 186–194. Springer, 1986.

[23] Christina Garman, Matthew Green, and Ian Miers. Accountable privacy for decentralized anonymous payments. In *International Conference on Financial Cryptography and Data Security*, pages 81–98. Springer, 2016.

[24] Christina Garman, Matthew Green, Ian Miers, and Aviel D Rubin. Rational zero: Economic security for Zerocoin with everlasting anonymity. In *International Conference on Financial Cryptography and Data Security*, pages 140–155. Springer, 2014.

[25] Oded Goldreich. A short tutorial of zero-knowledge, 2013.

[26] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on computing*, 18(1):186–208, 1989.

[27] Jens Groth. *Honest verifier zero-knowledge arguments applied.* BRICS, 2004.

[28] Carmit Hazay and Yehuda Lindell. A note on zero-knowledge proofs of knowledge and the ZKPOK ideal functionality. *IACR Cryptology ePrint Archive*, 2010:552, 2010.

[29] Jordi Herrera-Joancomartí. Research and challenges on Bitcoin anonymity. In *Data Privacy Management, Autonomous Spontaneous Security, and Security Assurance*, pages 3–16. Springer, 2015.

[30] Philip Koshy, Diana Koshy, and Patrick McDaniel. An analysis of anonymity in Bitcoin using P2P network traffic. In *International Conference on Financial Cryptography and Data Security*, pages 469–485. Springer, 2014.

[31] Jan Lansky. Possible state approaches to cryptocurrencies. *Journal of Systems Integration*, 9(1):19, 2018.

[32] Yehuda Lindell. How to simulate it - a tutorial on the simulation proof technique. In *Tutorials on the Foundations of Cryptography*, pages 277–346. Springer, 2017.

[33] Sarah Meiklejohn and Claudio Orlandi. Privacy-enhancing overlays in Bitcoin. In *International Conference on Financial Cryptography and Data Security*, pages 127–141. Springer, 2015.

[34] Sarah Meiklejohn, Marjori Pomarole, Grant Jordan, Kirill Levchenko, Damon McCoy, Geoffrey M Voelker, and Stefan Savage. A fistful of bitcoins: characterizing payments among men with no names. In *Proceedings of the 2013 conference on Internet measurement conference*, pages 127–140. ACM, 2013.

[35] Ian Miers. Decentralized anonymous payments, 2017.

[36] Ian Miers, Christina Garman, Matthew Green, and Aviel D Rubin. Zerocoin: Anonymous distributed e-cash from Bitcoin. In *Security and Privacy (SP), 2013 IEEE Symposium on*, pages 397–411. IEEE, 2013.

[37] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008.

[38] Arvind Narayanan, Joseph Bonneau, Edward Felten, Andrew Miller, and Steven Goldfeder. *Bitcoin and cryptocurrency technologies: a comprehensive introduction*. Princeton University Press, 2016.

[39] Micha Ober, Stefan Katzenbeisser, and Kay Hamacher. Structure and anonymity of the Bitcoin transaction graph. *Future internet*, 5(2):237–250, 2013.

[40] Christof Paar and Jan Pelzl. *Understanding cryptography: a textbook for students and practitioners*. Springer Science and Business Media, 2009.

[41] Bryan Parno, Jon Howell, Craig Gentry, and Mariana Raykova. Pinocchio: Nearly practical verifiable computation. In *Security and Privacy (SP), 2013 IEEE Symposium on*, pages 238–252. IEEE, 2013.

[42] Torben Pryds Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *Annual International Cryptology Conference*, pages 129–140. Springer, 1991.

[43] Andreas Pfitzmann and Marit Köhntopp. Anonymity, unobservability, and pseudonymity — a proposal for terminology. In *Designing privacy enhancing technologies*, pages 1–9. Springer, 2001.

[44] Fergal Reid and Martin Harrigan. An analysis of anonymity in the Bitcoin system. In *Security and privacy in social networks*, pages 197–223. Springer, 2013.

[45] Ransom Richardson and Joe Kilian. On the concurrent composition of zero-knowledge proofs. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 415–431. Springer, 1999.

[46] Dorit Ron and Adi Shamir. Quantitative analysis of the full Bitcoin transaction graph. In *International Conference on Financial Cryptography and Data Security*, pages 6–24. Springer, 2013.

[47] Eli Ben Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized anonymous payments from Bitcoin (extended version). In *Security and Privacy (SP), 2014 IEEE Symposium on*. IEEE, 2014.

[48] Claus-Peter Schnorr. Efficient identification and signatures for smart cards. In *Conference on the Theory and Application of Cryptology*, pages 239–252. Springer, 1989.

[49] Victor Shoup and Joël Alwen. σ-protocols continued & introduction to zero knowledge. 2007. `https://cs.nyu.edu/courses/spring07/G22.3220-001/lec3.pdf`. Accessed: 2018-03-06.

[50] Douglas R Stinson. *Cryptography: theory and practice*. CRC press, 2005.

[51] Zcoin. Zcoin and Zcash: similarities and differences. `https://zcoin.io/zcoin-and-zcash`. Accessed: 2018-04-30.