Making Privacy in Digital Currency Practical by Improving the Efficiency of the Zerocoin Protocol

Shao Fei

Supervisor: Prof. Bharadwaj Veeravalli

Advisor: Jestine Paul

Department of Electrical and Computer Engineering
National University of Singapore

This interim report is submitted for the degree of
B.Eng (Computer Engineering)

October 2016

Abstract

Digital currencies such as Bitcoin provide some level of privacy because users can use multiple arbitrary identities to make transactions. However the caveat is that transaction records are public and unencrypted, and thus there are various ways to de-anonymise users. This research uses Bitcoin as a starting point to analyse the issues surrounding privacy in digital currencies and explores the various ways to overcome these issues. Zerocoin, an anonymous digital currency protocol based on Non-Interactive Zero-knowledge proofs, shows promise and is examined in detail. The main performance drawbacks in Zerocoin are the large computational and storage requirements for its proofs. If these problems are addressed, Zerocoin can be a commercially viable anonymous digital currency. This research attempts to reduce the performance overheads in Zerocoin by modifying the Accumulator Proof of Knowledge (AccPoK) and Serial Number Signature of Knowledge (SNSoK) in the protocol. The proposed SNSoK is a modification of the proof for a committed value in a Pedersen commitment, and is smaller and faster that the original SNSoK by about 40 times and 80 times respectively. The proposed AccPoK employs an optimisation technique that is used in the original SNSoK, but its performance has not been tested. Going forward, this research plans to continue enhancing the proposed improvements and simulate a Zerocoin network to test them on a network level. If time permits, this research will also explore ways to enhance the functionality aspects of Zerocoin.

Table of contents

List of figures

List of tables

Introduction

Privacy is a major theme in any financial transaction system. While traditional banking systems offer privacy by keeping transaction records private, the centralised records can be exposed either maliciously or under the order of authorities. Since banks keep the real identities of transaction owners for accountability, privacy is completely destroyed when the records are exposed. Bitcoin as a digital currency system addresses some of the privacy problems of traditional banking by providing a level of "pseudonymity" through the use of arbitrary public key addresses as identities during payments (§1.2). This has partly driven the immense popularity of Bitcoin, which has a market capitalisation of USD$10 billion and daily transactions volume of USD$50 million as of October 2016[1].

However due to the decentralised nature of Bitcoin, Bitcoin transaction records are unencrypted and publicly available. Even though transactions are executed with cryptic public key addresses, there are many techniques to de-anonymise Bitcoin transactions (§1.4). In order to preserve the privacy of Bitcoin, different innovations has been developed. Methods such Mixing can be easily integrated with Bitcoin, while others require separate digital currency systems, referred to as Altcoins, to be implemented (§2).

One Altcoin, Zerocoin, which makes use of Non-interactive Zero Knowledge (NIZK) proofs to create and validate anonymous transactions, is identified as the subject of research due to its theoretical soundness and practical potential (§2.3.3). This research attempts to address the computational and storage inefficiencies in Zerocoin (§2.3.1) so that it can be a viable substitute to the current Bitcoin system. This will be mainly done by modifying the NIZK proofs algorithms and related data structures (§3.1.3) in the Zerocoin protocol. The proposed improvements will be benchmarked against the original Zerocoin protocol in terms of privacy preservation and computational performance (§4.2). The objective of the research is to achieve a considerable improvement in computational performance without significant compromise in privacy preservation.

At this point, this research has examined the theoretical basis and the construction of Zerocoin (§3). The source code of the Zerocoin protocol developed by its creators has also

[1] Based on statistics from https://coinmarketcap.com/

been analysed and some areas for improvement have been identified (§3.3). Improvements have also been proposed and tested by modifying the Zerocoin source code (§5 and §6). This research plans to continue refining the proposed improvements and test their performance on a network level by simulating a peer-to-peer Zerocoin network (§4.2).

Chapter 1

Privacy in Bitcoin

1.1 Overview of Bitcoin

To facilitate better understanding of the issues surrounding anonymity in Bitcoin and digital currencies in general, this paper introduces a basic model of Bitcoin. For conciseness, only parts of Bitcoin that relate to anonymity are presented and the other concepts are simplified. The explanations of Bitcoin are adapted from [2] and [20].

As defined by to the creator of Bitcoin Satoshi Nakamoto, Bitcoin is a purely peer-to-peer electronic cash system that allows direct online payments without going through a financial institution [19]. When a payer wants to make a payment, he constructs a *transaction* and broadcast it to the Bitcoin peer-to-peer network, where it can be validated by any peer (interchangeably referred to as node) through its digital signature. A simplified data structure for a *transaction* is shown in Fig. 1.1:

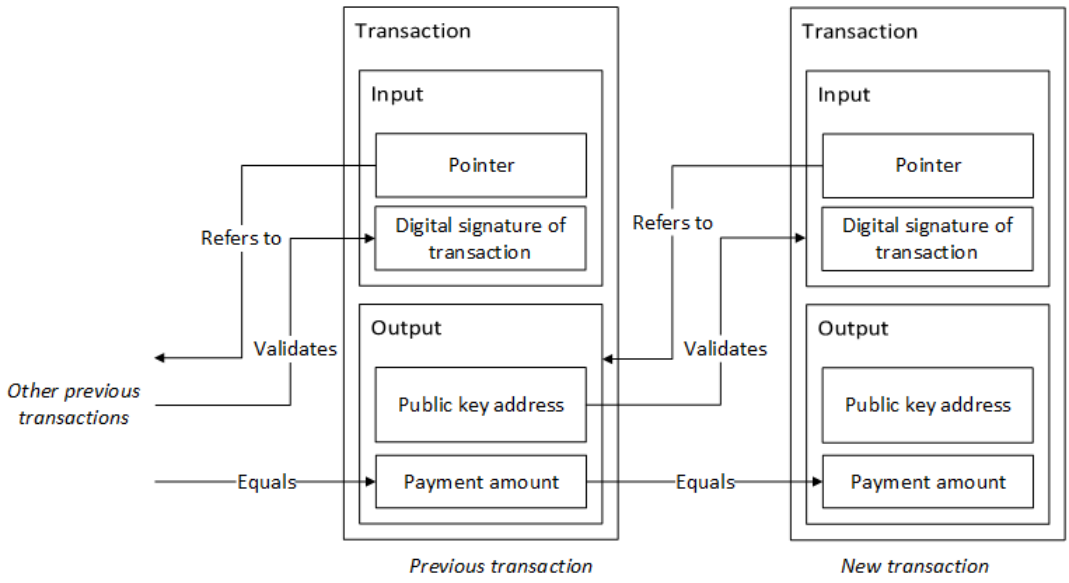

Fig. 1.1 Bitcoin *transactions*

Each *transaction* has a set of *outputs* which denotes public key addresses (destination of payment) and the payment amounts. For a payment to be valid, the payer must have the funds to make the payment. The set of *inputs* in a *transaction* denotes the source of funds used in the payment. To denote the payer's source of funds, an *input* refers to a previous *output* that has been paid to the public key address of the payer. Essentially a payer is using some funds that are previously paid to him/her to pay the payee. In order for the *transaction* to be valid, the payer must indeed own the public keys address that the previous *output* has paid to. Nodes in the Bitcoin network validates this by checking whether the digital signature signed by the payer's private key in the *input* is valid under the public key address in the previous *output* referred to by the *input*. If the digital signature is valid, it means that the payer knows the private key that corresponds to the public key address that the previous *output* has paid to, and he indeed owns the public key address. The example above shows the case where there is only one *input* and one *output* in a transaction, but a *transaction* can have multiple *inputs* and *outputs*. In such cases, the payer is combining sources of funds from multiple public key addresses and paying them to multiple payment addresses. At this point, it can be noted that a user can own multiple public key addresses which is achieved by simply creating them.

In addition to proof of ownership of funds, the amounts referenced by in the *input* and the amount stated in the *output* of a *transaction* must also be equal. For cases where there are multiple *inputs* and *outputs* in a transaction, the total amount in the previous *outputs* referenced by the *inputs* must equal to the total amount stated by the *outputs*. Essentially, this means that the source of payment must be in the correct amount to fulfil the payment.

Another point note is that a previous *output* referenced by an *input* must be consumed in full, which means that the funds in the previous *output* cannot be broken down to make a payment of an exact amount. Change is given back to the payer by creating an additional *output* in the *transaction* that pays back the excess amount to one of the payer's public key address. This mechanism ensures that the *input* and *output* amounts of a *transaction* always tally. Lastly, to prevent double spending, the previous *output* that is referenced by the *input* must not be already referenced by the *input* of another *transaction*.

Once a node has performed all the above checks and validated a transaction, it forwards the *transaction* to its neighbours who carry on the process. If the node who receives the *transaction* happens to be a miner node, it combines the *transaction* with other *transactions* to form a block, and attempts to publish the *block* to the Blockchain through the consensus protocol. The new *block* that passes the consensus protocol is accepted by all nodes and extends the Blockchain by pointing to the latest *block* in the Blockchain. A simplified example illustrating the relationships between *blocks* and *transactions* in the Blockchain is shown in Fig. 1.2.

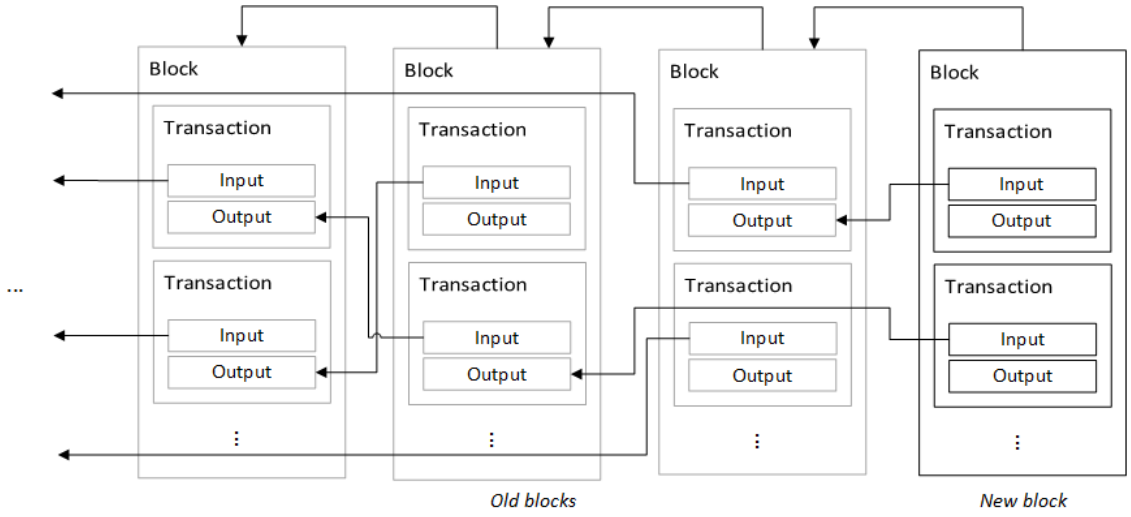

Fig. 1.2 *blocks* and *transactions* in the Blockchain

As seen, the Blockchain contains nothing but chains of transfers of ownership of funds from the public key address in one *output* to the public key address in another *output* across different *blocks*. Those *outputs* that are not referred to by any *input* represent unspent funds, and can be referenced to by a new transaction.

Once a *transaction* has made it to the Blockchain for some while, the payment is considered confirmed. The append-only property of the Blockchain prevents confirmed

transaction from being tampered, while the public nature of the Blockchain allows anyone to check for the validity of a *transaction*.These features ensures that Bitcoin payments are secure.

1.2 Pseudonymity of Bitcoin

Pseudonymity refers to the case of a user using an identity that is not his/her real identity [20]. Bitcoin achieves pseudonymity as payments made with public key addresses instead of real world identities. These addresses are generated randomly, and a user can generate as many public key addresses as he wants to further hide his/her identity. Hence it is impossible to tell who a public key address belongs to given only the information of the public key address. However pseudonymity alone in Bitcoin does not achieve privacy as public key addresses can still be linked to real world identities given other information (§1.4). Due to the public nature of the Blockchain, once a real world identity is linked to a public key address, all the *transactions* in the past, present and future using that address can be linked to that real world identity.

1.3 Definition of Anonymity and Anonymity Set

To achieve privacy in Bitcoin, the anonymity property needs to be satisfied. Anonymity in the context of Bitcoin requires pseudonymity and unlinkability [20]. ***Unlinkability*** refers to the case that it is hard to:

1. Link different public key addresses of the same user
2. Link different *transactions* made by the same user
3. Link a payer to the payee

If the above properties are satisfied, Bitcoin and digital currency systems in general can be truly anonymous and *transactions* can be fully private.

Another term that is relevant to anonymity is the ***anonymity set*** [20]. An anonymity set in Bitcoin is the set of *transactions* in which an adversary cannot distinguish a particular *transaction* from. In other words, even if an adversary knows that a *transaction* is linked to a particular user, he cannot identify which *transaction* it is if that *transaction* is in its anonymity set. An anonymity set containing one *transaction* essentially implies no anonymity, while an anonymity set containing all the *transactions* on the Blockchain implies full anonymity.

Hence the size of the anonymity set in a protocol can be used to measure its level of anonymity.

1.4 Ways to De-anonymise Bitcoin

1.4.1 Linking Payers to Payee

Given the properties of anonymity, it can be seen that Bitcoin is not anonymous. The current Bitcoin protocol clearly violates the property that it should be hard to link a payer to the payee. The *input* of a *transaction* provides the link to the payer's public key address, while the *output* of a *transaction* states the payee's public key address. For cases of direct payments without the use of an intermediary, anyone can easily tell the link between the payer and the payee since all *transactions* are public on the Blockchain.

1.4.2 Linking Public Key Addresses and Transactions

The other two anonymity properties apply to cases where users use different public key addresses for payments. These two properties can be violated when the different public key addresses used by the same user are linked together to form a ***cluster***. If this is achieved, then the only step left is to associate a real world identity to the cluster to fully de-anonymise the user.

The first step of forming clusters of addresses can be achieved by inferring joint control of addresses through ***shared spending*** and ***shadow addresses*** [1]. This technique is illustrated in Fig. 1.3.

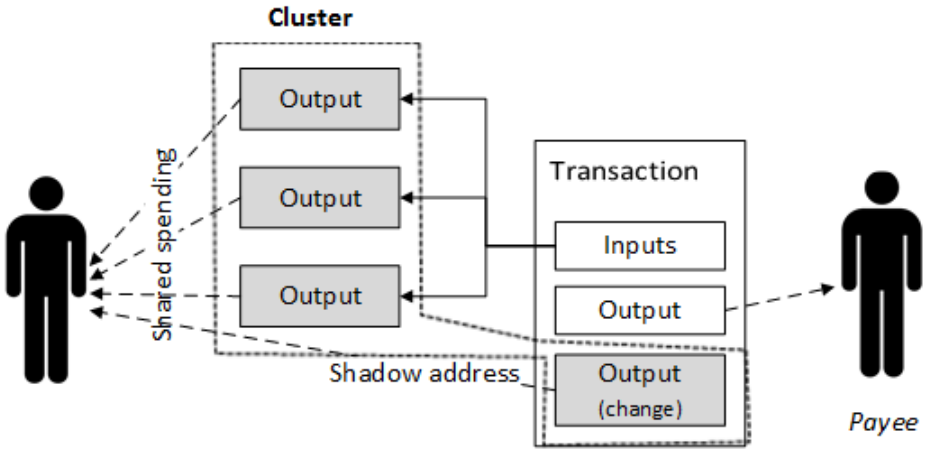

Fig. 1.3 Clustering by inferring joint control of addresses

A *transaction* can contain shared spending by having multiple *inputs* that refer to a different previous *outputs* (§1.1). One of the main reason for this kind of *transaction* is that the payment amount is too big to be fulfilled by any single *output* paid to the payer and the payer has to combine funds from different *outputs* to make the payment. An adversary can make use of this spending pattern and infer that the public key addresses in the *outputs* referred to by the *inputs* in a *transaction* belong to the same payer. In addition, a *transaction* often contains an *output* that pays back the change to the payer (§1.1). The public key addresses in these *outputs*. referred to as shadow addresses, are in fact the addresses of payer. Hence an adversary can attempt extract these shadow addresses from a *transaction*.These two techniques can be used to form clusters of public key addresses that belong the same user. In the above example, the *outputs* in grey form a cluster and the public key addresses in these *outputs* belong to the Payer. Once clusters have been formed, different clusters can be linked to each other transitively as long as one public key address from each of the different clusters can be linked together. An example of this technique is illustrated in Fig. 1.4.

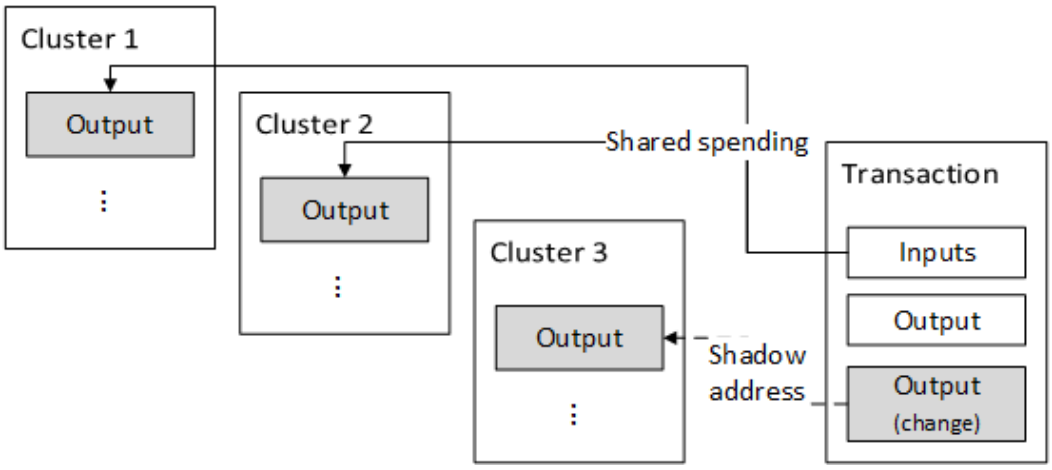

Fig. 1.4 Linking clusters transitively

1.4.3 Linking Public Key Addresses to Real World Identities

After forming clusters of public key addresses that each belongs to a user, the last step is to assign real world identities to the clusters. This can be done by transacting with users who are targets of de-anonymization [16]. Given that an adversary knows the real world identity of the user he is transacting with, he can learn one of the public key address of the user. With this one known public key address of the real world identity, a whole cluster containing that public key address can be transitively linked to the identity. Hence an adversary has the

ability to attribute a list of public key addresses to a real world identity and de-anonymize the *transactions* made by that identity.

1.4.4 Side Channels Attacks

Side channels in Bitcoin refer to the indirect disclosure of information that can aid in the discovery of the real world identity of users. Since all *transaction* information such as payment amounts and timings are publicly available, an adversary can correlate *transactions* patterns with other publicly available information to match *transactions* to real identities [20]. For example if activities on social media are correlated to Bitcoin *transactions* activity, then social media users (assuming they are using real world identities) can be linked with the *transactions* that happened during the period when they are active on social media. In addition, similar to clustering *transactions* by analysing shared spending and shadow addresses, an adversary can also cluster *transactions* together based on the similarities in the timings and amounts of the *transactions* [1]. The bottom-line is that as long as *transaction* details are public, there are always ways to make intelligent guesses about the real word identities of public key addresses.

Chapter 2

Methods to Improve Privacy in Digital Currencies

The key to achieving anonymity in Bitcoin and any digital currencies is to remove the ability to link a payer to a payee. This immediately satisfies the third property of anonymity that it should be hard to link a payer to a payee. A user can also take advantage of this and make an anonymous payment to himself. Since the public key address that he uses to make the payment cannot be linked to the public key address that he uses to receive the payment, the user can transact with the latter public key address on a clean slate. Hence, the first and second properties of anonymity that it should be hard to link public key addresses and transactions to the same user are also achieved.

2.1 Mixing

Mixing is a concept that removes the ability to link a payer to a payee. All methods to improve privacy in digital currencies adopt mixing in one way or another by. The principle of mixing is illustrated in Fig. 2.1.

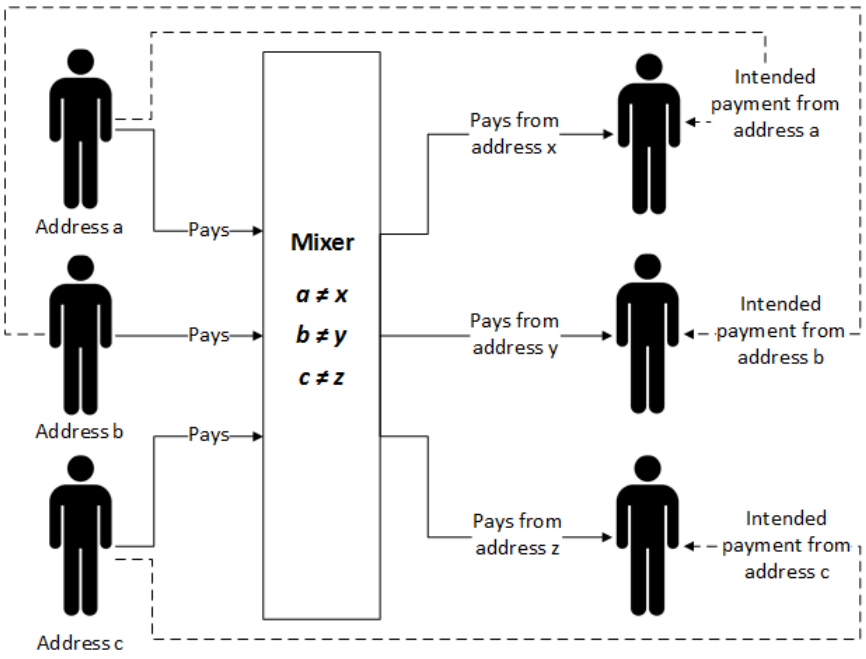

Fig. 2.1 Principle of mixing

To perform mixing, payers make payments via an intermediary called the mixer. The mixer collects payments from payers and pays their intended payee via *transactions* whose *inputs* do not refer to the public key address of the original payers. In this way observers of the Blockchain are only able to see *transactions* from the payers to the mixer and from the mixer to the payee, but cannot tell which payer paid to which payee. The anonymity set in this case is the number of payers participating in the mix, since one only knows that a payer participated in the mixing, but is unable to tell who the payer paid to among the payees that received payments from the mixer.

In order for mixing to work, the amounts that is being transacted via a mixer should be constant for all participants of the mixing protocol [20]. If different payers pay different amounts to the mixer, and subsequently the mixer pays different amounts to the payees, one can deduce the link between a payer and payee by matching the payment amounts between the payer and the mixer and those between the mixer and the payee. This research has explored several methods of mixing and the issues they face.

2.2 Dedicated Mixing Services and Decentralised Mixing

Dedicated mixing services have been commercially implemented in Bitcoin. Services such as Bitcoin Fog and BitLaundry routinely handle 6-digit dollar amounts everyday [18]. These

services act as centralised mixers, where participants make payments to the services and instruct them to transfer the funds to specific address. However dedicated mixing services face some major drawbacks:

1. Dedicated mixing services need to keep the transactions records of their users to make fund transfers. Thus privacy is built on the trust that mixing services will not disclose these records. This runs against decentralized principle of Bitcoin and digital currencies in general [20].

2. Since mixers are used for payments, users typically require funds to be transferred immediately to the intended payees. Thus mixers can only work with a limited pool of participants who happen to make payments at the same time. This limits the anonymity set in the mixing protocol, and lowers the level of anonymity in the protocol [20].

3. By exploiting the weakness in point 2 and analysing the mixing patterns adopted by the dedicated mixers, transactions in dedicated mixes can be de-anonymised [18].

Decentralised mixing adopts a peer-to-peer mixing protocol. One such protocol proposed is Coinjoin [11], which gathers payers who wish to participate in mixing to collectively construct a single transaction. Each *input* in the Coinjoin transaction refers to the funds from each payer, while each *output* in the transaction pays to the public key address of each intended payee. The *inputs* and *outputs* are ordered randomly in the transaction, and an adversary cannot tell which input (payer) corresponds to which output (payee). Here the mixer is the transaction, and the anonymity set is the number of participants in the transaction. Though Coinjoin achieves decentralisation, it still faces the same problem as dedicated mixing services of having a limited anonymity set, as the protocol is carried out with the few payers who want to make payments at the same time. In addition, since a Conjoin transaction is collectively constructed, the protocol faces denial-of-service attacks where an adversary who initially agrees to participate in a Coinjoin transaction refuses to complete his part in the creation of the transaction [20].

As seen, Bitcoin mixing techniques that involve the shuffling to payers and payees do not work well in both the centralised and decentralised form. There is also little room for improvement for these protocols as any bigger changes will involve modifications to the Bitcoin protocol that are too costly to implement. Altcoins, on the other hand, have the liberty implement any protocol and thus provide a higher level of anonymity compared to the mixing protocols.

2.3 Altcoins: Zerocoin and Zerocash

Altcoins are alternative digital currency systems to Bitcoin. The two popular proposed altcoins that provide privacy are Zerocoin and Zerocash. As opposed to the traditional mixers discussed in §2.2, Zerocoin and Zerocash provides anonymity using cryptographic guarantees. As these two protocols are very technical, they are only dealt with in concept, with a focus on their capabilities and drawbacks and the reasons why Zerocoin is chosen as the final subject of research. The Zerocoin protocol will then be examined in detail in §3. Unless otherwise stated, all technical information presented on Zerocoin and Zerocash are obtained from the original Zerocoin [17] and Zerocash [5] papers.

2.3.1 Zerocoin

Protocol

Zerocoin extends Bitcoin by incorporating another type of anonymous currency called the "Zerocoin" into the Bitcoin protocol. A comparison between Bitcoin and Zerocoin is shown in Fig. 2.2.

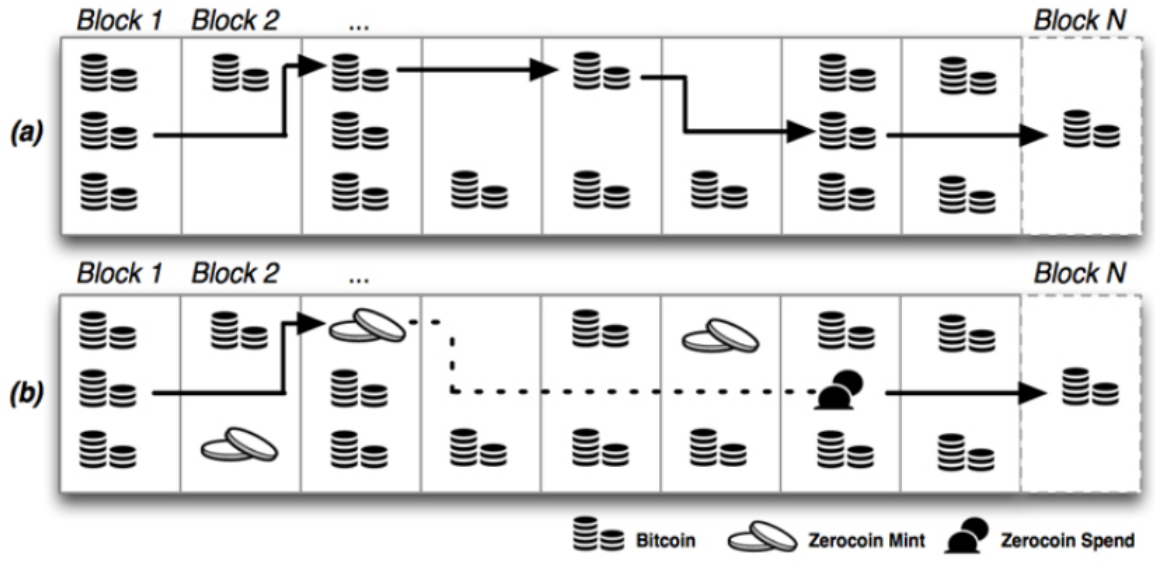

Fig. 2.2 Comparison between Bitcoin and Zerocoin [17]

Payments in Bitcoin (a) involves a transfer of funds (Bitcoins) via *transactions* recorded on the Blockchain. In any block, payments are made using Bitcoins that have been paid to the payer in a previous block (i.e. transaction *inputs* refer to *outputs* of past *transactions*).

The high level flow of Zerocoin (b) is also similar. In addition to using Bitcoins for payments, users can use their Bitcoins to create an anonymous "Zerocoin" (referred to as *coin* from now on) via a *Mint transaction*. The exchange rate between Bitcoin and *coin* is fixed, and thus a Bitcoin always mints the same amount of *coin*. A *coin* is a cryptographic commitment c of a serial number S and a secret trapdoor r which are only known to the user who minted the *coin*. The property of c is that both S and r is needed to obtain the value of c, and it is hard to compute the value of S or r from c. Like a Bitcoin *transaction*, a *Mint transaction* contains *inputs* that specify the Bitcoins that are used to mint the *coin*. A *Mint transaction* does not have an *output* as it does not make any payment and simply creates a *coin*. Once the *input* in a *Mint transaction* is verified to be valid by nodes, it is published on the Blockchain and the minted *coin* c is added to a public data structure called the **accumulator** (§3.1.2).

The minted *coin* c can be redeemed and converted to back to Bitcoins later on using a *Spend transaction*. Similar to a Bitcoin *transaction*, the *Spend transaction* contains an *input* that proves the ownership of c, and an *output* that specifies the public key address to pay the converted Bitcoins to. To prove of ownership of c in the *Spend transaction*, the redeemer must prove:

1. The knowledge of the serial number S and random trapdoor r that produced c (§3.1.3)
2. c has been previously minted and added to the accumulator (§3.1.3)

To prove the above, the *Spend transaction* only reveals S that is committed in the redeemed c and c itself is not revealed. Such proofs are called Non-Interactive Zero Knowledge (NIZK) proof which are discussed in detail in §3.1.3. Once the proof is verified the *Spend transaction* gets published on the Blockchain and the *output* in the transaction can be subsequently referred to and spent by its owner. To prevent the redeemed c from being redeemed again, S is disclosed and recorded on a separate public table. Any *Spend transactions* that spends a *coin* with a serial number that is listed on the will be rejected. A simplified example illustrating the links between normal Bitcoin *transactions*, *Mint transactions* and *Spend transactions* is shown in Fig. 2.3.

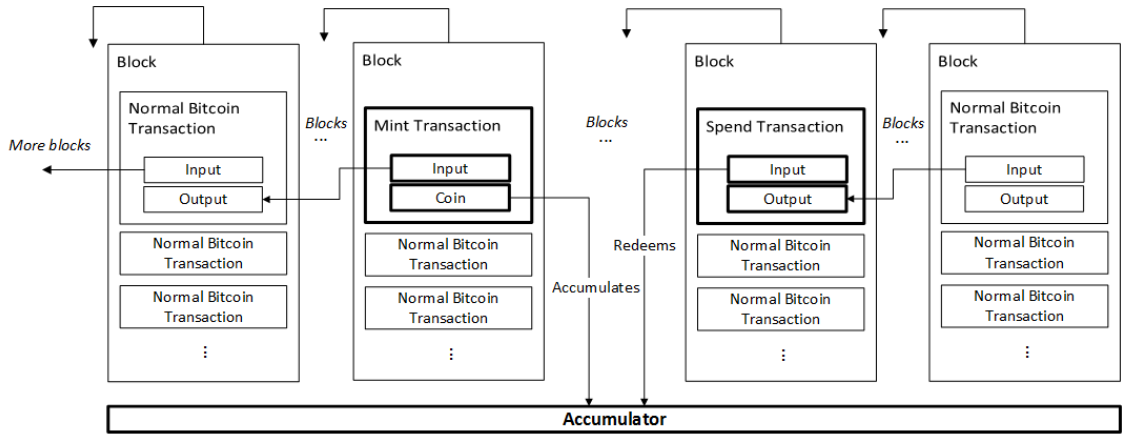

Fig. 2.3 Links between Bitcoin *transactions*, *Mint transactions* and *Spend transactions*

Zerocoin is in fact a mixing protocol, where minted *coins* act as the intermediary. When a new *Mint transactions* is published on the Blockchain, the minted *coin c* is mixed with the rest of the previously minted *coins* in the Accumulator. When a *Spend transaction* is published on the on the Blockchain, the only information available to the public is that one of all the previously minted *coins* has been redeemed to a public key address as the redeemed *coin* is not revealed. Even though the serial number *S* of the redeemed *coin* is revealed to verify that it has not been spent, the secret trapdoor *r* is private to the *coin*'s owner at all times. An adversary who only know *S* cannot determine which *c* is being redeemed and thus cannot link the redeemer of the Zerocoin (payee) to the user who minted this *coin* (payer). Hence the anonymity set in Zerocoin is all of the minted and unredeemed *coins* in history, which is much larger than the anonymity sets in the traditional Bitcoin mixing protocols (§2.2). Therefore Zerocoin achieves a much higher level of anonymity compared to these protocols.

Drawbacks

However Zerocoin is not a perfect system. Some of the drawbacks of Zerocoin are listed below:

1. *coins* can only have fixed denominations, making payments less flexible. The reason behind the fixed denominations is to ensure that adversaries find it hard to match transaction amounts between Mint and *Spend transactions* to determine links between them. This is largely similar to why transaction amounts in traditional mixing protocol must be uniform for all participants (§2.2).

2. There is no provision to transfer a *coin* to a payee directly, and actual payments still need to be done in Bitcoins. Every time a payer wants to make a payment, he needs to redeem a *coin* to Bitcoins via a *Spend transaction*, and make the payment using the Bitcoins via a normal Bitcoin *transaction*. If the payee wants to make an anonymous payment again using the received Bitcoins, he must first mint some *coins* via a *Mint transaction* and then redeem them via a *Spend transaction*. Thus *coins* need to be constantly minted and redeemed to cater for multiple anonymous transactions. This makes anonymising transactions using Zerocoin cumbersome.

3. The payment amounts, payment addresses and the timings of Zerocoin transactions are still publicly available on the Blockchain. This means that Zerocoin is still exposed to side channel attacks (§1.4.4).

4. Zerocoin *Spend transactions* are much larger in size (50KB) compared to Bitcoin *transactions* (1KB) due to the large size of the NIZK proofs that are used to prove the ownership of *coins*. This is undesirable in a peer-to-peer digital currency network. Firstly, since all nodes in the network maintains a copy of the Blockchain, large transactions sizes increase the storage requirement of nodes. Secondly, large transactions also take longer time to be propagated in the peer-to-peer network, which slows down the confirmation of transactions and the overall performance of the system.

2.3.2 Zerocash

Protocol

Zerocash is purported to be an improvement of Zerocoin. It is built on the same principles as Zerocoin, where a *Mint transaction* create some *coins* and a Pour transaction (Similar to *Spend transaction* in Zerocoin) spends a minted *coin* without disclosing which *coin* is spent. Instead of using the NIZK proofs in Zerocoin to prove ownership of *coins*, Zerocash does this by using zero-knowledge Succinct Non-interactive Arguments of Knowledge (zk-SNARKs) [6]. As zk-SNARKs is extremely complex and a relatively new field, this research is not able to describe how the proof works. However its features is sufficient to evaluate Zerocash as a protocol. The improvements in Zerocash from Zerocoin that are made possible by zk-SNARKs are listed below:

1. The size of the proofs for the ownership of *coins* in Zerocash is reduced from 45KB to 288B. As such Zerocash transactions have similar size as the standard Bitcoin *transactions* (1KB).

2. The time taken to verify a *Pour transaction* in Zerocash is reduce by 98.6

3. Zerocash transactions hide all transaction details. The only information that is publicly available in a transactions are the zk-SNARKs proofs to verify the transactions. Hence one can only observe the existence of a transaction on the Blockchain and other information such as transaction amounts and payment addresses are hidden. This makes side channel attacks (§1.4.4) impossible.

4. Zerocash supports arbitrary denominations since the transaction amounts are hidden and cannot be exploited by side channel attacks.

5. Zerocash allows minted *coins* to be directly transferred from payer to payee. There is no need to redeem a *coin* into Bitcoins before payments can be made like in Zerocoin.

Drawbacks

Though seemingly superior, Zerocash also has the following weaknesses when compared to Zerocoin:

1. The theoretical foundation of Zerocash has not be proven to be sound. zk-SNARKs is a relatively new area of research and its theories have not been used in practice as of 2015 [20]. In contrast, the NIZK proofs used by Zerocoin, which are based on RSA cryptography (§3.1.3), has been widely tested and proven for many years.

2. Zerocash requires a huge set of parameters (over 1GB) for the zk-SNARKs proofs. This places a huge storage constraint for anyone who wishes to use Zerocash, and makes deployment of Zerocash on mobile devices challenging. In contrast, Zerocoin only requires a single parameter of about 2.5KB for its NIZK proofs.

3. The construction of the zk-SNARKs proofs in Zerocash computationally expensive. Specifically, it takes 2 minutes to construct the proofs in a *Pour transaction* using an Intel Core i7 3.40GHz processor with 16GB of RAM. This means that a user of Zerocash needs to wait for 2 minutes before he/she can create a transaction on a high-end PC. This makes Zerocash impractical for mobile devices with lower processing power. In contrast, on a machine that has similar processing power as the above mentioned Intel Core i7 processor, Zerocoin only requires 0.7s to construct a *Spend transaction*.

2.3.3 Selection Zerocoin as the Subject of Research

A summary of the key features of Zerocoin and Zerocash is listed in Table 2.1.

	Zerocoin	**Zerocash**
Technology	RSA cryptography. Mature and proven in practice.	zk-SNARKs. New and not implemented in practice.
Level of anonymity	Payment amount, timing and addresses are public. Susceptible to side channel attack.	Only transaction timing is public, hides all other transaction information. Side channel attack is not possible.
Denomination	Fixed	Arbitrary
Direct payment using *coins*	No	Yes
Size of parameters	2.5KB	1GB
Size of proof	45KB	288B
Time taken to construct the proofs in a *Spend/Pour transaction* on a high end processor	2 minutes	0.7s
Time taken to verify the proofs in a *Spend/Pour transaction* on a high end processor	300ms	5.7ms

Table 2.1 Key features of Zerocoin and Zerocash

Zerocash definitely has much more to offer in terms of privacy and features. However Zerocash requires large storage and a long time to construct transactions even on high-end processors, and thus may face challenges running on clients that increasingly consist of mobile devices. Though the transaction sizes of Zerocoin are relatively large and may impose higher load on the peer-to-peer network, the overall storage and processing requirements of Zerocoin for both the clients and the network is still moderate. Slight improvements in Zerocoin's performance can make it a very practical protocol.

In addition, the technology behind Zerocash is also not mature. Literature on zk-SNARKs is both sparse and complex, while literature on NIZK proofs are readily available and well documented [27]. This makes research on Zerocoin more productive compared to Zerocash. Moreover the theoretical basis for Zerocash is uncertain due to the lack of implementation of zk-SNARKs in real-life applications, while Zerocoin is theoretically reliable as it uses NIZK proofs which are tested and proven.

Most importantly, the full anonymity provided by Zerocash may be undesirable. This is because full anonymity removes a key feature of the Blockchain, which is that the peer-to-peer network can collectively audit the public transactions. If payment amounts in transactions are hidden, any bugs or attacks that cause *coins* to be wrongly created will go unnoticed on the Blockchain. This can cause hyperinflation of Zerocash and render the system obsolete [27]. Thus as promising as it seems, Zerocash has its caveats and its implications as a real-life digital currency system are unknown.

Taking into account the above considerations, Zerocoin is a more attractive area of research due to its performance, the accessibility of literature and its potential application. Hence the rest of this research devotes to analysing Zerocoin in detail and finding ways to improve its performance, so that it can be a practical anonymous digital currency system.

Chapter 3

Analysis of Zerocoin

This chapter delves into the data structures and algorithms of Zerocoin and analyses the reasons behind some of its key weaknesses. A basic understanding of Zerocoin and the issues of privacy detailed in Chapter 1 and 2 are needed to appreciate the rest of this chapter. Unless otherwise stated, all technical information on Zerocoin is obtained from the original Zerocoin paper [17].

3.1 Construction of Zerocoin

3.1.1 Coin as a Pederson Commitment

The key element in Zerocoin is the *coin* that is minted and redeemed in Mint and *Spend transactions*. A *coin c* is a Pedersen commitment [23] of a serial number $S \in \mathbb{Z}_{q_{comm}}$ and a random trapdoor $r \in \mathbb{Z}_{q_{comm}}$ in the form $c = g_{comm}^{S} h_{comm}^{r} \bmod p_{comm}$. p_{comm} and q_{comm} are large primes and g_{comm} and h_{comm} are generators of the sub-group of $\mathbb{Z}_{q_{comm}}$ multiplicative group $\mathbb{Z}_{p_{comm}}^{*}$. That is to say $g_{comm}^{0}, g_{comm}^{1}, ..., g_{comm}^{q_{comm}} \bmod p_{comm}$ and $h_{comm}^{0}, h_{comm}^{1}, ..., h_{comm}^{q_{comm}} \bmod p_{comm}$ produce two sequences of distinct numbers. To satisfy this property, $g_{comm}^{q_{comm}} \bmod p_{comm} = 1$ and $h_{comm}^{q_{comm}} \bmod p_{comm} = 1$ and thus q_{comm} divides $p_{comm} - 1$ by Fermat's Little Theorem. In Zerocoin, p_{comm} and q_{comm} are prime numbers of 1024 and 256 bits respectively [1], and g_{comm} and h_{comm} are computed based using the algorithm defined in the Federal Information Processing Standards (FIPS) 186-34 [13]. For simplicity of presentation, the Pedersen commitment will be written as $c = g_{comm}^{S} h_{comm}^{r}$, where modulus p_{comm} is implied for generators g_{comm} and h_{comm}.

In Zerocoin, a single set of $g_{comm}, h_{comm}, p_{comm}, q_{comm}$ is generated once as public parameters during set up, and is subsequently used by all users. To mint a *coin*, a user generates

[1] Based on the security recommendations in Section VI(B) of the Zerocoin paper [17].

a random serial number $S \in \mathbb{Z}_{q_{comm}}$ and another random trapdoor $r \in \mathbb{Z}_{q_{comm}}$, computes a Pedersen commitment $c = g_{comm}^{S} h_{comm}^{r}$ and include c in the *Mint transaction*. S is kept secret when the *Mint transaction* is created and is subsequently revealed in the *Spend transaction* to prove the ownership of c (§3.1.3), while r is always secret to the creator of the *Mint transaction*.

After the *Mint transaction* is verified and published on the Blockchain, everyone can see that the creator of the transaction has minted c. Theoretically, an adversary can "steal" a *coin* by obtaining S and r from the public c and creating a valid *Spend transaction* using S and r. However the adversary can only succeed with negligible probability in practice as solving S and r from a known c where $c = g_{comm}^{S} h_{comm}^{r}$ is believed to be hard under the Discrete Logarithm assumption [22]. Due to this property, it is also hard to solve c from S without the knowledge of r. Since r is secret to the owner of the *coin* at all times, an adversary is also unable to attribute the S in the *Spend transactions* to a c in the *Mint transactions* on the Blockchain. This preserves the anonymity property that the redeemer (payee) of a *coin* cannot be linked to its creator (payer).

3.1.2 Public Accumulator

An accumulator "summarises" a large number of values into one value, and given a short witness a specific value can be proven to be accumulated in the accumulator. The accumulator in Zerocoin is a public data structure that contains every minted *coin* in history. As such, given a witness any nodes in the peer-to-peer network can verify that a *coin* has been minted using the accumulator. The size of the accumulator remains constant as more *coins* are accumulated, thus it can be distributed and stored efficiently by the peer-to-peer network. The accumulator used in Zerocoin is a dynamic accumulator proposed by Camenisch and Lysyanskaya [14] that extends the study by Baric and Pfitzmann [3]. The 3 main functions provided by this accumulator scheme are explained below.

$Accumulate((N,u),C) \rightarrow A$ accumulates a set of *coins* $C = \{c_1, \cdots, c_n \mid c_i \in [A,B] \wedge c_i \, is \, prime\}$ using parameters (N,u) to produce an accumulator $A = u^{c_1,\cdots,c_n} \bmod N$. Like the parameters used in the Pedersen commitment for *coins* (§3.1.1), (N,u) are public parameters that are generated once during set up. N is a product of 2 large primes while u is a quadratic residue modulo N [2]. N is recommended is to be 3072 bits long [3]. $\mathfrak{A}$ and $\mathfrak{B}$ [4] defines the range of the values that an accumulated *coin* c can take. In addition, c must be prime before

[2] A quadratic residue modulo N is the square of any integer modulo N.

[3] Based on the security recommendations in Section VI(B) of the Zerocoin paper [17].

[4] $\mathfrak{A} = -p_{comm} 2^{k'+k''+2}, \mathfrak{B} = p_{comm} 2^{k'+k''+2}$. k' and k'' are set at 160 and 128 respectively. The reasons behind these values can be found in [17] and [14].

it can be accumulated. Hence when minting a new *coin*, different S and r must be tried until $c = g_{comm}^{S} h_{comm}^{r}$ satisfies the above constraints.

Every time a new *coin* is successfully minted, nodes will accumulate the new *coin* into the public accumulator. Due to the construction of the dynamic accumulator, a new *coin* c_{new} can be incrementally accumulated [14] into the previous accumulator A using $Accumulate((N,A),\{c_{new}\}) \rightarrow A$. Thus when updating the accumulator, one do not need to recompute $u^{c_1,\cdots,c_n c_{new}} \bmod N$ but simply compute $A^{c_{new}} \bmod N$ using the latest A, making accumulation efficient. Zerocoin proposes that each *block* contains an accumulator called the accumulator checkpoint that contains all the *coins* minted up till that *block*. This way nodes do not need to compute the accumulator from scratch, and can choose an accumulator checkpoint from any *block* and incrementally accumulate the *coins* minted after that *block* to obtain the latest accumulator value.

For a *coin* $c \in C$, $GenWtiness((N,u),c,C) \rightarrow w$ produces a witness $w = Accumulate((N, u), C \setminus \{c\}) \rightarrow A$. That is, *is* an accumulation of all the *coins* in C except c. In Zerocoin, C is the set of all minted *coins* in history. With w, one can verify that that c is in C using $AccVerify((N,u),A,c,w) \rightarrow \{0,1\}$, where $A = Accumulate((N,u),C) \rightarrow A$. $AccVerify$ returns 1 if and only if $A = w^c \bmod N$ and c is a valid *coin* (c is prime and $c \in [\mathfrak{A},\mathfrak{B}]$). It can be seen that $A = w^c \bmod N$ is the correct verification equation as $A = Accumulate((N,u), C) \rightarrow A = u^{c_1,\cdots,c_n c} \bmod N = (u^{c_1,\cdots,c_n})^c \bmod N = w^c \bmod N$. To ensure security, A should be independently computed by each verifier, which can be achieved efficiently using the incremental method described in the previous paragraph.

In Zerocoin, *GenWitness* is used by the creator of a *Spend transaction* to create a witness to prove that the redeemed *coin* is in the accumulator. $AccVerify$ is used by nodes in the Zerocoin network to verify that claim when validating the *Spend transaction*. However $AccVerify$ is used only in principle and not directly, because w and c are not revealed to the verifier in a *Spend transaction* in order to keep c unkown. Why and how the *Spend transaction* is verified without the knowledge of w and c is discussed in §3.1.3.

3.1.3 Non-Interactive Zero Knowledge (NIZK) Proofs

Interactive Zero Knowledge Proofs

Zerocoin relies heavily on NIZK proofs, which are a subset of zero-knowledge (ZK) proofs. ZK proof is a proof of the knowledge of an element without disclosing any information beyond the legitimacy of the proof [15] The formal notation of ZK proof introduced by Camenisch and Stadler [8] is as follow:

$$ZKPoK\{(w) : statement\ proving\ knowledge\ of\ w\}$$

ZKPoK refers to Zero-Knowledge Proof of Knowledge. The proof statement is enclosed in the curly brackets. The element in parenthesis is the information that the prover wishes to show knowledge of, but cannot be disclosed. This notation will be used for all zero knowledge proofs in this paper.

ZK proofs often surrounds an NP-hard problem. In general, a prover wants to prove that he has a secret solution w to the NP-hard problem y. The prover first sends a random problem t that has a solution v that is unrelated to w. The verifier then sends a random challenge $\mathscr{C}$ to the prover, and the prover replies with a response s that looks random but in fact is a combination of v, w and $\mathscr{C}$. The verifier verifies the proof by composing s with $\mathscr{C}$ and the original problem y. s is constructed in a way such that if the prover knows w, it will cancel out y, leaving behind only the random problem t. Hence by checking that the result of the proof is t the verifier is able to verify the proof, but does not learn anything about w since all he sees are random values. The above is a form of interactive ZK proof which involves the verifier communicating the challenge $\mathscr{C}$ to the prover. A simple example of an interactive ZK proof for the knowledge of discrete logarithm can be found online [25]. This proof is chosen as an example as it is closely related to the proofs used in the Zerocoin protocol.

Transforming Interactive Zero Knowledge Proofs to NIZK Proofs

NIZK proof is a special kind of ZK proof that removes the need for the verifier to interact with the prover. The prover can achieve this by a generating a challenge $\mathscr{C}$ from y and t using a random oracle [5], and treat this $\mathscr{C}$ as the challenge that the verifier sends in the interactive version the ZK proof. The prover then computes s based on the generated $\mathscr{C}$ and sends the proof containing y, t and s to the verifier. Upon receiving the proof, the verifier recomputes $\mathscr{C}$ from the received y and t using the random oracle, and verifies the proof in the same way as the interactive version of the ZK proof. Since there is no interaction between the prover and the verifier, the NIZK is achieved.

The Fiat-Shamir heuristics [10] is a popular technique to transform interactive ZK proofs into NIZK proofs. Using the Fiat-Shamir heuristics, the random oracle is implemented as a cryptographic hash function [4], and $\mathscr{C}$ is obtained by hashing y and t together with and other parameters. The NIZK proofs used in Zerocoin and the proofs proposed by this research all use the Fiat-Shamir heuristics in one way or another. The same online resource that illustrates

[5] A theoretical black box that responds to every unique query with a truly random response chosen uniformly from its output domain. If a query is repeated it responds the same way every time that query is submitted. [26]

interactive ZK proof for the knowledge of discrete logarithm also shows how the proof can be modified into a NIZK form using the Fiat-Shamir heuristics [25].

Properties of Zero-Knowledge Proofs

All ZK proofs (NIZK proofs included) need to fulfil the below properties [15]:

1. ***Completeness***: If the prover indeed knows the secret w, and both the prover and verifier follow the procedures of the proof, then the proof will always be verified.

2. ***Soundness***: If the prover does not know the secret w, and both the prover and verifier follow the procedures of the proof, then the proof will not be verified. This property can be proven if a hypothetical program called the knowledge extractor that can extract w from the prover using two accepting proofs that uses the same t but different $\mathscr{C}$ and s. In order to do this, the knowledge extractor has access to the prover's program (but not the w itself) and can "rewind" the proof to obtain another set of $\mathscr{C}$ and s. The success of the knowledge extractor is needed to prove for soundness because if a prover who does not know w manages to prove the knowledge of w, there is no way that a knowledge extractor can extract a w that is non-existent in the proof.

3. ***Statistical Zero-Knowledge***: The verifier does not gain any knowledge after interacting with the prover. This can be proven if a hypothetical program called a simulator with no knowledge of the secret w, is able to generate a response s' to the challenge $\mathscr{C}$, such that s' is statistically indistinguishable from the actual response s generated by the prover. In this way the verifier does not learn anything more from the s sent by the prover than from the s sent by the simulator.

NIZK Proofs in Zerocoin

NIZK proofs are used in Zerocoin to prove the ownership of a valid unredeemed *coin* in a *Spend transaction*. The proof needs to be zero-knowledge so that the verifier or anyone who sees the *Spend transaction* does not know which *coin* is being spent. The proof also needs to be non-interactive because a *Spend transaction* can be verified by any node in the Zerocoin network at any time, and it is not possible for a verifier to interact with the creator of the transaction. The three NIZK proofs in a *Spend transaction* are:

1. ***Accumulator Proof of Knowledge (AccPoK)***: Proves that a *coin* c has been minted, without disclosing c. This is achieved by proving that c is contained in the public accumulator in which every *coin* that has been minted should have been included in. Details of this proof is discussed in §3.1.3.

2. ***Serial Number Signature Proof of Knowledge (SNSoK)***: Proves the knowledge of the serial number S and the secret trapdoor r that are committed in the redeemed *coin* c, without disclosing r and c. Since only the owner of a *coin* knows its S and r, this proof shows that the prover owns the *coin* being proven for. This proof also acts as a digital signature for the *Spend transaction*. A digital signature is needed to prevent the contents of the *Spend transaction*, which contains the public key address that the redeemed *coin* should be paid to, from being tampered with. Details of this proof is discussed in §3.1.3.

3. ***Commitment Proof of Knowledge (CommPoK)***: Proves that the *coin* being proven for in the accumulator Proof of Knowledge and the *coin* being proven for in the Serial Number Signature of Knowledge are the same *coin*. This ensures the *coin* which the prover is proving to be minted is the same *coin* that he is proving ownership for. With this proof, one cannot redeem a minted *coin* that he does not own. Details of this proof is discussed in §3.1.3.

Accumulator Proof of Knowledge

For a *coin* to be spent, it must first be proven that the *coin* has been minted. The Accumulator Proof of Knowledge (AccPoK) proves this claim using the public accumulator which contains all the minted *coins* in history.

As seen in §3.1.2, the claim that a *coin* c is in the set of minted *coins* can be verified by presenting $AccVerify$ with the witness w. However this cannot be done directly in Zerocoin as c needs to be secret in order to preserve unlinkability between the minter and the redeemer of c. In addition, w also needs to be secret as it can be used to obtain c. This is possible because all the minted *coins* in history are publicly available in the *Mint transactions* on the Blockchain. An adversary can run $GenWtiness((N,u),c',C) \rightarrow w'$ repeatedly using all $c' \in C$, where C is all the minted *coins* in history, and the c' that produced a $w = w'$ is the *coin* that is being proven for. Hence to hide c and w, the AccPoK is as such:

$$NIZKPoK\{(c,w) : AccVerify((N,u),A,c,w) = 1 \wedge c \in [\mathfrak{A},\mathfrak{B}]\}$$

The implementation of the AccPoK is adapted from an interactive version of the proof by Camenisch and Lysyanskaya [14] and modified to a non-interactive form using the Fiat-Shamir heuristics. The rest of the section describes the implementation details.

In order to hide c and w, the prover creates the following commitments of c and w using random trapdoors:

1. $\mathfrak{C}_c$, a commitment of c with a random trapdoor $\varphi \in \mathbb{Z}_{\lfloor N/4 \rfloor}$ such that $\mathfrak{C}_c = \mathfrak{g}^c \mathfrak{h}^\varphi \bmod \mathfrak{p}$. $\mathfrak{g}$ and $\mathfrak{h}$ are generators for sub-group $\mathbb{Z}_\mathfrak{q}$ of multiplicative group $\mathbb{Z}^*_\mathfrak{p}$, where $\mathfrak{q} > 2\mathfrak{B}$.

2. C_c, another commitment of c with a random trapdoor $\eta \in \mathbb{Z}_{\lfloor N/4 \rfloor}$ such that $C_c = g_{QRN}^c h_{QRN}^\eta \bmod N$. g_{QRN} and h_{QRN} are quadratic residues modulo N.

3. C_w, a commitment of the witness w with a random trapdoor $\varepsilon \in \mathbb{Z}_{\lfloor N/4 \rfloor}$ such that $C_w = w h_{QRN}^\varepsilon \bmod N$.

4. C_r, a commitment of ε with a random trapdoor $\zeta \in \mathbb{Z}_{\lfloor N/4 \rfloor}$ such that $C_r = g_{QRN}^\varepsilon h_{QRN}^\zeta \bmod N$.

Like the parameters used in the accumulator (§3.1.2), $\mathfrak{g}, \mathfrak{h}, g_{QRN}, h_{QRN}, \mathfrak{p}, \mathfrak{q}$ are public parameters that are generated once during set up. For simplicity of presentation, the modulus of the above commitments are omitted and implied. With these commitments, the prover implements the AccPoK by constructing a NIZK proof that shows that $\mathfrak{C}_c$ and the accumulator A contains the same c in the following form:

$$NIZKPoK(c, \beta, \gamma, \delta, \varepsilon, \zeta, \varphi, \psi, \eta, \sigma, \xi) : \{$$

$$\mathfrak{C}_c = \mathfrak{g}^c \mathfrak{h}^\varphi \wedge \mathfrak{g} = (\mathfrak{C}_c / \mathfrak{g})^\gamma \mathfrak{h}^\psi \wedge \mathfrak{g} = (\mathfrak{g}\mathfrak{C}_c)^\sigma \mathfrak{h}^\xi \wedge$$

$$C_r = g_{QRN}^\varepsilon h_{QRN}^\zeta \wedge C_c = g_{QRN}^c h_{QRN}^\eta \wedge A = C_w^c (1/h_{QRN})^\beta \wedge$$

$$1 = C_r^c (1/h_{QRN})^\delta (1/g_{QRN})^\beta \wedge c \in [\mathfrak{A}, \mathfrak{B}]\}$$

The proof follows the standard flow of a NIZK proof described in §3.1.3. Since there are 8 sub proofs in the NIZK proof, the prover first commits some random values v_i to create commitments $\mathfrak{C}_c, C_c, C_w, C_r$ that are not related to,for each of the sub-proof. Following the principle of the Fiat-Shamir heuristics, the prover computes the challenge string $\mathscr{C} = H(\mathfrak{C}_c \| C_c \| C_w \| C_r \| \mathfrak{g} \| \mathfrak{h} \| g_{QRN} \| h_{QRN} \| t_i \cdots)$ [6] and further computes s_i that correspond to each t_i using $\mathscr{C}$. The final $\mathfrak{C}_c, C_c, C_w, C_r, t_i, s_i$ are written into the *Spend transaction* and broadcasted to the Zerocoin network to be verified. Upon receiving the *Spend transaction*, the nodes re-computes $\mathscr{C} = H(\mathfrak{C}_c \| C_c \| C_w \| C_r \| \mathfrak{g} \| \mathfrak{h} \| g_{QRN} \| h_{QRN} \| t_i \cdots)$ using the received $\mathfrak{C}_c, C_c, C_w, C_r, t_i, s_i$ and the public parameters $\mathfrak{g}, \mathfrak{h}, g_{QRN}, h_{QRN},$. The verifier then verifies the proof by checking if each received t_i equals to the t_i' computed from $\mathscr{C}$ and s_i using some

[6] H denotes the SHA-256 hash function used by Zerocoin. $\|$ denotes a concatenation operation

verification formulas. The verification formulas are detailed in Appendix A of [14]. The completeness, soundness and statistical zero-knowledge properties of the proof are also detailed in the §3.3 of the same study.

Serial Number Signature of Knowledge

Proving that a *coin* has been accumulated in the public accumulator only shows that the prover knows some *coin* has been minted, but does not legitimise his ownership of the *coin*. Anyone who looks at the Blockchain can simply take a *coin* from any past *Mint transactions*, obtain a witness for the *coin* using *GenWitness* and construct a valid AccPoK for that *coin*. Hence, the creator of a *Spend transaction* must also prove that he owns the redeemed *coin*. This is done using the Serial Number Signature of Knowledge (SNSoK) where the prover proves that he knows the serial number S and the random trapdoor r that are committed a *coin* c without disclosing r and c. Since r is secret to the owner of c at all times, only the owner of the c can produce a valid proof. This proof is a "signature" because it binds to the other contents of the *Spend transaction* such as the public key address of the payee. As the SNSoK becomes invalid once the contents of the *Spend transaction* changes, adversaries cannot tamper with the *Spend transaction*. Given the above objectives and constraints, the SNSoK in Zerocoin is as such:

$$ZKSoK[m]\{(c,r) : c = g_{comm}^{S} h_{comm}^{r}\}$$

The m enclosed in the square brackets denotes the contents in the *Spend transaction* to be signed. As the serial number S of the redeemed c is disclosed in the proof, the trapdoor r must be kept secret to prevent c from being computed by observers of the Blockchain. The implementation of the SNSoK is adapted from the proof by Camenisch [21].

In order to hide c, the prover creates y, a Pedersen commitment of c using a random trapdoor $z \in \mathbb{Z}_{q_{SoK}}$, such that $y = g_{SoK}^{c} h_{SoK}^{z} \bmod p_{SoK} = g_{SoK}^{g_{comm}^{S} h_{comm}^{r}} h_{SoK}^{z} \bmod p_{SoK}$. As per Pedersen commitment scheme, g_{SoK} and h_{SoK} are generators of the sub-group $\mathbb{Z}_{q_{SoK}}$ of multiplicative group $\mathbb{Z}_{p_{SoK}}^{*}$, and $q_{SoK} | p_{SoK} - 1$. As the exponent in a Pederson commitment must be an element of $\mathbb{Z}_q$ [23], $c \in \mathbb{Z}_{q_{SoK}}$ and since c is a value modulo p_{comm} (§3.1.1), it is required that $q_{SoK} = p_{comm}$. Like the parameters used in the Pedersen commitment for *coins* (§3.1.1), $g_{SoK}, h_{SoK}, p_{SoK}, q_{SoK}$ are public parameters that are generated once during set up. For simplicity of presentation, the modulus p_{SoK} is omitted and implied for g_{SoK} and h_{SoK}. With y, the prover implements the SNSoK by constructing a NIZK proof that shows that y contains a c that is a commitment to the serial number S and trapdoor r in the following form:

$$ZKSoK[m]\{(c,r,z) : y = g_{SoK}^{g_{comm}^{S} h_{comm}^{r}} h_{SoK}^{z}\}$$

The flow of the NIZK proof is largely similar to the standard NIZK proof under the Fiat-Shamir heuristics described in §3.1.3, but has a slightly different way of computing and using the challenge $\mathscr{C}$. The prover first creates t_i for i from 0 to λ_{zkp} using random numbers $u_i \in \mathbb{Z}_{q_{comm}}$ and $v_i \in \mathbb{Z}_{q_{SoK}}$ such that $t_i = g_{SoK}^{g_{comm}^{S} h_{comm}^{u_i}} h_{SoK}^{v_i}$. To ensure security, the Zerocoin paper recommends that $\lambda_{zkp} = 80$ [7]. Following the principle of the Fiat-Shamir heuristics, the prover then computes the challenge string $\mathscr{C} = H(y\|S\|g_{SoK}\|h_{SoK}\|t_1\|\cdots\|t_{\lambda_{zkp}}\|m)$. The inclusion of m in the hash $\mathscr{C}$ ensures that the contents of the *Spend transaction* is bound to the challenge string. Any tampering with m will cause $\mathscr{C}$ to be invalid and the proof fail. Thus $\mathscr{C}$ also serves as the signature for m. Using $\mathscr{C}$ the prover then computes s_i and s'_i in the following manner:

---

for $i \leftarrow 1, \lambda_{zkp}$ **do**
  if i^{th} bit in $\mathscr{C} = 0$ **then**
    $s_i = (u_i - r)$; $s'_i = (v_i - z h_{comm}^{u_i - r})$
  else
    $s_i = u_i$; $s'_i = v_i$

The prover writes $y, \mathscr{C}, t_1, \cdots, t_{\lambda_{zkp}}, s_1, \cdots, s_{\lambda_{zkp}}, s'_1, \cdots, s'_{\lambda_{zkp}}$ and the transaction content m into the *Spend transaction* and broadcast it to the Zerocoin network to be verified. Upon receiving the *Spend transaction*, the nodes computes $\mathscr{C}' = H(y\|S\|g_{SoK}\|h_{SoK}\|t'_1\|\cdots\|t'_{\lambda_{zkp}}\|m)$ where:

---

for $i \leftarrow 1, \lambda_{zkp}$ **do**
  if i^{th} bit in $\mathscr{C}' = 0$ **then**
    $t'_i = y^{h_{comm}^{s_i}} h_{SoK}^{s'_i}$
  else
    $t_i = g_{SoK}^{g_{comm}^{S} h_{comm}^{s_i}} h_{SoK}^{s'_i}$

The proof is verified if and only if $\mathscr{C}'$ equals to the $\mathscr{C}$ received. In order for $\mathscr{C}' = \mathscr{C}$, it must be the case that $t_i = t'_i$ for i from 1 to λ_{zkp}. In addition, the m used to compute $\mathscr{C}'$ must also be the original m used to compute $\mathscr{C}$, and thus $\mathscr{C}$ acts as the signature that preserves the integrity of m. The details of the SNSoK are taken from Appendix B of the Zerocoin paper. The completeness, soundness and statistical zero-knowledge properties of the proof are also detailed in the same paper.

[7] Based on the security recommendations in Section VI(B) of the Zerocoin paper [17].

Commitment Proof of Knowledge

The AccPoK and SNSoK independently proves a certain property of a *coin*. However with only these two proofs, nothing is stopping one from constructing an AccPoK and a SNSoK for two different *coins*. A dishonest user can exploit this by constructing a valid SNSoK for an arbitrary *coin* using an arbitrary serial number and secret trapdoor, and a valid AccPoK for any minted *coin* on the Blockchain (§3.1.3). Since the *coins* in the two proofs are not known to the verifying nodes, the dishonest user is able to spend the arbitrary *coin*, and claim that it is the minted *coin* without being found out. This is analogous to the ability to printing money in real life. Hence there must be some way to ensure that the *coins* being proven for in the AccPoK and SNSoK are the same *coin*. This is achieved using the Commitment Proof of Knowledge (CommPoK), where the prover proves that that commitment of the *coin* $\mathfrak{C}_c = \mathfrak{g}^c \mathfrak{h}^{\varphi}$ used in the AccPoK contains the same c as the commitment of the *coin* $y = g_{SoK}^c h_{SoK}^z$ used in the SNSoK. Hence the CommPoK is implemented in the following form:

$$NIZKPoK\{(c, \varphi, z) : \mathfrak{C}_c = \mathfrak{g}^c \mathfrak{h}^{\varphi} \wedge y = g_{SoK}^c h_{SoK}^z\}$$

The CommPoK is an extension of the proof by Camenisch to prove for the equality of two secret keys [21]. The proof follows the standard flow of a NIZK proof described in §3.1.3. The prover first creates commitments $t_1 = \mathfrak{g}^{v_1} \mathfrak{h}^{v_2}$ and $t_2 = g_{SoK}^{v_1} h_{SoK}^{v_3}$ using random numbers v_1, v_2, v_3. Following the principle of the Fiat-Shamir heuristics, the prover computes the challenge string $\mathscr{C} = H(\mathfrak{C}_c \| y \| t_1 \| t_2 \| \mathfrak{g} \| \mathfrak{h} \| g_{SoK} \| h_{SoK})$ and further computes $s_1 = v_1 + c\mathscr{C}$, $s_2 = v_2 + \varphi\mathscr{C}$ and $s_3 = v_3 + z\mathscr{C}$. The prover then includes $\mathfrak{C}_c, y, t_1, t_2, s_1, s_2, s_3$ in the *Spend transaction* together with the other proofs and broadcasts the transaction to the Zerocoin network. Upon receiving the proof in the *Spend transaction*, the verifying nodes re-compute $\mathscr{C} = H(\mathfrak{C}_c \| y \| t_1 \| t_2 \| \mathfrak{g} \| \mathfrak{h} \| g_{SoK} \| h_{SoK})$ using the received $\mathfrak{C}_c, y, t_1, t_2$ and the public parameters $\mathfrak{g}, \mathfrak{h}, g_{SoK}, h_{SoK}$. The verifier then verifies the proof by checking if both equalities $t_1 = \mathfrak{g}^{s_1} \mathfrak{h}^{s_2} \mathfrak{C}_c^{-\mathscr{C}}$ and $t_2 = g_{SoK}^{s_1} h_{SoK}^{s3} y^{-\mathscr{C}}$ are true. The completeness, soundness and statistical zero-knowledge properties of the proof are detailed in the study by Camenisch [21].

Public Parameters

A summary of the public parameters used in the Zerocoin protocol is listed in Table 3.1. These parameters only need to be generated once during the setup of the Zerocoin system, and are subsequently used by all nodes for creating and verifying transactions.

	Public Parameter	Recommended value/size	Properties
Coin commitment	p_{comm}	1024 bits	$q_{comm} \mid p_{comm} - 1$
	q_{comm}	256 bits	$q_{comm} \mid p_{comm} - 1$
	g_{comm}	N.A.	Generator for sub-group $\mathbb{Z}_{q_{comm}}$ of multiplicative group $\mathbb{Z}^*_{p_{comm}}$
	h_{comm}	N.A.	Generator for sub-group $\mathbb{Z}_{q_{comm}}$ of multiplicative group $\mathbb{Z}^*_{p_{comm}}$
Public Accumulator	N	3072 bits	Product of two large primes
	$[\mathfrak{A}, \mathfrak{B}]$	N.A.	$[-p_{comm}2^{k'+k''+2}, p_{comm}2^{k'+k''+2}]$
	k	160	
	k'	128	
AccPoK/CommPoK	$\mathfrak{p}$	N.A.	$\mathfrak{q} \mid \mathfrak{p} - 1$
	$\mathfrak{q}$	$> 2\mathfrak{B}$	$\mathfrak{q} \mid \mathfrak{p} - 1$
	$\mathfrak{g}$	N.A.	Generator for sub-group $\mathbb{Z}_{\mathfrak{q}}$ of multiplicative group $\mathbb{Z}^*_{\mathfrak{p}}$
	$\mathfrak{h}$	N.A.	Generator for sub-group $\mathbb{Z}_{\mathfrak{q}}$ of multiplicative group $\mathbb{Z}^*_{\mathfrak{p}}$
	g_{QRN}	N.A.	Quadratic residue modulo N
	h_{QRN}	N.A.	Quadratic residue modulo N
SNSoK/CommPoK	λ_{zkp}	80	
	p_{SoK}	N.A.	$q_{SoK} \mid p_{SoK} - 1$
	q_{SoK}	N.A.	Equals p_{comm}
	g_{SoK}	N.A.	Generator for sub-group $\mathbb{Z}_{q_{SoK}}$ of multiplicative group $\mathbb{Z}^*_{p_{SoK}}$
	h_{SoK}	N.A.	Generator for sub-group $\mathbb{Z}_{\mathfrak{q}}$ of multiplicative group $\mathbb{Z}^*_{\mathfrak{p}}$

Table 3.1 Public parameters used in Zerocoin

3.2 Zerocoin Implementation: libzerocoin

The main protocols in Zerocoin have been implemented in C++ by the Zerocoin authors and the source code is publicly available on Github under the name of libzerocoin [8]. libzerocoin is not a full-fledged digital currency system. Instead, it is a library that provides the functions of setting up Zerocoin public parameters (§3.1.3), creating (§3.1.1) and accumulating *coins*

[8]Code repository can be accessed at https://github.com/Zerocoin/libzerocoin

(§3.1.2) and constructing and verifying the various zero-knowledge proofs (§3.1.3). Since libzerocoin has implemented all of the key elements of Zerocoin, it can be used to analyse the performance of the Zerocoin in isolation. More importantly, libzerocoin provides a comprehensive set of functions to generate protocol parameters and perform cryptography-related computations such as SHA-256 and modulo arithmetic on large numbers. This allows the high level protocol of Zerocoin to be modified without the need to meddle with the details, and makes implementing improvements convenient. Hence this research builds on the code base of libzerocoin to conduct tests and make potential improvements.

3.3 Performance of Zerocoin and Areas for Improvement

libzerocoin has been modified to test the computational and storage requirements of the key data structures and algorithms in Zerocoin. The tests are conducted using the public parameter values and sizes recommended by the Zerocoin paper (Table 3.1). The machine used to run the tests is a ____. The results of the tests are shown in Table 3.2.

	Average over 50 iterations	
	Time taken	**Size of output**
Generation of public parameters	849ms	2578 bytes
Creation of one *coin*	324ms	199 bytes
Accumulation of one *coin*	39ms	391 bytes (accumulator size)
Construction of one AccPoK	146ms	**6,974 bytes**
Construction of one SNSoK	146ms	**17,420 bytes**
Construction of one CommPoK	5ms	714 bytes
Verification of one AccPoK	**100ms**	N.A.
Verification of one SNSoK	**163ms**	N.A.
Verification of one CommPoK	4ms	N.A.

Table 3.2 Computational and storage requirements of Zerocoin

The results highlighted in bold are identified as the performance weaknesses of Zerocoin which this research aims to address. With reference to Table 3.2, the next two sections elaborate on why these weaknesses are identified.

Computational Performance

The computations that affect the performance of Zerocoin the most are those which need to be carried out by the Zerocoin peer-to-peer network. For the generation of public parameters, the

Zerocoin network does no incur any computational costs since the parameters are generated once by the trusted party that sets up the system. Thus the cost of generating public parameters is not considered a performance issue. The creation of *coins* and the construction of the NIZK proofs are carried out by Zerocoin users whenever they want to make a transaction. Even though the time taken for these operations is in the order of hundreds of milliseconds, the computation requirement is contained within the individual users. As such *coin* creation and proofs construction do not impose on the peer-to-peer network, and its relatively high computation requirement is still acceptable.

The accumulation of *coins* is performed by all nodes in the Zerocoin network as they need the most updated accumulator value to verify *Spend transactions*. Although the 39ms taken to accumulate a *coin* is not small, nodes only need to accumulate the newly minted *coins* to the previous accumulator each time a new *block* is received since the accumulator can be incrementally computed (§3.1.2). As blocks are only added to the Blockchain every 10 minutes (as per Bitcoin), nodes only need to periodically accumulate a small subset of the minted *coins* in history. With this optimisation, the computational requirement to accumulate a *coin* is acceptable.

On the other hand, verification of NIZK proofs is done by all nodes in the Zerocoin network whenever they receive a *Spend transaction*. Thus the time needed to verify *Spend transactions* grows linearly with the number of *Spend transactions* received. The verification time of the AccPoK (100ms) and the SNSoK (163ms), which make up the bulk of the verification time of a *Spend transaction*, is an area of concern. In fact, the Zerocoin paper showed that the rate at which nodes verify transactions decreases by about half when only 12.5% of all transactions in the network are *Spend transactions* and the other 12.5% and 75% are *Mint transactions* and standard Bitcoin transactions respectively [17]. In addition, a node may need to verify the same *Spend transactions* twice – once before it forwards the *Spend transaction* to its peers, and another time when a new *block* containing the *Spend transaction* is received. The longer time taken by nodes to verify *Spend transactions* increases the time taken for transactions to reach mining nodes and make it to the Blockchain, and slows the Zerocoin entire network. Thus, the inefficiency in verifying the AccPoK and the SNSoK is a serious limiting factor to Zerocoin's performance, and this research aims to reduce the time taken to verify these proofs.

Storage Requirements

Each node in Zerocoin stores a single copy of the public parameters, thus the 2.5KB of storage required by the public parameters is negligible. The other Zerocoin data structures need to be stored by nodes on a *transaction* or *block* basis as part of the Blockchain, and

their sizes affect the storage requirements of Zerocoin significantly. The rest this section uses Bitcoin as a point of reference to demonstrate the storage implications of Zerocoin.

The accumulator is stored on a *block* level. As of October 2016, the average size of one *block* in Bitcoin is about 0.8MB [7]. Thus an additional accumulator of 391 bytes for each *block* has negligible effect on the *block* size. On the other hand, a *coin* is stored in each *Mint transaction*, while the AccPoK, SNSoK and CommPoK are stored in each *Spend transaction*. As of 2015, the average size of one standard Bitcoin *transaction* in is 566 bytes [24]. For simplicity of analysis, the Zerocoin data structures are added to the standard Bitcoin *transaction* to estimate the size of Zerocoin transactions. As such, a *Mint transaction* is bigger than a Bitcoin *transaction* by the size of a *coin* (199 bytes), and a *Spend transaction* is bigger than a Bitcoin *transaction* by the total size of the AccPoK, SNSoK and CommPoK (25KB). This means that a *Mint transaction* increases transaction size by 35% while a *Spend transaction* increases transaction size by 4,400%.

While a 35% increase in size for a *Mint transaction* is still reasonable, a 44-fold increase in size for a *Spend transaction* is undesirable. To illustrate, if *Spend transactions* make up just 10% of all the transactions on the Blockchain, the total size of the transactions on the Blockchain will increase by 4.4 times. Since transactions make up the bulk of Blockchain stored in every node, the Zerocoin protocol imposes an enormous storage requirement compared to Bitcoin. In addition, the large size of the Zerocoin transaction also slows down the propagation of transactions in the network. As a result, Zerocoin transactions takes longer to reach the mining nodes and make it to the Blockchain, and slows down the entire Zerocoin network. Since the *Spend transaction* contributes significantly the storage requirements and the AccPoK (6974 bytes) and the SNSoK (17,420 bytes) make up the bulk of a *Spend transaction*, this research aims to reduce the size of the AccPoK and SNSoK to improve the overall performance of Zerocoin.

Chapter 4

Contribution of Research

4.1 Objectives

Based on the performance analysis of the Zerocoin in §3.3, this research aims to improve these aspects of Zerocoin:

1. Reduce the time taken to verify the Accumulator Proof of Knowledge
2. Reduce the time taken to verify the Serial Number Signature of Knowledge
3. Reduce the size of the Accumulator Proof of Knowledge
4. Reduce the size of the Serial Number Signature of Knowledge

If time permits, the research will also explore how to improve on the functionality drawbacks of Zerocoin (§2.3.1). For example, this research can look into how to allow direct payments with coins without the need to redeem a coin into Bitcoins before paying with Bitcoins, or how to create coins with arbitrary denomination.

4.2 Performance Metrics and Benchmarks

Some of the performance metrics that will be used to evaluate the proposed improvements are adapted from the Zerocoin paper [17]. These performance metrics are:

1. Size of the AccPoK and SNSoK
2. Time taken to verify the AccPoK and SNSoK

3. Transaction verifications per minute by a node, across different percentages of Zerocoin transactions being processed

However the metrics used by the Zerocoin paper is inadequate as they only measure performance at a node level. To make the tests more realistic, this research will use additional metrics that measure performance at a network level. These performance metrics are:

1. Time taken for transactions to propagate to every node in a network, across different percentages of Zerocoin transactions in the network

2. Time taken for *blocks* to propagate to every node in a network, across different percentage of Zerocoin transactions in the network

The propagation times already include the time taken to verify transactions or *block* as every node is supposed to verify the transactions or *blocks* they receive before forwarding them to its peers. Thus propagation time is a good measure of the overall performance of the Zerocoin protocol. At this point the exact setup of the network that will be used to test the propagation times has not been determined. While it is desirable to conduct the experiments on a network that has a comparable scale to Bitcoin (3500 active nodes with an average of 32 random peers each [9]), it is unlikely to be achieved given limitations in resource. However to make the experiments realistic, the scaled-down network will roughly follow Bitcoin's ratio of active nodes to the number of peers for each node.

Using these performance metrics, the proposed improvements to Zerocoin will be benchmarked against the original Zerocoin protocol to quantify the degree of improvement. In addition, Bitcoin statistics that relate to the performance metrics will also be obtained to see if the proposed improvements are able to bring the performance of Zerocoin to a commercially realistic level.

4.3 Current Status

At this point, this research has proposed a SNSoK that has a smaller size and takes a shorter time to verify than the original SNSoK. The proposed SNSok has been analysed for the three properties of zero-knowledge proofs. It has also been successfully implemented in libzerocoin and its performance has been measured using some basic performance metrics defined in §4.2. The proposed SNSok and its results are detailed in §5. In addition, this research has also proposed an optimisation to the AccPoK that can reduce its size. However the performance of this optimisation is only theoretically evaluated and has not been implemented or tested. The proposed optimisation is detailed in §6.

Chapter 5

Contribution 1: A more Efficient Serial Number Signature of Knowledge

5.1 Problem with Original SNSoK

The most problematic NIZK proof out of the three proofs in Zerocoin is the SNSoK as it has the biggest size and takes the longest time to verify. The inefficiency in the SNSoK is mainly due to the proof requiring 80 iterations of the same proof process (§3.1.3), which means that 80 sets of proof parameters are generated and verified to in one SNSoK. This differs from the standard scheme of the Fiat-Shamir heuristic, where the verification of the resultant proof only requires a single iteration.

5.2 Construction of Proposed SNSoK

This research proposes a more efficient SNSoK that modifies the ZK proof for a committed value in a Pedersen commitment outlined by Hohenberger [12]. The standard Fiat Shamir Heuristics is used to make the proof non-interactive and the resultant proof can be verified in a single iteration.

The proposed SNSoK starts off in a same way as the original SNSoK. In order to hide the *coin c* that is being proven, the prover creates a Pedersen commitment y of c using a random trapdoor $z \in \mathbb{Z}_{q_{SoK}}$, such that $y = g_{SoK}^{c} h_{SoK}^{z} = g_{SoK}^{S_{comm}} h_{comm}^{r} h_{SoK}^{z}$. Also, like in the original proof, the SNSoK proves in zero-knowledge that y contains a c that is a commitment of the serial number S and trapdoor r as follows:

$$ZKSoK[m]\{(c,r,z) : y = g_{SoK}^{S_{comm}} h_{comm}^{r} h_{SoK}^{z}\}$$

However of the proposed SNSoK does not required 80 (t,s,s') values to be included in the proof. Instead, alluding to the methods by Hohenberger, the prover computes a single $t = g_{SoK}^{g_{comm}^{S} h_{comm}^{v_1}} h_{SoK}^{v_2}$ where $v_1 \in \mathbb{Z}_{q_{comm}}, v_2 \in \mathbb{Z}_{q_{SoK}}$. Following the principle of the Fiat-Shamir heuristics, the prover then computes the challenge string $\mathscr{C} = H(y\|S\|g_{SoK}\|h_{SoK}\|t\|m)$ which also doubles up as the signature for the transaction contents m (§3.1.3). The prover then computes a single $s = h_{comm}^{v_1} - \mathscr{C} h_{comm}^{r}$ and $s' = v_2 - \mathscr{C} z$. The prover writes y,t,s,s' and the transaction content m into the *Spend transaction*. Upon receiving the *Spend transaction*, the verifying node recomputes $\mathscr{C} = H(y\|S\|g_{SoK}\|h_{SoK}\|t\|m)$ using the received y,t,m, the serial number S of the *coin* and the public parameters g_{SoK}, h_{SoK}. The verifier then verifies the proof by checking $t = y^{\mathscr{C}} g_{SoK}^{g_{comm}^{S} s} h_{SoK}^{s'}$.

Since only a single tuple (t,s,s') is included in the proposed SNSoK as opposed to the 80 (t,s,s') in the original SNSoK, the proof size of the proposed SNSoK should be smaller by about 80 times. Similarly, the time needed to verify the proposed SNSoK should also decrease by a similar amount since verification is done in one iteration instead of the 80 iterations in the original SNSoK. The exact improvements in performance brought by the proposed SNSoK is shown in §5.4.

5.3 Zero-Knowledge Properties of Proposed SNSoK

In order for the proposed SNSoK to be valid, it must satisfy the three ZK properties defined in §3.1.3.

5.3.1 Completeness

The completeness property of the improved SNSoK can be shown by demonstrating that the proof produced by an honest prover who knows S,r,z can always to be verified. Hence it suffices to show that the verification equation is correct. To prove that $t = y^{\mathscr{C}} g_{SoK}^{g_{comm}^{S} s} h_{SoK}^{s'}$, the equation can be expanded as such:

$$
\begin{aligned}
t &= y^{\mathscr{C}} g_{SoK}^{g_{comm}^{S} s} h_{SoK}^{s'} \\
&= (g_{SoK}^{g_{comm}^{S} h_{comm}^{r}} h_{SoK}^{z})^{\mathscr{C}} g_{SoK}^{g_{comm}^{S} (h_{comm}^{v_1} - \mathscr{C} h_{comm}^{r})} h_{SoK}^{v_2 - \mathscr{C} z} \\
&= g_{SoK}^{\mathscr{C} g_{comm}^{S} h_{comm}^{r}} h_{SoK}^{\mathscr{C} z} g_{SoK}^{g_{comm}^{S} h_{comm}^{v_1} - \mathscr{C} g_{comm}^{S} h_{comm}^{r})} h_{SoK}^{v_2 - \mathscr{C} z} \\
&= g_{SoK}^{g_{comm}^{S} h_{comm}^{v_1}} h_{SoK}^{v_2} \\
&= t
\end{aligned}
$$

As seen, the verification equation is indeed correct and the proposed SNSoK is complete.

5.3.2 Soundness

The soundness property of improved SNSoK can be shown by demonstrating that a hypothetical knowledge extractor can obtain c and z from two accepting proofs that uses the same t but different $(\mathscr{C}, s, s')$ (§3.1.3). Given these conditions, and let the two different $(\mathscr{C}, s, s')$ be $(\mathscr{C}_1, s_1, s'_1)$ and $(\mathscr{C}_2, s_2, s'_2)$, the following equations be can constructed:

$$t = y^{\mathscr{C}_1} g_{SoK}^{g_{comm}^{S} s_1} h_{SoK}^{s'_1} = y^{\mathscr{C}_2} g_{SoK}^{g_{comm}^{S} s_2} h_{SoK}^{s'_2}$$

Hence it can be derived that:

$$\begin{aligned} y^{\mathscr{C}_1} g_{SoK}^{g_{comm}^{S} s_1} h_{SoK}^{s'_1} &= y^{\mathscr{C}_2} g_{SoK}^{g_{comm}^{S} s_2} h_{SoK}^{s'_2} \\ y^{\mathscr{C}_1 - \mathscr{C}_2} &= g_{SoK}^{g_{comm}^{S}(s_2 - s_1)} h_{SoK}^{s'_2 - s'_1} \\ y &= g_{SoK}^{\frac{g_{comm}^{S}(s_2 - s_1)}{\mathscr{C}_1 - \mathscr{C}_2}} h_{SoK}^{\frac{s'_2 - s'_1}{\mathscr{C}_1 - \mathscr{C}_2}} \end{aligned}$$

Since $y = g_{SoK}^{c} h_{SoK}^{z}$, the knowledge extractor can successfully determine that $c = \frac{g_{comm}^{S}(s_2 - s_1)}{\mathscr{C}_1 - \mathscr{C}_2}$ and $z = \frac{s'_2 - s'_1}{\mathscr{C}_1 - \mathscr{C}_2}$. Hence the success of the knowledge extractor in extracting the secret *coin* c and the secret trapdoor z shows that the proposed proof is sound.

5.3.3 Statistical Zero-Knowledge

The statistical zero-knowledge property of the improved SNSoK can be shown by demonstrating that a hypothetical simulator can generate s and s' that are statistically indistinguishable from the s and s' generated by the prover (§3.1.3).

Firstly it can be shown that the s and s' generated by the prover are random numbers. Since h_{comm} is the generator for the subgroup of order q_{comm}, h_{comm}^{x} produces all element in the subgroup exactly once for x from 1 to q_{comm}. Thus there is a one-to-one mapping between v_1 and $h_{comm}^{v_1}$ and since $v_1 \in \mathbb{Z}_{q_{comm}}$ and v_1 is random, $h_{comm}^{v_1}$ is also random. As a random number added with a constant still produces a random number, $s = h_{comm}^{v_1} - \mathscr{C} h_{comm}^{r}$ is also random. Similarly, since v_2 is random, $s' = v_2 - \mathscr{C} z$ is also random.

As such the simulator can be just a random number generator that produces random numbers $s \in \mathbb{Z}_{q_{comm}}$ and $s' \in \mathbb{Z}_{q_{SoK}}$. Since the s and s' produced by the simulator and the s and s'

produced by the prover are both uniformly distributed, they are statistically indistinguishable. Hence the proposed SNSoK achieves statistical zero-knowledge.

5.4 Performance of Proposed SNSoK and Future Work

The performance of the proposed SNSoK is measured using some basic performance metrics defined in §4.2. The results are shown in Table 5.1.

	Average over 50 iterations	
	Proposed	**Original**
Size of one SNSoK	390 bytes	17,420 bytes
Verification time of one SNSoK	1.8ms	163ms

Table 5.1 Performance of proposed SNSoK using basic performance metrics

As seen, the proposed SNSoK has drastically reduced the verification time and size of one SNSoK. The verification time is reduced by about 80 times. This is within expectation as the proposed SNSoK essentially reduces the 80 iterations required for each verification to one iteration. The size of the SNSoK is reduced by about 44 times. While this is a huge improvement, it does not match up to expectations as the proposed SNSoK should have reduced the size of the proof by 80 times since it reduces the 80 sets of proof parameters (t, s, s') to just a single set.

The original SNSoK is smaller than expected because it has been optimised. The idea of the original SNSoK is the same as the standard NIZK proof, where the verifier test whether the t sent by the prover equals to the one that it computes using the response s from the prover and the challenge $\mathscr{C}$. Specifically the verifier of the original SNSoK checks for the equality between the received t and the t' computed based on the received $s_i, s'_i, \mathscr{C}$ (§3.1.3). However, the verifier in the original SNSoK does not check for this equality directly. Since the $\mathscr{C}$ sent by the prover is a hash that contains of all the t_i (§3.1.3), the verifier simply checks that the $\mathscr{C}'$ obtained by hashing all the t'_i equals to $\mathscr{C}$. Under this scheme, a single 256 bits $\mathscr{C}$ is included in the proof instead of the 80 1024 bits t_i, resulting in a much smaller proof size. Currently, the proposed SNSoK does not make use of this optimisation and includes t in the proof. However such optimisation is likely to be applicable to the proposed SNSoK and will be explored in the subsequent phase of the research.

The tests conducted to evaluate the proposed SNSoK are still preliminary. This research has yet to show the effects of the proposed SNSoK on the Zerocoin network. Going forward, the research will conduct experiments to examine the performance of the original and the

improved Zerocoin protocol on a network level using the performance metrics outlined in §4.2.

Chapter 6

Contribution 2: A Smaller Accumulator Proof of Knowledge

6.1 Problem with Original AccPoK

The AccPoK is the other NIZK proof in Zerocoin that is relatively inefficient. This is mainly because the AccPoK contains 8 sub-proofs that must be all verified (§3.1.3). While the steps in the proof are made clear by its authors Camenisch and Lysyanskaya [14], the reasons behind every sub-proof are not described in detail. To revisit the AccPoK, the proof statement is shown:

$$NIZKPoK(c,\beta,\gamma,\delta,\varepsilon,\zeta,\varphi,\psi,\eta,\sigma,\xi):\{$$

$$\mathfrak{C}_c = \mathfrak{g}^c\mathfrak{h}^{\varphi} \wedge \mathfrak{g} = (\mathfrak{C}_c/\mathfrak{g})^{\gamma}\mathfrak{h}^{\psi} \wedge \mathfrak{g} = (\mathfrak{g}\mathfrak{C}_c)^{\sigma}\mathfrak{h}^{\xi} \wedge$$

$$C_r = g_{QRN}^{\varepsilon}h_{QRN}^{\zeta} \wedge C_c = g_{QRN}^{c}h_{QRN}^{\eta} \wedge A = C_w^c(1/h_{QRN})^{\beta} \wedge$$

$$1 = C_r^c(1/h_{QRN})^{\delta}(1/g_{QRN})^{\beta} \wedge c \in [\mathfrak{A},\mathfrak{B}]\}$$

$\mathfrak{C}_c = \mathfrak{g}^c\mathfrak{h}^{\varphi}$ is a sub-proof that $\mathfrak{C}_c$ contains the *coin* c, $A = C_w^c(1/h_{QRN})^{\beta} = (wh_{QRN}^{\varepsilon})^c (1/h_{QRN})^{\beta} = w^c h_{QRN}^{c\varepsilon-\beta}$ is a sub-proof for the membership of c in A using witness w, and $c \in [\mathfrak{A},\mathfrak{B}]$ is a sub-proof that the value of c is within the allowed range. However this research is unable to understand the reasons behind the other sub-proofs and thus little can be done to improve the AccPoK at the theoretical level. Nevertheless, this research has found an implementation optimisation to reduce the size of the AccPoK.

6.2 Proposed Size Optimisation for AccPoK

The verifier of the current AccPoK receives the following values from the prover:

$$s_\alpha, s_\beta, s_\zeta, s_\sigma, s_\eta, s_\varepsilon, s_\delta, s_\xi, s_\varphi, s_\gamma, s_\psi, \mathfrak{t}_1, \mathfrak{t}_2, \mathfrak{t}_3, t_1, t_2, t_3, t_4, \mathscr{C}$$

$\mathscr{C}$ is the challenge string that has the value $H(\mathfrak{C}_c \| C_c \| C_w \| C_r \| \mathfrak{g} \| \mathfrak{h} \| g_{QRN} \| h_{QRN} \| \mathfrak{t}_1 \| \mathfrak{t}_2 \| \mathfrak{t}_3 \| t_1 \| t_2 \| t_3 \| t_4)$. $\mathscr{C}$ is 256 bits long while the average size of each s_i, t_i and $\mathfrak{t}_i$ are 2080, 3070 and 550 bits respectively [1]. The large amount of proof parameters and the relatively large size of each parameter is the main reason for the large size of the AccPoK. Borrowing the concept used in the original SNSoK (§3.1.3), the size of the proof can be reduced by removing the need to send t_i and $\mathfrak{t}_i$ to the verifier.

When verifying the AccPoK, the verifier essentially computes a set of t_i' and $\mathfrak{t}_i'$ using the $\mathscr{C}$ and s_i received from the prover and checks if t_i' and $\mathfrak{t}_i'$ equal to the corresponding t_i and $\mathfrak{t}_i$ received from the prover. For example, one equality that the verifier checks is whether $\mathfrak{t}_1 = \mathfrak{C}_c^{\mathscr{C}} \mathfrak{g}^{s_\alpha} \mathfrak{h}^{s_\zeta}$. This process can be modified such that the verifier does not check for the equalities directly. To achieve this, the verifier computes t_i' and $\mathfrak{t}_i'$ like before, but also further computes $\mathscr{C}' = H(\mathfrak{C}_c \| C_c \| C_w \| C_r \| \mathfrak{g} \| \mathfrak{h} \| g_{QRN} \| h_{QRN} \| \mathfrak{t}_1' \| \mathfrak{t}_2' \| \mathfrak{t}_3' \| t_1' \| t_2' \| t_3' \| t_4')$. Then, the verifier simply checks that $\mathscr{C}'$ equals to the $\mathscr{C}$ received from the prover to verify the proof. The proposed optimization works based on the same idea detailed in §5.4. Since the $\mathscr{C}$ sent by the prover is a hash of all the t_i and $\mathfrak{t}_i$ together with other public parameters, $\mathscr{C} = \mathscr{C}'$ if and only if the t_i' and $\mathfrak{t}_i'$ that are used to compute $\mathscr{C}'$ equals to their corresponding and t_i and $\mathfrak{t}_i$.

With the proposed optimisation, the prover includes $\mathscr{C}$ in the AccPoK instead of t_i and $\mathfrak{t}_i$. Given that $\mathscr{C}$ is only 256 bits long while the four t_i and three $\mathfrak{t}_i$ have an average size of 3070 bits and 550 bits respectively, the proposed optimisation should be able to reduce the size of the AccPoK by about 1.7KB. However, the proposed optimisation has not been implemented and its performance remains to be seen.

6.3 Future Work

To ensure that the AccPoK works with the proposed optimisation, it will first be implemented and tested in libzerocoin. Some basic performance metrics such as the size of the optimised AccPoK and the time needed to verify it will be collected. Eventually, the Zerocoin protocol that includes the optimised AccPoK will be tested on a network level using the set up outlined in §4.2. In the meantime, this research will also explore other areas of optimisation for the AccPoK.

[1] Values are obtained by running libzerocoin.

Conclusion and Research Timeline

Up till this point, this research has accomplished the following:

- Examined the issue of anonymity in Bitcoin and digital currencies
- Explored the different ways to anonymise digital currencies and identified Zerocoin as the area of interest
- Examined in the detail the Zerocoin protocol and its theoretical basis
- Analysed the implementation of Zerocoin and identified areas of improvement
- Established the performance metrics and benchmarks to evaluate the proposed improvements
- Proposed an improved Serial Number Signature of Knowledge in Zerocoin Spend transactions and conducted some preliminary tests
- Proposed an optimisation for the Accumulator Proof of Knowledge in Zerocoin Spend transactions

Going forward, this research plans to work on the following areas:

- Set up a peer to peer network that simulates the Zerocoin network to conduct network level tests
- Compare the performance of the proposed improvements against the original Zerocoin protocol at the network level
- Continue to evaluate the theoretical soundness of the proposed improvements and make necessary adjustments
- Further enhance the proposed improvements
- Explore ways to improve the other drawbacks in Zerocoin's functionalities

A research timeline outlining the brief schedule for the completed and future tasks is shown in Fig. 6.1

<table>
<tr><th rowspan="2">Month</th><th colspan="4">Tasks performed / to perform</th></tr>
<tr><th colspan="2">Acquiring of Knowledge</th><th colspan="2">Analysis</th><th>Implementation</th></tr>
<tr><td>Aug 2016</td><td>Background reading on Bitcoin and the issue of anonymity</td><td rowspan="2">Understand the Zerocoin protocol and its associated zero-knowledge proofs</td><td colspan="2" rowspan="2">Identify the problems with Zerocoin</td><td rowspan="2">Understand the C++ source code of Zerocoin (libzerocoin)</td></tr>
<tr><td>Sep 2016</td><td>Explore the different ways to achieve anonymity in digital currency systems</td></tr>
<tr><td>Oct 2016</td><td rowspan="4"></td><td rowspan="4">Explore other efficient zero-knowledge proofs that may be applicable to Zerocoin</td><td rowspan="2">Develop improvements for the SNSoK and the AccPoK</td><td rowspan="4">Develop ways to improve the other drawbacks in Zerocoin's functionalities if possible</td><td rowspan="2">Implement the proposed SNSoK and conduct some preliminary tests</td><td>Implement the proposed AccPoK and conduct some preliminary test</td></tr>
<tr><td>Nov 2016</td><td>Examine the theoretical soundness of the proposed SNSoK and AccPoK, and make necessary adjustments</td><td rowspan="2">Set up a peer-to-peer network to test the proposed improvements to Zerocoin on a network level</td><td rowspan="2">Update the implementation of the proposed SNSoK and AccPoK with any changes or enhancements</td></tr>
<tr><td>Dec 2016</td><td>Further enhance the proposed SNSoK and AccPoK if possible</td></tr>
<tr><td>Jan 2017</td><td rowspan="2">Finalise all improvements</td><td rowspan="2">Conduct network level tests on the proposed improvements to Zerocoin. Also perform the same tests for the original Zerocoin protocol as a benchmark.</td><td rowspan="2">Implement all improvements</td></tr>
<tr><td>Feb 2017</td></tr>
<tr><td>Mar 2017</td><td colspan="5">Collate experimental results and complete research report</td></tr>
<tr><td>Apr 2017</td><td colspan="5">Submission of research report</td></tr>
</table>

Fig. 6.1 Research timeline

References

[1] Androulaki, E., Karame, G. O., Roeschlin, M., Scherer, T., and Capkun, S. (2013). Evaluating User Privacy in Bitcoin. *Financial Cryptography and Data Security*, pages 34–51.

[2] Antonopoulos, A. M. (2014). Mastering Bitcoin: Unlocking Digital Cryptocurrencies.

[3] BariĆ, N. and Pfitzmann, B. (1997). Collision-free accumulators and fail-stop signature schemes without trees. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 1233:480–494.

[4] Bellare, M. and Rogaway, P. (1993). Random oracles are practical: A paradigm for designing efficient protocols. *Proceedings of the 1st ACM conference on Computer and communications security*, (November 1993):62–73.

[5] Ben-Sasson, E., Chiesa, A., Garman, C., Green, M., Miers, I., Tromer, E., and Virza, M. (2014). Zerocash: Decentralized anonymous payments from bitcoin. In *Proceedings - IEEE Symposium on Security and Privacy*, pages 459–474.

[6] Ben-sasson, E., Chiesa, A., and Tromer, E. (2013). Succinct Non-Interactive Arguments for a von Neumann Architecture. *USENIX Security*, pages 1–35.

[7] Blockchain.info (2016). Average Block Size.

[8] Camenisch, J. and Stadler, M. (1997). Efficient Group Signature Schemes for Large Groups. *CRYPTO '97: Advances in Cryptology*, pages 410–424.

[9] Decker, C. and Wattenhofer, R. (2013). Information propagation in the Bitcoin network. *13th IEEE International Conference on Peer-to-Peer Computing, IEEE P2P 2013 - Proceedings*.

[10] Fiat, A. and Shamir, A. (1987). How To Prove Yourself : Practical Solutions to Identification and Signature Problems. pages 186–194.

[11] Gmaxwell (2013). CoinJoin: Bitcoin privacy for the real world.

[12] Hohenberger, I. S. (2002). Lecture 10 : More on Proofs of Knowledge Examples of Proofs of Knowledge. *Compute*, 1:1–8.

[13] Information Technology Laboratory, N. I. o. S. and Technology (2009). Digital Signature Standard (DSS).

[14] Jan Camenisch1 and Anna Lysyanskaya2 (2002). Dynamic Accumulators and Application to Efficient Revocation of Anonymous Credentials. *Crypto*, page 16.

[15] Kiayias, A. (2010). Zero-Knowledge Proofs. pages 1–10.

[16] Meiklejohn, S., Pomarole, M., Jordan, G., Levchenko, K., McCoy, D., Voelker, G. M., and Savage, S. (2013). A fistful of Bitcoins: Characterizing payments among men with no names. *Proceedings of the Internet Measurement Conference - IMC '13*, (6):127–140.

[17] Miers, I., Garman, C., Green, M., and Rubin, A. D. (2013). Zerocoin: Anonymous distributed e-cash from bitcoin. In *Proceedings - IEEE Symposium on Security and Privacy*, pages 397–411.

[18] Moser, M., Bohme, R., and Breuker, D. (2013). An inquiry into money laundering tools in the Bitcoin ecosystem. *eCrime Researchers Summit, eCrime*.

[19] Nakamoto, S. (2008). Bitcoin: A Peer-to-Peer Electronic Cash System. *Www.Bitcoin.Org*, page 9.

[20] Narayanan, A., Bonneau, J., Felten, E., Miller, A., and Goldfeder, S. (2016). *Bitcoin and Cryptocurrency Technologies: A Comprehensive Introduction*. Princeton University Press.

[21] No, E. T. H., Sciences, T., Maurer, U., and Damg, I. B. (1998). Group Signature Schemes and Payment Systems Based on the Discrete Logarithm Problem. (12520).

[22] Paar, P. D.-I. C. and Pelzl, D.-I. J. (2010). *Digital Signatures*.

[23] Pedersen, T. P. (1992). Non-interactive and information-theoretic secure verifiable secret sharing. *Advances in Cryptology Crypto 91*, 91:129–140.

[24] TradeBlock (2015). Analysis of Bitcoin Transaction Size Trends.

[25] Unknown (2016a). Fiat–Shamir heuristic.

[26] Unknown (2016b). Random oracle.

[27] Zcoin (2016). Zcoin and Zcash: Similarities and Differences.