An Analysis of Anonymity in the Zcash Cryptocurrency

by

Jeffrey Quesnelle

A thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science
(Computer and Information Science)
in the University of Michigan-Dearborn
2018

Master's Thesis Committee:

Associate Professor Di Ma, Chair
Assistant Professor Anys Bacha
Associate Professor Shengquan Wang

ACKNOWLEDGMENTS

I would like to give my thanks to everyone who supported me patiently throughout my long academic career. First, to my parents, who put up with a nagging child when he insisted that he *had* to go to summer computer camp at the University of Michigan to learn programming. To Dave Robins and Intrepid Control Systems for supporting my desire to continue my education. To Daniel Steffy at Oakland University for showing me the ropes of the research and writing process, as well as for being an all-around excellent teacher. To Brandon Goodell of Monero Research Labs for his insightful notes on the content of this thesis. Lastly, to my wife for her love and undying support of all my endeavors.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

Cryptocurrencies such as Bitcoin have shown that a game theory approach to decentralized consensus can create value. In Bitcoin's game theory, as long as an adversary does not acquire a majority of computational power it is more profitable for them to obey by the rules of the network. Moreover, Bitcoin's transparent, immutable, publicly auditable ledger allows any party to trivially verify the correctness of transactions. This transparency means that an adversary may, while obeying the rules of the network, trace the flow of transactions. By corresponding a transaction to an individual, the adversary may determine the source and destination of that user's funds, resulting in a serious loss of privacy.

Several alternative cryptocurrencies ("altcoins") have endeavored to create systems that preserve privacy. The chief difficulty in creating such a system is devising a way that the correctness of transactions can be easily verified while obscuring the underlying details of the transactions. Such systems are akin to homomorphic encryption, where operations carried out on ciphertext correspond to the same operation on the cleartext. In this thesis, we review a cryptographic method known as zk-SNARKs for anonymizing transactions in cryptocurrencies. We summarize the mathematical foundations of this construction, tracing the development of its underlying principles through the literature. We also analyze Zcash, a publicly traded cryptocurrency that uses zk-SNARKs. Using blockchain analysis along with certain heurestics, we are able to potentially deanonymize transactions that account for 31.5% of Zcash's private transaction volume.

CHAPTER 1

Introduction

1.1 Motivation

For as long as computers have been available commercially, they have been used to process currency transactions. One of the first uses of IBM's mainframes of the 1950s and early 1960s was automating the sorting and validating of paper checks. Soon, account balances became merely an entry in a bank's database. However, these systems were internal to each organization. The creation of SWIFT in 1973 connected these systems, allowing banks to electronically transfer funds between each other.

The SWIFT network, despite being electronic, is and was a "walled garden". Banks agreed to act in good faith through so-called correspondent relationships. As general networking reached academics in the 1980s and consumers in the 1990s, attempts at true "digital money" were made. True to their cyberpunk roots, the creators of these early systems desired a decentralized network with each member acting in their own self-interest and outside the control of any state. The chief difficulty such systems encountered was ensuring the correctness in the presence of malicious actors. A second concern was defining monetary policy (i.e. how money is issued) without a central planner.

The first widespread success of digital money, Bitcoin, tied the solution of these two problems together. The Bitcoin whitepaper defined a system using public-key cryptography along with properties of cryptographic hash functions to generate a decentralized consensus via a novel technique

known as proof-of-work. Furthermore, users are incentivized to act in good faith by the monetary policy that only creates new currency as a reward for contributing to the consensus state. At last, the dream of fully decentralized and censorship resistant digital money not tied to any nation-state's monetary policy was viable. The term *cryptocurrency* was born.

Although Bitcoin succeeded in creating a decentralized digital currency, privacy or anonymity were never an explicit goal of the system. As Bitcoin has become more popular, the effects of its lack of privacy guarantees have become more apparent. In a hypothetical world where all transactions use Bitcoin, the transparent nature of Bitcoin transactions, which trivially allow anyone to follow the flow of money from one user to another could have serious social repercussions. To this end, several alternative cryptocurrencies have endavored to design systems where a user's transactions, both their amount and destinations, are private.

The principle difficulty in such a system is ensuring its correctness can be audited while still maintaining privacy guarantees. The decentralized nature of cryptocurrencies mandates that each participant be able to independently verify and deduce the consensus state. For example, the two most important invariants are that no user can spend the same money more than once and that money can only be spent by the real owner. When the amounts and parties involved in transactions are public this is simple to verify. Can a system be designed that guarantees these invariants in a private manner? Our thesis reviews the mathematical underpinnings of one such system, evaluates a concrete implementation called Zcash, and discovers a potential flaw that may invalidate its anonymity guarantees.

1.2 Contributions

Constructions attesting that a party is in possession of a satisfying assignment to a problem instance have long been studied in computer science. Many such constructions fix the problem statement, varying only the parameters of the instance. For example, an RSA signature attests to

the knowledge of an exponent in particular modular arithmetic statement without revealing what the exponent is. In our thesis, we first review a recent construction for generalized zero-knowledge proofs of arbitrary statements in NP. These proofs are known as zk-SNARKs. Our first contribution is an extended review of the mathematical principles behind zk-SNARKs.

Using zk-SNARKs as the principle cryptographic primitive for anonymous transactions, a cryptocurrency named Zcash was launched in 2016. Our second contribution is an introduction to Zcash, with particular emphasis on the application of zk-SNARKs to enforce the privacy guarantees.

Although zk-SNARKs allow for anonymous transactions in Zcash, the use of such transactions are optional. Zcash also allows for transparent transactions, which do not give the same privacy guarantees. The Zcash coins themselves can be used interchangeably in anonymous and non-anonymous transactions. When this interchange happens, the parties involved are anonymous, but the amounts are not. Our final contribution is an analysis on how certain uses of these interchange transactions can lead to a non-obvious privacy loss. Our experimental analysis discovered that these conditions are pervasive in Zcash's transaction history, and using specific heuristics we potentially deanonymize transactions that account for 31.5% of all Zcash coins.

1.3 Outline

We have organized our thesis in the following way. In Chapter 2 we provide the historical context of cryptocurrencies, and argue that privacy and fungiblity are essential characteristics of a functional electronic cash. We further argue that Bitcoin in particular performs poorly by these measurements. We also review some definitions related to knowledge and proofs. In Chapter 3 we give a through review of zk-SNARKs, a technical construction that enables anonymous cryptocurrencies. In Chapter 4 we review Zcash, an implementation of this construction. In Chapter 5 we investigate a flaw in Zcash which allowed us to potentially deanonymize several real-world

transactions. We conclude in Chapter 6.

CHAPTER 2

Preliminaries

2.1 Currency Evolution

In early human history most economic transactions were made with instruments that had inherent value. For example, when chickens are exchanged for bags of flour, the value transferred is the underlying utility the items provide. However, under such a system, two parties may find it difficult to transact unless they both have a need for the utility provided by real goods the other party possesses. Often, several high-demand goods such as grain, cloth, and copper became *de facto* currencies [Muh16], since many parties had demand for such items, or at least had reasonable expectation that other parties with whom they wished to transact with soon would. To ease the burden of physically transacting items, some early cultures traded tokens imprinted with the likeness of the goods. Eventually, most cultures shifted to coinage cast out of precious metals, starting first with the ancient kingdom of Lydia in the 600s B.C. [Sch04].

The switch to precious metals from real (or representations of real) goods was a slight yet profound shift in monetary theory. For such a shift to have been successful, precious metals must have been able to emulate the properties of real goods, at least insofar as facilitating a transaction goes. But what are these properties? The economist Philipp Bagus in his treatise *The Quality of Money* says that good money must be divisible, portable, durable, and stable in value [Bag09]. The first three of these qualities were met by the physical properties of precious metals. The fourth, stable in value, was a social economic consensus derived by the scarcity of the metal itself.

Judging by the aforementioned qualities, precious metals were a superior medium of exchange as compared to real goods. To further reduce the burden of having to physically transact in actual metals, banks began issuing "bank notes", which were pieces of paper that guaranteed the bearer a right to the prescribed amount of precious metals on demand. As long as the banks were able to guarantee the unforgeability of the instruments, bank notes could be used in replacement of actual metals. However, the ultimate value of the note was always derived from the value of the underlying metal, which itself was derived from its relative physical scarcity. Eventually, national banks began issuing notes and the era of national paper currencies began.

Starting with the creation of the Federal Reserve in 1913, the United States dollar was only fractionally backed by precious metals [Leh14]. In 1971, the convertibility of the dollar to gold was completely abolished [Eic10]. Suddenly, the world's largest economy was backed only by "the full faith and credit of the United States", which is to say, by the economy itself. The elasticity of such a system has allowed the government to intercede directly in the economy, a tool unavailable when every dollar must be upheld by a quantity of physical gold. For example, to help mitigate the impact of the 2008 financial crisis, through a process known as *quantitative easing*, the Federal Reserve purchased nearly $1.5 trillion in under performing assets from troubled banks, effectively "creating" money [Bul10].

Under the modern fiat system it is incumbent on the central bank to manage the money supply and, through police power, punish those that violate its soundness (e.g. through counterfeiting). In an electronic cash system the soundness must be guaranteed through technical means. Recall the four properties of quality money previously introduced: divisible, portable, durable, and stable in value. The first three deal mainly with *how* money is transferred. An electronic cash system that is merely a digital representation of a currency (e.g. PayPal) is then only concerned with these challenges. The fourth property, store of value, is principally socio-economic. For electronic cash systems that aim to replace (or at least operate independently) of the traditional fiat world, we shall simply note that the currency's value is partially driven by how well it achieves the other three

properties in an efficient and secure manner.

Ubiquitous authentication cryptography is due to the advent of public-key cryptosystems [DH76; RSA78; HPS98]. The underlying principles of public-key crpytosystems have remained relatively unchanged since their inception. To encrypt message M, the sender generates a key pair (e, d) and communicates d to the receiver out-of-band. The sender, equipped with a function f such that $C = f(e, M)$ and $M = f(d, C)$ transmits C to the receiver. The receiver, now in possession of d and C and with prior knowledge of f can recover M. For such a system to work, the function f and the key pair (e, d) must be carefully constructed to satisfy certain mathematical constraints. One such critical constraint is that determining e given a known d and C must be computationally infeasible for the specific threat model under consideration.

Consider if instead the sender merely wishes to attest that they are in possession of e. In this instance, the user selects a public message M to encrypt with e, yielding C. The user publishes M, C, and d; any verifier can use d to verify that C decrypts back to M. This method for creating digital signatures is the fundamental mathematical procedure behind nearly all crpytocurrencies. For example, in Bitcoin a transaction occurs when the owner of coins signs a hash of the previous transaction of the coins along with the public key of the intended recipient. In essence, the sender has given a *witness* or *proof* that they are in possession of e without revealing any information about e itself.

The first suggestion to use these cryptographic tools to create a payment system was [Cha82] by Chaum. [Cha82] introduced *blind signatures*, which used digital signatures of both the payer and payee as well as a central authority to verify the validity of a transaction. The properties of blind signatures gave the payer anonymity; the payer uses their signature along with the bank to create a unforgeable instrument that can be traded anonymously. In this way, [Cha82] described a *fungible* currency implemented through cryptographic methods. We shall return to this concept of fungibility later.

Although [Cha82] used asymmetric cryptography to ensure the correctness of transactions, it

was not explicitly electronic. In fact, the protocol itself is presented in terms of nested carbon copy envelopes and a central banking institution. Chaum et. al. later expanded upon this work in [CFN90] to develop a full cryptographic electronic cash system. The DigiCash company was created to leverage this technology to market under the name *ecash*, but it ultimately did not gain wide adoption.

The ecash system required a central authority to act as a clearinghouse for transactions, acting as the ultimate arbiter in a dispute. Also, the central authority was responsible for issuing new currency. A significant question is how such a process can be made decentralized while maintaining correctness. In 1998, Dei proposed *b-money* [Dai98], which introduced the concept of decentralized digital scarcity. In b-money, new currency is created when an actor on the network broadcasts the solution to a difficult computational problem. The computational task, known now as *proof-of-work* should be a solution that is difficult to compute, but easy to verify (e.g. an NP-Hard problem). When verified, each actor credits to the broadcaster, in an internal database of all balances, an amount proportional to the cost of the computation time needed to solve the problem. The availability of new currency is made scarce by imposing a real-world cost on its creation, since no known efficient algorithm for producing the solution is known.

Proof-of-work schemes were first studied in other contexts such as spam reduction. The symmetric proof-of-work function Hashcash [Bec97] was envisioned as a method to rate-limit spammers by requiring all e-mails to include a unique proof-of-work in their headers. The Hashcash function itself is a double hash of input data along with a nonce, and the target solution being a hash with predetermined number of leading zero bits. Assuming a collision-resistant hash function, the difficulty of finding a solution is directly proportional to the number of leading zeroes, since every choice of nonce is equally likely to produce the desired output. The adjustable difficulty along with relative simplicity of its implementation and the fact that the solution also functioned as a hash of its input led to speculation of its efficacy in other domains. Indeed, b-money was first introduced on a mailing list discussing such applications.

Although b-money's use of proof-of-work to mint new currency was a novel development, it only functioned under non-adversarial conditions. For example, users were not incentivized to accept a new proof-of-work to create currency, since the newly minted currency devalued their own holdings via inflation. Likewise, transactions could be arbitrarily ignored. In the end, b-money did not describe a system that was likely to reach consensus under non-optimal conditions.

How can a decentralized payment system ensure the correct behavior of its users, especially when the rewards for cheating can potentially be so lucrative? A generalized version of this problem is known as the *Byzantine Generals Problem* [LSP82]. It was shown that quorum systems were effective for Byzantine problems assuming any chosen subset of participants were honest [MR98], but decentralized networks were open to Sybil attacks [Dou02].

2.2 Bitcoin

In 2009 the pseudonymous Satoshi Nakomoto published *Bitcoin: a peer-to-peer electronic cash system* [Nak09], which proposed an economic incentive solution to the Byzantine Generals Problem. The setup was simple: transactions would be made by digitally signing the amount transfered and the public key of the intended recipient, and a peer-to-peer ledger of all transactions would serve as the official record.

Bitcoins are transferred by *transactions*. A Bitcoin transaction sends coins by signing a hash of the public key of the intended recipient[1] and the amount. This hash of the intended recipient's public key is known as an *address*. Typically, this hash is displayed in a Base58 format, e.g. 3D2oetdNuZUqQHPJmcMDDHYoqkyNVsFk9r. To receive Bitcoin, a user shares their address with the sender, who then indicates the address as the destination in their transaction. These coins can later be sent by signing a new transaction with the corresponding private key (see Figure 2.1[2]).

[1] In truth, there are several different ways to transact coins, since Bitcoin includes a built-in scripting language. However, for illustrative purposes this is the most common method

[2] Image © Satoshi Nakomoto, used under MIT License

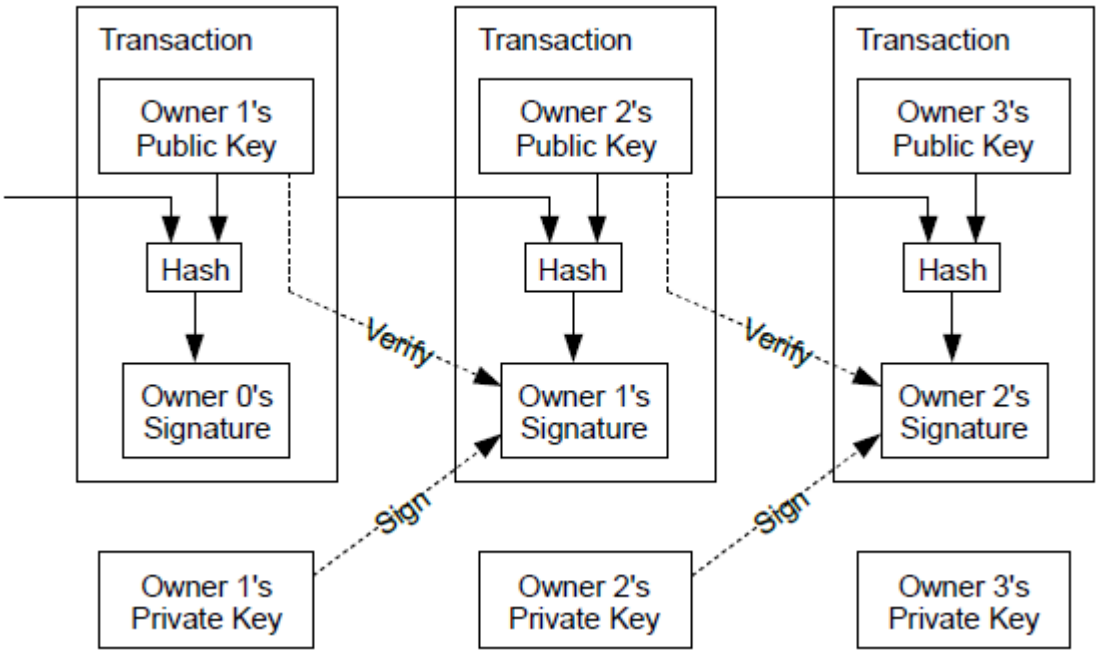


Figure 2.1: Transaction construction from the original Bitcoin whitepaper

Under the rules of Bitcoin, a transaction is not considered completed until it is included in a block, which is a collection of transactions published along with a proof-of-work hash of the block itself. To incentivize good behavior, the successful publishing of a block accompanied with a valid proof-of-work is rewarded with newly created coins, known as the *block reward*. Users that attempt to solve the proof-of-work and produce a new block are known as *miners*.

In addition to the block reward, every transaction may include a fee that will be paid to the miner that includes the transaction in a block. Users compete economically for inclusion in a block by the amount of the fee their transaction offers. A malicious miner may refuse to include certain transactions, but they pay an opportunity cost by including less profitable transactions in the block, which will garner them fewer fees.

Since each block also contains a hash of the previous block, the longest chain of blocks (i.e. a *blockchain*) that conforms to the Bitcoin rules is deemed to be the consensus state of the network. Although an attacker is free to ignore blocks, conforming clients will always follow the longest

valid chain with the most proof-of-work securing it. If any specific adversary controls more than 50% of the total computational power of the network, they will always be able to produce the longest proof-of-work chain, and dictate which transactions will be included in the consensus state. However, as long as this situation is avoided, it is more economical to obey the rules and attempt to obtain the block reward and fees in good faith.

Bitcoin uses a Hashcash-style proof-of-work function whose difficulty is updated every 2016 blocks. The network attempts to adjust the difficulty to reach an equilibrium of one block every ten minutes on average. Each block is around 1 MB in size, although the specific maximum varies depending on the composition of the transactions included in the block.

Verifying the correctness of the Bitcoin blockchain is relatively simple. Since coins are created exclusively by the block reward, the entire ledger can be validated by starting at the first block (known as the *genesis block*) and verifying, for each transaction, that the signature is correct and the sender had the valid balance of coins to send the specified amount. The specific invariants that make a transaction valid are known as the *consensus rules*. Any block that includes transactions that do not conform to the consensus rules are not candidates for the longest chain competition. It is worth noting that since a transaction is not valid until it is included in a block, a user is free to generate many transactions all sending the same coins. However, only one of the these transactions will be included in a block.

2.2.1 Privacy Concerns in Bitcoin

Bitcoin's unique solution to decentralized consensus has proven revolutionary. As of February 2018 the value of the sixteen million bitcoins in circulation exceeds $183 billion. By using a modified version of the Hashcash proof-of-work, any user can independently download and verify every single transaction. Since coins are only created as a reward for a successful proof-of-work demonstration resulting in a new block, the balance of every Bitcoin address is a product of trans-

fers that can be traced back to their minting as a block reward. Thus, each user independently and collectively enforces the correctness of the network, relaying valid transactions and rejecting invalid ones.

However, this radical transparency comes at an unexpected cost. If a Bitcoin address is ever associated with a real-world identity, the sources and destinations of their coins, as well as the balance of their wallet, is laid bare. This is a serious loss of privacy, with potentially catastrophic real-world consequences. For example, suppose Alice is paid her salary in Bitcoin. Later, Alice sends Bob, a coworker, some Bitcoin to pay for coffee. Bob now knows Alice's Bitcoin address. If Bob runs a Bitcoin node, he necessarily knows the total spendable balance of Alice's wallet. In addition, he can see all incoming transactions to Alice's wallet, and their amounts. From this he can surmise the amount of Alice's salary.

Linkability

This loss of privacy can be mitigated by never re-using the same address, although even the original Bitcoin whitepaper notes that such methods offer little real-world privacy against a determined attacker. Several studies of the anonymitity of Bitcoin have shown that analysis can overcome even a dedicated effort at obfuscation [RH11; MPJ+13]. Two transactions are said to be *linked* if they are originate from the same user. If transactions are able to be linked, then a user's activity on blockchain can be tracked, even if mitigations such as multiple addresses are used.

Suppose Alice uses a unique address for every Bitcoin payment she receives. Each address will carry a portion of her total Bitcoin. For example, she may have 0.1 BTC at address X, 0.2 at address Y, and 0.5 at address Z, for a total balance of 0.8 BTC. If Alice wishes to send Bob 0.75 BTC, she will need to use all three of her addresses as inputs to the transaction. Bob, upon receiving the transaction, now knows that X, Y, and Z are all controlled by Alice. Any incoming transactions to these addresses are now linked to Alice.

One method for linking Bitcoin transactions is by observing the *closure* of an address [RW16].

Definition 1. The *closure* of address A contains A itself, and any address B where there exists a transaction that uses A and B as inputs.

Closure membership is an equivalence relation, and thus form a partition on the set of Bitcoin addresses. Knowing a single address controlled by Alice, Bob can calculate its closure, effectively removing the anonymity provided by multiple addresses.

2.3 Witnesses, Knowledge, and Proofs

2.3.1 Interactive Proof Systems

[GMR89] introduced the concept of *interactive proof systems*. In an interactive proof system, two Turing machines share a pair of common tapes over which they can communicate. Each machine shares the same input but is equipped with a unique random tape and a private worker tape. This setup is known as an *interactive protocol*. One machine, the verifier, is limited to a polynomial amount of work on a given input, but may make use of the output of the prover machine in addition to allowing for an arbitrarily small amount of error given a sufficient input length.

Definition 2. Let $L \subseteq \{0,1\}^*$ and (P, V) be an interactive protocol. Then, (P, V) is an *interactive proof system* if:

1. (Completeness) For each k and sufficiently large $x \in L$ as input to (P, V), V accepts with probability at least $1 - |x|^{-k}$.

2. (Soundness) For each k, interactive Turning machine P', and sufficiently large $x \notin L$ as input to (P', V), V halts and accepts with probability at most $|x|^{-k}$.

If such machines exist, then the language is said to be in the interactive polynomial time (IP) class. For IP to be interesting, it should contain some languages not in NP. Otherwise, the interactivity would be merely a novelty; the language's membership in NP guarantees the existence

of a non-interactive polynomial verifier. Several problems not known to be in NP (such as graph non-isomorphism) were shown to be in IP [GMW86], which tells us that IP is more expressive than NP. In a celebrated result, IP = PSPACE, from which we can conclude that when randomization and interaction are allowed, the proofs that can be verified in polynomial time are exactly those proofs that can be generated with polynomial space [Sha92].

Recall the presence of the k error in the above definition, wherein the verifier V will accept strings not in L at a probability at most $\frac{1}{k}$ when connected to an arbitrary "prover" machine P'. Thus, the verifier must be resilient even to a malicious prover with unlimited power, maintaining its soundness guarantee even when facing an adversary with a decided computational advantage.

Definition 3. An interactive proof system is an *interactive argument* if it is an interactive proof system with the soundness guarantee relaxed to

2. (Soundness) For each k, interactive probabilistic polynomial time Turning machine P', and sufficiently large $x \notin L$ as input to (P', V), V halts and accepts with probability at most $|x|^{-k}$.

Interactive arguments, also known as computationally-sound proof systems or argument systems, provide interesting gains in the expressiveness of the proof systems within the class. In particular, all languages in NP have interactive arguments [BCC88]. Moreover, this class of languages with argument systems maintains this property when the complexity of the interactivity is bounded to polylogarithmic size, while interactive proof systems do not [GH96].

It is important to note that interactive arguments merely speak to "convincing" the verifier that $x \in L$. For example, a prover in the problem of circuit satisfiability may convince a verifier that a satisfying assignment exists, but it does not necessarily mean that the prover actually "knows" the satisfying assignment.

Definition 4. An interactive argument (P, V) is a *proof of knowledge* if, for accepting instance $x \in L$, there exists a polynomial time machine E that can extract a witness for w from P.

The above definition lacks mathematical rigor; for a full treatment see [BG92]. For now we will simply say that in addition to being able to convince V, P is truly in possession of some extra knowledge about x.

2.3.2 Zero Knowledge

Intuitively, to say that something is "proved" is to say that there is a sufficiently convincing argument that it is true. For cryptographic applications it is useful if that proof itself can be verified quickly. Formally, a language L is said to be in the NP class if there exists an algorithm P that accepts or rejects in polynomial time a candidate string x given a witness w_x that is polynomial in length of x. Since w_x need not be computable from x in polynomial time, it can be thought of as encapsulating or serializing some large amount of computation on x. For example, the statement "n is not prime" can be accepted for input 1337 given witness factors $(7, 191)$ since it can be quickly verified that $7 \times 191 = 1337$.

For problems such as the decision version of integer factorization, the witness may provide significant information about the underlying search problem (i.e. the factors themselves are used as the witness). Although the IP class is expressive, it is important to ask what information the verifier can extract from the witness string provided by the prover. Can a witness be provided that reveals no information about the underlying solution other than $x \in L$? Intuitively, for an interactive proof system to be "zero-knowledge" a malicious verifier having access to the prover should come away from the exchange with no additional computational ability (in particular any knowledge that would allow the malicious verifier to replicate the prover). [GMR89] formalized this reasoning.

To begin, let $\text{View}_V[P(x) \leftrightarrow V(x)]$ be the record of all interactions between P and V on x, as well as the random tape for V.

Definition 5. Let (P, V) be an interactive proof system for language L. Then, (P, V) is *perfectly*

zero-knowledge if, for all $x \in L$, $h \in \{0,1\}^*$, and probabilistic polynomial time Turing machine V', $\text{View}_{V'}[P(x) \leftrightarrow V'(x,h)] = S(x,h)$, where $S(x,h)$ is an expected probabilistic polynomial time algorithm.

The universal quantifiers in the definition capture the power of zero-knowledge; for all members of L, there exists (some) expected polynomial time algorithm that could recreate the communication between any verifier and the prover give some "extra" bit string h. Thus, the interaction between P and all verifiers reveals no new knowledge since there already exists S that can replicate it in polynomial time. This relaxation is analogous to that from interactive proof systems to interactive arguments; in both cases the existence of adversaries with unbounded power is set aside.

Definition 6. An interactive proof system is *computationally* zero-knowledge if no probabilistic polynomial time Turing machine can distinguish $\text{View}_{V'}[P(x) \leftrightarrow V'(x,h)]$ and $S(x,h)$.

Computational indistinguishability [GM84] is a relaxing of the requirement that the quantities in the proof be identical. Rather, we simply say that there is no efficient Turing machine that can tell the two apart. This relaxing aids in the expressiveness of those problems with zero-knowledge proofs. All NP-complete languages having perfect zero-knowledge proof systems would require a collapse of the polynomial time hierarchy to the second level [For87], which is believed to be unlikely[3]. However, all statements in NP have a computational zero-knowledge proof system [GMW91].

2.3.3 Probabilistically Checkable Proofs

Both NP and IP have interactive verifiers that take polynomial time for some given error $\frac{1}{k}$. Depending on the degree and factors of the polynomial, these verifiers may be prohibitively ex-

[3]While this is not the same as P = NP, it is a step closer

pensive in practice. [BFL+91] showed that every nondeterministic computational task (including interactive arguments) has a verifier that is exceptionally more efficient.

Theorem 7. *Let S be a nondeterministic computational task described in error-correcting code on instance x with witness y. Then, there exists a task S' such that*

1. *S' accepts the same instances as S,*
2. *each instance/witness pair is verifiable in polylogarithmic time, and*
3. *a witness for S' can be computed from a satisfying witness for S in polynomial time.*

Such constructions are known as *probabilistically checkable proofs*, which were formalized by [BFL+91]. By paying a fixed polynomial cost once[4], a witness for P can be constructed that takes polylogarithmic (or "near-linear") time, given a specific encoding of the problem. [FGL+91] considered the modified case of problems in NP with no encoding requirements, specifically giving an approximation algorithm for determining the size of the largest clique in a graph that used polynomial time to verify a logarithmically sized witness. This lead to a new characterization of NP as given in [AS98].

Definition 8. The class NP are those languages whose proofs can be verified in probabilistically polynomial time using a logarthmic number of random bits and a sublogarithmic number of bits from the proof.

2.3.4 Succinctness

Using probabilistically checkable proofs and collision-resistant hashes, [Kil92] detailed a zero-knowledge proof (interactive argument) for circuit satisfiability running in polynomial time giving 2^{-k} error. The construction uses four messages between the prover and verifier. Given sufficiently large problems the system was shown to be more efficient than naive verification.

[4] $|P|^{1+\epsilon}$ for proof P and arbitrary error $\epsilon > 0$

Definition 9. Let x be a string in an NP language L which takes t time to verify membership on the machine M_L. Then, the interactive argument system (P, V) is *succinct* if communication and verification time are $O(\text{poly}(k + |M_L| + |x| + \log t))$ and the proving time is $O(\text{poly}(k + |M_L| + |x| + t))$ where $\text{poly}(n)$ is some fixed polynomial independent of the other parameters.

Since circuit satisfiability is NP-complete, this corresponds to a zero-knowledge proof for all of NP. This performance can be used as a baseline for measuring other zero-knowledge (interactive or otherwise) systems.

CHAPTER 3

zk-SNARKs

We review zk-SNARKs, which are non-interactive, zero-knowledge witnesses for arbitrary problems in NP. To put it another way, zk-SNARKs are digital signatures that prove possession of the input to an arbitrary algorithm which shall produce some desired output. This ability has proven useful for creating succinct and auditable transaction ledgers in cryptocurrencies while maintaining privacy.

3.1 From Proofs to zk-SNARKS

3.1.1 SNARGs

In the previous chapter we reviewed proof systems that are interactive. [Kil92; BG08] demonstrated a four message interactive succinct proof of knowledge for recognizing all languages in NP. This construction, which commits to a probabilistically checkable proof and then shows consistency with a Merkle hash tree, was made non-interactive with one message in the random oracle model [Mic00] by applying the Fiat-Shamir heuristic [FS87]. In the standard model such arguments exist only for a strict subset of NP [BCC+16]. However, this difficulty can be effectively sidestepped by using a two message argument where one of the messages is generated independently from the problem itself [GW11].

Definition 10. Let R be the product of all members of an NP language L and their corresponding witnesses (i.e. $R = \{(x, w) \,|\, x \in L \text{ with witness } w\}$). Let $\Pi = (G, P, V)$ be efficient algorithms

and λ be an arbitrary security parameter. Then, Π is a *succinct, non-interactive argument (SNARG)* if

1. (Completeness) For all $(x, w) \in R$, when $G(\lambda) \to (\sigma, \tau)$ and $P(\sigma, x, w) \to \pi$, the probability that $V(\tau, x, \pi) = 0$ is negligible in terms of $|\lambda|$.

2. (Soundness) For all efficient P', when $G(\lambda) \to (\sigma, \tau)$ and $P'(\sigma, \tau) \to (x, \pi)$, the probability that $V(\tau, x, \pi) = 1$ and $x \notin L$ is negligible in terms of $|\lambda|$.

3. (Succinctness) $|\pi| = \text{poly}(|\lambda|)(|x| + |w|)^{o(1)}$.

The definition for SNARGs bears a close resemblance to that of interactive arguments. The primary difference is the third machine G, and the bounds for the size of the output of P. G is known as the "generator", and given a security parameter λ outputs σ, a common reference string, and τ, a "verification state". Since G relies only on λ (and not x or even L) the generator can be run offline to create parameters (σ, τ). Armed with a witness w for x, the prover takes the common reference string σ and produces a proof π for the statement $x \in L$. Finally, the verifier utilizes the verification state τ to verify the proof π, ultimately being convinced that $x \in L$.

There exist several variants of SNARGs based on the specific soundness condition used. The above definition is an *adaptive, publicly-verifiable* SNARG. A SNARG is adaptive if its soundness guarantee holds even if the adversarial prover P' can choose x; a non-adaptive SNARG would have $P'(\sigma, \tau, x) \to \pi$. Furthermore, a SNARG is publicly-verifiable if P' has access to τ in addition to σ. Otherwise, the SNARG is said to be *designated-verifier*.

A final variant of SNARGs are *preprocessing* SNARGs. Informally, preprocessing SNARGs are those where the generator is permitted to be "expensive". Since generation is generally counted towards the verification time, a *fully-succinct* SNARG is bound by $\text{poly}(\log t)$ for traditional verification time t, while a preprocessing SNARG is bound by $\text{poly}(t)$.

For SNARGs to be secure we must make certain knowledge extractability assumptions. [GW11]

showed that SNARGs cannot be proven secure under any falsifiable assumption via block-box reduction. However, [BCC+16] proved that existence of extractable collision-resistant hash functions are necessary and sufficient conditions for secure SNARGs.

3.1.2 SNARKs

Given secure SNARGs, it makes sense to search for an analogue of proofs of knowledge for interactive arguments. Recall that a proof of knowledge codifies the idea that the prover really "knows" w, defined in terms of an extractor that can determine the witness given the prover. We will use a similar idea, incorporated into a modified soundness requirement.

Definition 11. A *succinct non-interactive argument of knowledge (SNARK)* is a SNARG with the soundness requirement changed to:

2. (Proof of knowledge and soundness) For all efficient P' there is an efficient E such that when $G(\lambda) \to (\sigma, \tau)$, $P'(\sigma, \tau) \to (x, \pi)$, and $E(\sigma, \tau) \to w$, the probability that $V(\tau, x, \pi) = 1$ and $x \in L$ is negligible in terms of $|\lambda|$.

The variants of adaptive/non-adaptive, publicly-verifiable/designed-verifier, and fully-succinct/ preprocessing are comparable for SNARKs and SNARGs.

3.1.3 zk-SNARKs

The last piece of the puzzle is to apply zero-knowledge to SNARKs.

Definition 12. Let (G, P, V) be a SNARK. Then, (G, P, V) is a *perfect zk-SNARK* if there is an efficient simulator S such that for all stateful distinguishers D, whenever $D(\pi) = 1$ and $x \in L$, the probability of $G(\lambda) \to (\sigma, \tau)$, $D(\sigma, \tau) \to (x, w)$, and $P(\sigma, x, w) \to \pi$ is the same as the probability of $G(\lambda) \to (\sigma, \tau, \text{trap})$, $D(\sigma, \tau) \to (x, w)$, and $S(\text{trap}, x) \to \pi$.

To summarize, a zk-SNARK is a SNARG where the prover "knows" the witness w and where the proof π does not give any information that would help an adversary determine what w is. Thus, we have fully realized the vision of an efficient "digital signature" for proving membership in arbitrary languages in NP.

3.2 zk-SNARK Constructions

The definition of a zk-SNARK tells us what a zk-SNARK *is*, but not how one might be constructed. Several such constructions have been developed, ranging from the theoretical to the practical.

3.2.1 Theoretical Constructions

Pairing Based

[Mic00]'s computationally sound proofs were the first appearance of what we now call zk-SNARKs, although the explicit formulation used here had not been defined yet. [Gro10] first improved upon [Mic00] by detailing a preprocessing zk-SNARK for circuit satisfiability in sublinear amount of communication without relying on the random oracle model. The construction uses pairing in bilinear groups to commit to the witness of satisfiability while revealing no information about the witness, assuming the hardness of Diffie-Hellman and knowledge of exponent. Without loss of generality, the circuit C is assumed to consist of NAND gates. From a high level, the scheme works by committing to tuples $(a_1, a_2, \ldots, a_n, b_1, b_2, \ldots, b_n)$, $(b_1, b_2, \ldots, b_n, 0, \ldots, 0)$ and the corresponding outputs $(-u_1, -u_2, \ldots, u_n, 0, \ldots, 0)$ where a_i and b_i are inputs to gate i and u_i is the gate's output[1] for $n = |C|$. Then, commitments are made that show the internal consistency with the of the circuit (e.g. $u_i = -a_i b_i$ for all $i \in n$). The commitments themselves use

[1]For convenience, $+1$ is used for true and -1 is used for false

the Pedersen commitment scheme [Ped92], which provides the zero-knowledge guarantees while maintaining consistency.

The [Gro10] construction provides for trade-offs between time and space complexities ranging from a $\Theta(1)$ argument and $\Theta(|C|^2)$ reference string paired with $\Theta(|C|^2)$ proving to a $\Theta(|C|^{\frac{2}{3}})$ argument and reference string and $\Theta(|C|^{\frac{4}{3}})$ proving. [Lip12] improved the construction's argument and reference string complexity to $\Theta(|C|^{\frac{1}{2}+o(1)})$.

Quadratic Span Based

In an attempt to generalize and improve on the prover complexity of [Gro10] and [Lip12], [GGP+13] introduced the quadratic span programs, with the goal of quasi-linear prover times. Quadratic span programs accept an input whenever a target polynomial can be expressed as a product of two linear combinations of vectors of polynomials.

Definition 13 ([GGP+13])**.** A *quadratic span program (QSP)* Q over field F contains two sets of polynomials $V = \{v_k(x)\}$, $W = \{w_k(x)\}$ for $k \in \{0, \ldots, m\}$, and a target polynomial $t(x)$, all from $F[x]$. Q also contains a partition of the indices $I = \{1, \ldots, m\}$ into two sets $I_{labeled}$ and I_{free}, and a further partition of $I_{labeled}$ as $\cup_{i\in[n],j\in 0,1} I_{ij}$. For input $u \in 0, 1^n$, let $I_u = I_{free} \cup_i I_{i,u_i}$ be the set of indices that "belong" to input u. Q accepts an input $u \in \{0,1\}^n$ iff there exist tuples $(a_1, \ldots, a_m)$ and $(b_1, \ldots, b_m)$ from F^m, with $a_k = 0 = b_k$ for all $k \notin I_u$, such that $t(x)$ divides $\big(v_0(x) + \sum_{k=1}^{m} a_k \cdot v_k(x)\big) \cdot \big(w_0(x) + \sum_{k=1}^{m} b_k \cdot w_k(x)\big)$.

Crucially, it was shown that QSPs and boolean circuits are interchangeable with only a constant overhead. Once converted, the properties of polynomials allow for easy construction of a preprocessing zk-SNARK. Using QSPs, a zk-SNARK for circuit satisfiability was shown using only seven group elements, with the reference string linear in the size of the circuit and prover time quasi-linear. In addition, [GGP+13] also presented a variation of QSPs that work on arithmetic circuits, known as quadratic arithmetic programs, and again showed a zk-SNARK construction.

Definition 14 ([GGP+13])**.** A *quadratic arithmetic program (QAP)* Q over field F contains three sets of polynomials $V = \{v_k(x)\}$, $W = \{w_k(x)\}$, $Y = \{y_k(x)\}$ for $k \in \{0, \ldots, m\}$, and a target polynomial $t(x)$, all from $F[x]$. Let f be a function having input variables with labels $1, \ldots, n$ and output variables with labels $m - n' + 1, \ldots, m$. We say that Q is a QAP that computes f if the following is true: $a_1, \ldots, a_n, a_{m-n'+1}, a_m \in F^{n+n'}$ is a valid assignment to the input/output variables of f iff there exist $(a_{n+1}, \ldots, a_{m-n'}) \in F^{m-n-n'}$ such that $t(x)$ divides $\big(v_0(x) + \sum_{k=1}^{m} a_k \cdot v_k(x)\big) \cdot \big(w_0(x) + \sum_{k=1}^{m} a_k \cdot w_k(x)\big) - \big(y_0(x) + \sum_{k=1}^{m} a_k \cdot y_k(x)\big)$.

Fully Succinct

The previously discussed constructions are all preprocessing zk-SNARKs. The more general type, fully succinct zk-SNARKs (whose provers are linear only in the security parameter λ), have also been explored. [BCC+13] gave a general construction for converting preprocessing SNARKs into fully succinct SNARKs. Using *incrementally-verifiable computation*, several SNARKs are composed that effectively "prove" the steps taken during the extra preprocessing phase. By doing this, asymptotically fully succinct SNARKs are given.

3.2.2 Practical Implementations

Pinocchio

Although constructions of zk-SNARKs were known in the literature, a fully implemented end-to-end system for describing a problem and generating a zk-SNARK for it remained open until [PGH+13]. Towards this end, [PGH+13] created a system capable of compiling a subset of C programs into a structure that a specific zk-SNARK could verify. Although the subset was fairly strict[2], the ability to ingest regular C code made SNARKs generally accessible to cryptographic implementors.

[2]Non-self-modifying with fixed memory access and compile-time constants for loops and array access

The [PGH+13] system consists of two parts. The first part is a compiler, qcc, which converts the supported C subset to an arithmetic circuit. Recall that an arithmetic circuit is akin to boolean circuits with the gates of logical AND, OR, etc. replaced with mathematical operations such as addition and multiplication. Through various techniques all of the fundamental operations in C are modeled using arithmetic gates[3]. Once converted, a quadratic arithmetic program is constructed for the arithmetic circuit. The second portion of the system is a zk-SNARK for QAPs, the primary operation of which is exponentiation in an elliptic curve group.

From a practical standpoint the [PGH+13] system was the first practical end-to-end system for using zk-SNARKs in software. The performance was shown to be orders of magnitude faster than previous constructions: a sample problem (multivariate polynomial evaluation) is quoted as having 41 second generation time, 246 second proof creation time, and 12 millisecond verification time.

TinyRAM

The construction [PGH+13] supported a large subset of C from a semantic standpoint, but the programs themselves could not have any data dependencies, e.g. a conditional on some calculated value derived from the input. To formalize an environment for such computations to take place, [BCG+13b] created the TinyRAM architecture, an idealized random-access machine equipped with a fixed word size and number of registers, a program counter and conditional flag, and addressable memory. The instructions in TinyRAM are those one would generally expect in a RISC architecture: loads, stores, compares, jumps, and so on. In addition, a special instruction signifies the machine has reached an accepting state and should terminate.

The [BCG+13b] system uses a modified version of gcc to generate TinyRAM instructions from C. Then, as with [PGH+13], a conversion to an arithmetic circuit is performed, although the conversion in [BCG+13b] is much more complicated to account for the consistency of the memory. From the TinyRAM instructions a routing network with constraints is created, and the generated

[3]Surisingly, is was more efficient to still use QAPs for C's boolean operations

arithmetic circuit verifies that the constraints are met for a particular input.

Once the arithmetic circuit is constructed, all that remains is to pass it into a zk-SNARK for satisfiability. The SNARK presented is a modified version of that found in [BCI+13], which itself generalizes the quadratic arithmetic programs found in [GGP+13].

vnTinyRAM

Both [PGH+13] and [BCG+13b] require that the generation step be rerun for each different program. This generation step is not trivial, and accounts for a significant portion of the overall time the system takes to execute. While the reference string and verification state can be reused for different inputs to the same program, expensive work is required whenever a new program is used. [BCT+14] created a universal circuit generator which is bound only by the maximum number of instructions in a program l, the maximum input size n, and some specific time bound t. This "once-and-for-all key generation" can be used to verify all programs up to a given size. The size of the generated circuit is $O((l + n + t) \cdot \log(l + n + t))$, which means that the circuit grows essentially linearly in all three inputs.

In addition to a universal circuit generator, [BCT+14] operates on a more powerful machine than [BCG+13b]: vnTinyRAM. vnTinyRAM is an extension of TinyRAM that allows for self-modifying code, and can be thought of as an idealized von Neumann machine. As was done previously, a modified version of gcc takes C programs and outputs vnTinyRAM instructions. vnTinyRAM also switches to byte-addressable (as opposed to word-addressable) memory.

Following the familiar construction, the [BCG+13b] system converts the intermediate machine instructions into an arithmetic circuit. Then, a zk-SNARK for arithmetic circuits provides the proof and verification mechanisms for the program execution. Rather than construct a new zk-SNARK in its entirety, [BCT+14] provided several tailored optimizations to the SNARK found in [PGH+13].

Detailed complexities in time and space are provided, and apples-to-apples comparison with previous implementations showed modest performance gains. For example, a one-million gate

circuit with a one-thousand bit input with 128 bit security had a generation time of 117 seconds, a proving time of 147 seconds, and a verification time of 5 milliseconds. In addition, all proofs are 288 bytes regardless of the program or input.

The correctness of the optimizations made in [BCT+14] relies on an unproven lemma presented in the paper. [Par15] showed that this lemma is incorrect, and the efficiency gains were (at least theoretically) incorrect. More seriously, it was shown how to create invalid proofs that would be accepted by the verification algorithm. Since the publication of [Par15], modifications have been made to open source implementations of vnTinyRAM that correct the flaw, with negligible performance implications[4].

The libsnark project is a free, open source software library for constructing zk-SNARKS (and regular SNARKs) via a variety of the s presented in literature. The zk-SNARK system used by libsnark is the that presented in [BCT+14] (with the correction from [Par15]).

[4]See https://github.com/scipr-lab/libsnark/commit/af725eeb

CHAPTER 4

Zcash

For a cryptocurrency to be trustworthy, a user must be able independently verify the correctness of transactions. In Bitcoin, this ability is achieved via a publicly auditable transaction ledger. Every transaction is fully transparent; the source and destination addresses as well as the amount of the transaction are "in the clear". If a real world identity is associated with a Bitcoin address then a serious loss of privacy can occur. Solutions for solving Bitcoin's privacy problems range from cycling through many addresses, to "mixers", where several users send coins to one address which are subsequently pooled and combined before being returned.

Since Bitcoin's inception several new cryptocurrencies have been developed that attempt to include privacy features directly into the protocol of the coin, the most popualar of which are Zcash[1] and Monero[2]. The central challenge for these coins is maintaining correctness (e.g. that coins are not double spent) while ensuring the privacy of the users engaging in the transactions. Zcash uses zk-SNARKs to enforce this property.

4.1 Zerocash

The essential operation of a cryptocurrency is the transaction, where some amount of "coins" are sent from Alice to Bob. Afterwards, Bob is free to send these coins to someone else, but Alice must not be able to "respend" the coins. Zerocash [BCG+14] provides privacy and fungibility by

[1] https://z.cash/
[2] https://getmonero.org/

encoding these constraints as an arithmetic circuit with a zk-SNARK proof for the circuit appended to the blockchain as a record of the transaction, which can be efficiently verified by anyone.

Zerocash defines the concept of a "decentralized payment scheme", and then shows how zk-SNARKs can be used to build such a system.

Definition 15. A *decentralized payment scheme* (DAP) is a set of polynomial algorithms with the following properties

1. *Setup.* Generates a set of public parameters used by the remaining algorithms. Setup must be run by a trusted party.

2. *CreateAddress.* From the public parameters, generate a public address to which coins can be sent, and a corresponding private key which will allow coins sent to the public address to be spent.

3. *Mint.* Creates new coins and logs their creation on the blockchain.

4. *Pour.* Transfers the value from input coins to new outputs coins and logging the transaction on the blockchain, optionally revealing their amounts. Allows subdiving, merging, and transferring coins.

5. *Verify.* Verify that a transaction log is correct.

6. *Receive.* Determines the balance of unspent coins for a particular address based on all transactions saved on the blockchain.

In addition, the DAP must be complete, which means that any unspent coin is able to spent, and secure, which means that it maintains ledger indistinguishability, transaction non-malleability, and balance correctness.

The DAP operations (Mint, Pour, etc.) can be generalized to nearly any cryptocurrency. The security guarantees of ledger indistinguishability (no information is provided by transactions other

than that which is strictly public), and transaction non-malleability (transactions cannot be modified and still be valid) provide the core differentiator as an anonymous payment system. The balance correctness guarantee (cannot spend more coins that one's unspent balance) is a necessity for any viable cryptocurrency. Zerocash is an implementation of a DAP which makes extensive use of zk-SNARKs in its operations.

The Pour operation, where coins are combined, subdivided, and moved is the essential core of Zerocash, as it facilitates the transfer of value between users. An instance of a Pour operation involves the certain public commitments generated from the coins to be poured. A witness for Pour are the private keys for the coins, as well as the specific amounts, sources, and destinations. To maintain privacy protections, the witness must remain secret. To facilitate this a zk-SNARK for Pour was created.

The program or algorithm for the Pour zk-SNARK accepts if the witness is valid for the given instance under the rules of the DAP. The actual zk-SNARK used is the QAP zk-SNARK from [BCT+14]. However, rather than generate the circuit for the zk-SNARK via a high-level language (e.g. the vnTinyRAM construction in [BCT+14]), Zerocash uses a much more efficient "hand-designed" circuit. The dominating operation in the circuit is the SHA-256 hash function. An arithmetic circuit for SHA-256 was created "from scratch" which consisted of around 28,000 gates. For comparison, implementing the same logic in C and "compiling" to TinyRAM via the method detailed in [BCG+13b] resulted in a circuit with $5.7 \cdot 10^6$ gates.

The circuit for the Pour zk-SNARK is constant. Thus, the same reference string and verification state will be used in every Pour. The Setup procedure generates these public parameters (the prepocessing phase for the zk-SNARK). Then, to create a transaction the zk-SNARK's proof generation is performed using the witness which requires information only the valid coin owner should know. The output of the proof generation is the proof π (only 288 bytes), and is appended to the blockchain to record the transaction. The correctness of π can be efficiently verified by any user of the system by performing the zk-SNARK's verification operation. While the circuit itself ensures

balance correctness, the inherent properties of zk-SNARKs confer ledger indistinguishability and transaction non-malleability.

4.2 The Zcash Cryptocurrency

The Zerocash system described in [BCG+14] has been implemented as a real-world cryptocurrency known as Zcash. Rather than trust a single party to honestly generate the public parameters, the Zcash creators designed a multi-party parameter generation scheme. This method was later formalized in [BCG+15]. Six people took part in the parameter generation, and the system is secure if at least one of the parties was honest.

Zcash began as a fork of the Bitcoin Core codebase. As with Bitcoin, Zcash coins are sent to addresses, which can be thought of as public keys. Only the owner of the corresponding private key can generate a new transaction to transfer the coins. In Zcash there are two classes of addresses: transparent (*t-addrs*) and shielded (*z-addrs*). When a transaction occurs between transparent addresses the transfer is akin to Bitcoin transactions, where the parties involved and the amounts are fully visible.

Shielded addresses provide a mechanism to obscure the source, destination, and amounts of transactions. Only when a transaction occurs between two shielded addresses are these privacy guarantees in full effect. These transactions are known as *shielded* transactions, the use of which are optional.

The privacy of shielded transactions is achieved by using zk-SNARKs. Rather than signing a transaction with a private key, to transfer coins from a shielded address the owner produces a zk-SNARK proof. This is a proof showing that the sender is in possession of a satisfying assignment to a program encoded as an arithmetic circuit that enforces the protocol's correctness. Zcash's privacy guarantees are provided by the zero-knowledge properties of the proof.

Transactions involving z-addrs are carried out in the JoinSplit structure, which is a new structure

Table 4.1: The three types of operations performed by a JoinSplit

	Shielding	**De-shielding**	**Shielded**
Source	t-addr	z-addr	z-addr
Destination	z-addr	t-addr	z-addr
Amount	Public	Public	Private

added to the Bitcoin transaction format. A JoinSplit contains three essential fields: the number of coins entering the shielded pool (known as vpub_old), the number of coins exiting the shielded pool (known as vpub_new), and a field holding a zero-knowledge proof attesting to the legitimacy of the transaction.

There are three different operations that a JoinSplit may perform (see Table 4.1). The first is a *shielding transaction*, where coins are sent from a t-addr to a z-addr. This corresponds with a non-zero vpub_old. Thus, in shielding transactions the amount being sent to a z-addr is public, but the z-addr itself is not. The second operation is a *de-shielding transaction*, where coins are sent from a z-addr to a t-addr. This corresponds with a non-zero vpub_new. Likewise, the z-addr remains private but the amount is public. The final operation is a *shielded transaction*, where coins are transferred between two z-addrs. For shielded transactions both addresses are private, as well as the amounts.

CHAPTER 5

Linkability in Zcash

Zcash retains nearly all of the original Bitcoin functionality. Transactions between transparent addresses are essentially equivalent in form to Bitcoin transactions. In Bitcoin, coins are transferred by referencing the outputs of previous transactions and providing a digital signature that proves ownership of the address the coins were sent to. Since coins are only created as part of the block reward for miners, it is straightforward to audit the correctness of every transaction: simply trace back each transaction output, verifying digital signatures along way back to a block reward.

As previously discussed, the radical simplicity of Bitcoin's transparent ledger has significant privacy drawbacks. If an address is ever associated with a real-world identity it becomes trivial to trace the source and destination of all the user's coins. This loss of privacy can be mitigated by never re-using the same address, although even the original Bitcoin whitepaper notes that linking will remain unavoidable. Thus, when transferring between two t-addrs in Zcash, the transaction is fully linkable.

We discovered that many transactions involving shielded addresses exhibit a particular pattern: first, coins are sent from a transparent address to a shielded address. Soon after, an identical or nearly identical amount of coins are sent back to a transparent address. We call such transactions *round-trip transactions* and argue that the controlling party is likely the same for both transactions. In this chapter we show that 31.5% of all coins sent to shielded addresses are likely involved in a round-trip transaction, potentially removing their unlinkability. This result was shared privately

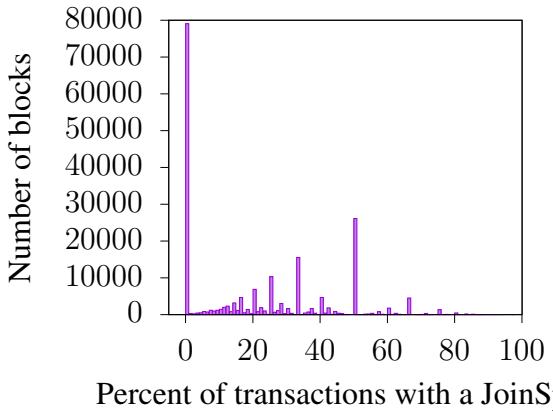
with the Zcash Company before publication[1].

5.1 Shielded Transactions

The anonymity of z-addrs is the fundamental differentiator between Zcash and Bitcoin. We examine several metrics of this anonymity as it pertains to real-world usage of Zcash. All analysis performed was on the Zcash blockchain ending in block 196304, which was mined on October 4, 2017.

5.1.1 Transaction Metrics

Figure 5.1: Transactions in each block that contain at least one JoinSplit



Overall, 19.6% (259,220) of all Zcash transactions contained a JoinSplit operation, with 40.2% of blocks containing no transactions with JoinSplits (see Figure 5.1). Thus, 80.4% of all Zcash transactions are trivially linkable back at least one transaction.

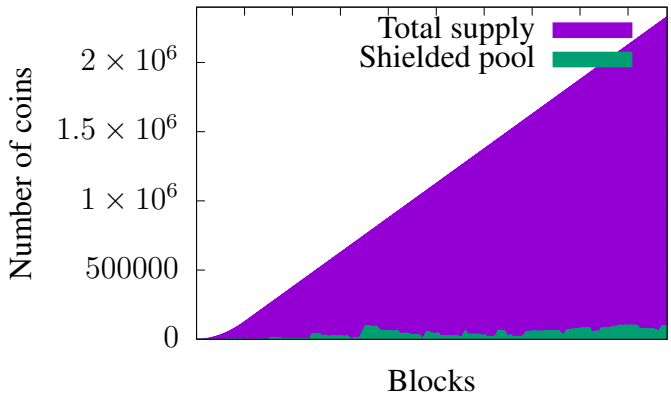
Of the 19.6% of transactions that do contain a JoinSplit, we further classified the transactions into the types of JoinSplit operations being performed. Every JoinSplit may contain an unknown number of shielded transactions, so there is no way to place an upper bound on the number of shiel-

[1]The official Zcash Company response can be found at https://blog.z.cash/new-research-on-shielded-ecosystem/

ded transactions. However, only 1.9% (5,450) of all JoinSplits have no shielding or deshielding portion, while 40.6% (116,743) of JoinSplits included a shielding operation and 57.5% (165,394) included a deshielding operation[2]. This evidence supports our eventual hypothesis that most coins sent to z-addrs are sent back to t-addrs.

5.1.2 Coin Metrics

Figure 5.2: Size of shielded pool compared to total supply of Zcash coins



We also investigated the participation of coins in shielded transactions. As with Bitcoin and other cryptocurrencies, transactions are grouped into blocks, and miners must solve a difficult proof-of-work problem on the block and announce it to the network. Valid blocks reward the miner with a certain number of coins in what is known as the block reward. Thus, with each block the available supply grows slightly, until some fixed[3] cap.

In Zcash, coins exist either in the *transparent pool*, where they are controlled by a t-addr, or the *shielded pool*, controlled by a z-addr. To determine the size of the shielded pool, the difference between vpub_old and vpub_new of each JoinSplit is calculated. This is the net differential into the shielded pool. The total pool size is kept as a running sum of the differential for each block. Our

[2]No JoinSplit included both a shielding and a deshielding operation. It is unclear if the Zcash software allows this
[3]21 million for Bitcoin and Zcash

results (see Figure 5.2) show that the shielded pool only ever contains a relatively small percentage of the overall supply of Zcash coins[4]. For the final block cataloged, the percentage of coins in the shielded pool was 4.3%. The average percentage per block was 3.5%.

5.1.3 Discussion

Taken together, the previous results suggest an environment where transactions using z-addrs are infrequently used. In an environment where shielded transactions (i.e. z-addr to z-addr and fully private) are commonplace, it is reasonable to assume that many to most JoinSplits would contain no shielding or deshielding operation. However, out of the 287,587 JoinSplits cataloged, only 5,450 met these conditions. Furthermore, only a relatively small percentage of coins are ever present in the shielded pool. Together, these two conditions lead us to hypothesize that when coins are introduced to the shielded pool, they are soon returned back to the transparent pool.

Why are z-addr transactions so rare? We suggest that the reason is likely practical: most wallet programs do not support z-addrs. According to the Zcash community website[5] no web-based wallets support z-addrs. In addition, all cryptocurrency exchanges that trade Zcash only accept t-addrs. For users that wish to take advantage of the privacy features afforded by z-addrs while still maintaining the practical utility of t-addrs, a simple solution would be to send coins to a z-addr, and then send them back to a t-addr. This would have the effect of removing the linkability of the transactions. However, because the amounts of the shielding and deshielding operations are public, these types of transactions may still be linkable.

[4]For coin totals, only the whole number portion of coins are reported

[5]https://www.zcashcommunity.com/wallets/ accessed on November 1, 2017

5.2 Round-trip Transactions

In this section we use the public amount of shielding and deshielding operations to build a candidate set of linkable transactions that pass through the shielded pool. The official Zcash website notes[6] that such analysis is possible. However, we believe this study is the first to perform the analysis and show the prevalence of such transactions.

5.2.1 Defining

Definition 16. Let j_1 and j_2 be JoinSplits included in blocks of height h_1 and h_2 with $h_2 > h_1$. If there exists exactly one pair of transactions (t_1, t_2) with $j_1 \in t_1$, $j_2 \in t_2$, and vpub_new$(j_2) =$ vpub_old(j_1), then we say (t_1, t_2) is a *round-trip transaction* (RTT).

RTTs are an ordered pair of transactions containing JoinSplits where the shielding amount in the first transaction is equal to the deshielding amount in the second transaction. In addition, the second transaction must appear in a later block than the first transaction. Lastly, we are only concerned about those pairs where there is exactly one pair that fits the criteria, e.g. if t_1 shields 10.1234 coins, we are looking if there is exactly one JoinSplit in any later block that deshields 10.1234 coins.

We note that although there is strong circumstantial evidence that t_1 and t_2 in a RTT are linked, the conclusion is not definitive. Not all RTTs identified by our heuristic are actually linked; the match may be coincidental.

5.2.2 Methodology

To detect RTTs, we first modified[7] an existing Bitcoin blockchain parser to support Zcash. We then used this to build a relational database of the blockchain, linking blocks, transactions, and

[6] https://z.cash/blog/transaction-linkability.html accessed on November 2, 2017

[7] We published the source code on the open source collaboration site Github. https://github.com/jquesnelle/ZcashBlockchain https://github.com/jquesnelle/ZcashDatabaseGenerator

Figure 5.3: Pseudocode for SQL query to find potential round-trip transactions

```
SELECT *
FROM JoinSplits E,F
(
   SELECT *
   FROM JoinSplits G,H
   WHERE G.vpub_old > 0
      AND G.vpub_old = H.vpub_new
      AND H.Block > G.Block
   GROUP BY G.vpub_old
   HAVING COUNT(G.Id) = 1
) I
WHERE E.vpub_old = F.vpub_new
   AND E.vpub_old = I.vpub_old
```

JoinSplits. Figure 5.3 gives the pseudocode for a query to find JoinSplits that meet the criteria for RTTs.

5.2.3 Results

Our analysis found 10,075 transaction pairs that met our criteria (see Table 5.1), transferring a total of 919,220 coins. The total number of all coins entering the shielded pool was 2,911,734. Thus, we determined that 31.5% of all coins entering the shielded pool were involved in a RTT.

We could not conclusively determine that all RTTs are linked transactions. However, there is strong circumstantial evidence that the false positive rate of our heuristic is low, and that most of the transactions are indeed linked. 96% (9673) of our matches involved transactions which appeared within two hours of each other on the blockchain. In addition, by the definition of RTTs, the amounts being shielded and unshielded are globally unique amongst the entire history of Zcash. Given the high divisibility[8] of Zcash coins, we believe that a single exact match of the shielding and deshielding amounts occurring within a few hours is strong evidence that the transactions are

[8] A Zcash coin can be divided out to 8 decimal places

linked.

Table 5.1: Round-trip transactions by block time difference in minutes

Δ block time	# RTT	Σ coins
$[0, 5)$	1373	156,237
$[5, 15)$	5022	421,021
$[15, 30)$	1479	147,546
$[30, 60)$	1015	95,034
$[60, 120)$	500	35,741
$[120, 1440)$	284	60,518
$[1440, \infty)$	402	3,120

Table 5.2: Top JoinSplits by vpub_old that are part of a round-trip transaction

Top n JoinSplits	# RTT	Σ coins
10	10	34,153
50	49	143,924
250	236	500,163
500	460	765,212
1000	585	834,301

Large denomination transfers were particularly likely to be RTTs (see Table 5.2). Of top 250 JoinSplits by shielding amount, 236 were part of a RTT. Upon further investigation[9] it was discovered that many of these large JoinSplits were Zcash mining pools sending their block rewards to a z-addr before distributing the coins to the t-addrs of miners. In this case, miners may be under the impression that source of their coins is private since they are receiving their payout from the pool via a z-addr. However, because the pool is engaging in RTTs, we were able to determine the t-addrs of the pool's members.

5.2.4 Fee-adjusted RTTs

When creating a Zcash transaction, the sender may choose to offer a small amount of Zcash as an incentive for miners to include the transaction in a block. This is known as a *fee*. Although fees

[9]The t-addrs of Zcash miners are known

Table 5.3: Most common fees for non-coinbase transactions

Fee	# tx	%
0.0001	523,036	46.40%
0.001	34,203	3.03%
0.0002	33,662	2.99%
0.00009	30,400	2.70%
0.00005	24,127	2.14%
0.00000226	23,679	2.10%
0	16,154	1.43%

Table 5.4: 1-fee round-trip transactions

Fee	# RTT	Σ coins
0.0001	85	1,278
0.001	149	1,360
0.0002	143	1,400
0.00009	2	20
0.00005	9	20

are not required, 98.6% of all non-coinbase[10] Zcash transactions include a fee. Table 5.3 shows the most common fees. The fee used most frequently was 0.0001 Zcash, which 46.4% of transactions used.

For RTTs, vpub_new $=$ vpub_old. However, if the party performs any shielded transactions (z-addr to z-addr) before transferring back to the transparent pool, the vpub fields may not match, since the shielded transactions may have also paid fees. We relax our definition of an RTT to vpub_new $=$ vpub_old $- f$ where f is some combination of common fees. We call such transactions *fee-adjusted round trip transactions*.

For fee-adjusted RTTs we considered only transactions appearing within 24 hours of each other. In addition, to limit false positives only the five most common fees were used. We first searched for 1-fee RTTs, which attempts to detect the following pattern: t-addr $\rightarrow$ z-addr $\stackrel{\text{fee}}{\rightarrow}$ z-addr $\rightarrow$ t-addr.

Table 5.4 gives the results for 1-fee RTTs. A total of 388 such transaction pairs were found,

[10]The first transaction in a block is the *coinbase* transaction, which specifies the receiver of the block reward. Coinbase transactions do not have fees

Table 5.5: 2-fee round-trip transactions

Fee	# RTT	Σ coins
0.002	146	1,305
0.0012	131	1,264
0.0011	29	167
0.00109	0	0
0.00105	2	18
0.0004	137	1,316
0.0003	19	111
0.00029	3	30
0.00025	4	30
0.00019	7	80
0.00018	1	10
0.00015	3	40
0.00014	3	40

accounting for a total of 6,058 coins. Given the apparent scarcity[11] of shielded transactions, it is unsurprising that relatively few were found.

We also searched for 2-fee RTTs, attempting to detect the following: t-addr $\rightarrow$ z-addr $\overset{\text{fee}}{\rightarrow}$ z-addr $\overset{\text{fee}}{\rightarrow}$ z-addr $\rightarrow$ t-addr. We considered combinations of the five most common fees, except for those which summed to a common fee (e.g. $0.0001 + 0.0001 = 0.0002$). A total of 485 transaction pairs were found, accounting for a total of 4,411 coins. For a list of sample round trip-transactions, see Appendix A.

[11] Recall that only 1.9% (5,450) JoinSplits have no shielding or deshielding operation

CHAPTER 6

Conclusion

Bitcoin's transparent ledger allows the source and destination of coins to be traced by any third party. This greatly simplifies verification of the consensus chain, since it is straightforward to verify that the amounts and owners of each transaction are correct through simple addition and signature checking. However, if a Bitcoin address ever become associated with an identity, this linking can result in a significant loss of privacy.

To prevent this linking, Zcash employs shielded transactions to obscure these details. The specific construction used to create this obfuscation are zk-SNARKs, which provide general zero-knowledge proofs for arbitrary statements in NP. For Zcash, a specific fixed statement is used, namely an arithmetic circuit encoding that correctness of the necessary invariants for a viable currency.

Zcash itself has two classes of addresses: t-addrs, which are similar to Bitcoin addresses and do not use zk-SNARKs, and z-addrs, which do. However, only transfers between two z-addrs are truly private. Empirical evidence suggests that most usage of z-addrs involves shielding or deshielding operations, where the amount transferred is still public. We have shown that a third party can use this information to link entries and exits from the shielded pool. In our experiments we were able to identify 31.5% of all shielded pool coins as likely being involved in round-trip transactions.

When privacy features are optional, users often take the path of least resistance. Given the large computational costs of shielded transactions, they are relatively rare. Round-trip transactions may be an effort to "have the best of both worlds", but if used incorrectly they do not deter a

determined attacker from linking transactions. To be entirely sure that the source of coins cannot be traced, a user must perform a second, fully shielded transaction after receiving coins in a shielding operation. If they wish to return the coins to the transparent pool, they must also take care to leave some portion remaining in the shielded pool to avoid performing a RTT.

Lastly, we highlight a real-world case where RTTs can lead to a serious privacy loss. Many of the coins involved in RTTs come from mining pools distributing the block reward proportionally to users based on their contribution to the pool. Several of the popular Zcash pools perform an RTT before actually distributing the reward. It may be that the pool's members do not wish for it to be known that they are engaged in mining. By virtue of receiving coins from a z-addr, they may believe that the source is obscured. However, by identifying RTTs, the true source of their coins is revealed, which may have serious repercussions.

APPENDIX A

Sample Round-trip Transactions

t_1 and t_2 are the transaction identifiers, which can be looked up on any Zcash block explorer website[1]. vpub_old and vpub_new are the matching shielding and deshielding amounts. Δ block time is the difference in time between the block where the first transaction was confirmed and the block where the second transaction was confirmed.

A.1 Zero-fee

t_1: a2c9f7ad3b1993c40e692da61966f8633d85cb96c07b8810c6b14493978f2b46

t_2: ab3b717b85a64541c6d4bb2da8c0806da9666fa1979e0f640c7f49c44fea3bca

vpub_old: 3479.51898254

vpub_new: 3479.51898254

Δ block time: 2 minutes

t_1: d4e0047df31d0e1c8a7d311064314a74c43d0677ffcc430f8d093bb1867dd21b

t_2: b63f4948b405b91c28bd59affc06e12aa8e126cb1f101ab36e1114ee882bb0b3

vpub_old: 12.14981195

vpub_new: 12.14981195

Δ block time: 3 minutes

[1]For example, https://explorer.zcha.in/ or https://zcash.blockexplorer.com/

t_1: a6c87c8e2f20b729a33fec7031b2ead3ec6a001e4aa4c575207c44f2690870e4

t_2: 9f300ecfdfb6a8658f34bd469d74f401dd7233d7a610cb91faaeb4a2b3fdc299

vpub_old: 3.77326919

vpub_new: 3.77326919

Δ block time: 928 minutes

t_1: 709e38ab58148f6b2a3eb56621ea502790270386b7c6648baf06a510cf48efaa

t_2: 9f300ecfdfb6a8658f34bd469d74f401dd7233d7a610cb91faaeb4a2b3fdc299

vpub_old: 220.01805591

vpub_new: 220.01805591

Δ block time: 15 minutes

A.2 1-fee

t_1: 2641aeece9df50c5275b692a20da6f900a1a42440adc454765d7f3e6a1b1aeef

t_2: 4d83b22ab6967c83f11e4cb6f417623c553364ddc5c8d658027356bc28fa6f1a

vpub_old: 0.67209594

vpub_new: 0.67199594

f: 0.0001

Δ block time: 8 minutes

A.3 2-fee

t_1: 84a11d9794e0eb318327dd960b7bfa4e1146855fcb1f0aaf6eb40ceadaf9ecbb

t_2: 855e94b007d66f1ee283374c91b559d02fa397079d6f9b5b9012a668680efd71

vpub_old: 6.3805

vpub_new: 6.3794

f: $0.0001 + 0.001 = 0.0011$

Δ block time: 35 minutes

Bibliography

[AS98] Sanjeev Arora and Shmuel Safra. "Probalistic checking of proofs: a new characterization of NP". In: *Journal of the ACM* 45.1 (1998), pp. 70–112.

[Bag09] Phillip Bagus. "The quality of money". In: *Quarterly Journal of Austrian Economics* 12.4 (2009), pp. 22–45.

[BBF+15] Michael Backes, Manuel Barbosa, Dario Fiore, and Raphael Reischuk. "ADSNARK: nearly practical and privacy-preserving proofs on authenticated data". In: *Proceedings of 36th IEEE Symposium on Security and Privacy (S&P '15)*. 2015, pp. 271–286.

[BCC+13] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. "Recursive composition and boostrapping for SNARKs and proof-carrying data". In: *Proceedings of the 45th Annual ACM Symposium on Theory of Computing (STOC '13)*. 2013, pp. 111–120.

[BCC+16] Nir Bitansky, Ran Canetti, Alessandro Chiesa, Shafi Goldwasser, Huijia Lin, Aviad Rubinstein, and Eran Tromer. "The Hunting of the SNARK". In: *Journal of Cryptology* (2016), pp. 1–79.

[BCC88] Gilles Brassard, David Chaum, and Claude Crépeau. "Minimum disclosure proofs of knowledge". In: *Journal of Computer and System Sciences* 37.2 (1988), pp. 156–189.

[BCG+13a] Eli Ben-Sasson, Alessandro Chiesa, Daniel Genkin, and Eran Tromer. "On the concrete efficiency of probabilistically-checkable proofs". In: *Proceedings of the 45th Annual ACM Symposium on Theory of Computing (STOC '13)*. 2013, pp. 585–594.

[BCG+13b] Eli Ben-Sasson, Alessandro Chiesa, Daniel Genkin, Eran Tromer, and Madars Virza. "SNARKs for C: verifying program executions succinctly and in zero knowledge". In: *Proceedings of the 33rd Annual International Cryptology Conference (CRYPTO '13)*. 2013, pp. 90–108.

[BCG+14] Eli Ben-Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. "Zerocash: Decentralized Anonymous Payments from Bitcoin". In: *Proceedings of the IEEE Symposium on Security and Privacy*. 2014, pp. 459–474.

[BCG+15] Eli Ben-Sasson, Alessandro Chiesa, Matthew Green, Eran Tromer, and Madars Virza. "Secure sampling of public parameters for succinct zero knowledge proofs". In: *Proceedings of the IEEE Symposium on Security and Privacy*. 2015, pp. 287–304.

[BCI+13] Nir Bitansky, Alessandro Chiesa, Yuval Ishai, Rafail Ostrovsky, and Omer Paneth. "Succinct non-interactive arguments via linear interactive proofs". In: *Proceedings of the 10th Theory of Cryptography Conference (TCC '13)*. 2013, pp. 315–333.

[BCT+14] Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza. "Succinct non-interactive zero knowledge for a von Neumann architecture". In: *Proceedings of the 23rd USENIX security symposium (Security '14)*. 2014, pp. 781–796.

[Bec97] Adam Beck. *A partial hash collision based postage scheme*. 1997. URL: `http://www.hashcash.org/papers/announce.txt` (visited on 10/12/2017).

[Ben17] Eli Ben-Session. *Zerocash, Bitcoin, and transparent computational integrity*. 2017. URL: `https://cyber.stanford.edu/sites/default/files/elibensasson.pdf` (visited on 05/31/2017).

[BFL+91] László Babai, Lance Fortnow, Leonid A. Levi, and Mario Szegedy. "Checking computations in polylogarithmic time". In: *Proceedings of the 23rd Annual ACM Symposium on Theory of Computing (STOC '91)*. 1991, pp. 21–31.

[BG08] Boaz Barak and Oded Goldreich. "Universal arguments and their applications". In: *SIAM Journal on Computing* 38.5 (2008), pp. 1661–1694.

[BG92] Mihir Bellare and Oded Goldreich. "On defining proofs of knowledge". In: *Advances in Cryptology (Crypto 92) Proceedings*. 1992.

[Bul10] James Bullard. *Quantitative easing: Uncharted waters for monetary policy*. Federal Reserve Bank of St. Louis. 2010. URL: `https://www.stlouisfed.org/~/media/Files/PDFs/publications/pub_assets/pdf/re/2010/a/presidents-message.pdf` (visited on 10/09/2017).

[CFN90] David Chaum, Amos Fiat, and Moni Naor. "Untraceable electronic cash". In: *Lecture notes in computer science* 403 (1990), pp. 319–327.

[Cha82] David Chaum. "Blind signatures for untraceable payments". In: *Advances in Cryptology: Proceedings of Crypto 82*. 1982, pp. 199–203.

[Dai98] Wei Dai. *b-money*. 1998. URL: `http://www.weidai.com/bmoney.txt` (visited on 10/12/2017).

[DH76] Whitfield Diffie and Martin Hellman. "New directions in cryptography". In: *IEEE Transactions on Information Theory* 22.6 (1976), pp. 644–654.

[Dou02] John Douceur. "The Sybil attack". In: *International Workshop on Peer-to-Peer Systems*. 2002.

[Eic10] Barry Eichengreen. *Exorbitant privilege: The rise and fall of the dollar and the future of the international monetary system*. Oxford University Press, 2010.

[FGL+91] Uriel Feige, Shafi Goldwasser, László Lovász, Shmuel Safra, and Mario Szegedy. "Approximating clique is almost NP-complete". In: *Proceedings of the 32nd Annual IEEE Symposium on Foundations of Computer Science*. 1991, pp. 2–12.

[For87] Lance Fortnow. “The complexity of perfect zero-knowledge”. In: *Proceedings of the 19th Annual ACM Symposium on Theory of Computing (STOC ’87)*. 1987, pp. 204–209.

[FS87] Amos Fiat and Adi Shamir. “How to prove yourself: practical solutions to identification and signature problems”. In: *Proceedings of the 6th Annual International Cryptology Conference (CRYPTO ’87)*. 1987, pp. 186–194.

[GGP+13] Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. “Quadratic span programs and succinct NIZKs without PCPs”. In: *Proceedings of the 32nd Annual International Conference on Theory and Application of Cryptographic Techniques (EUROCRYPT ’13)*. 2013, pp. 626–645.

[GH96] Oded Goldreich and Johan Håstad. “On the complexity of interactive proofs with bounded communication”. In: *Information Processing Letters* 67.4 (1996), pp. 169–192.

[GM84] Shafi Goldwasser and Silvio Micali. “Probabilistic encryption”. In: *Journal of Computer Science and Systems* 28 (1984).

[GMR89] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. “The knowledge complexity of interactive proof systems”. In: *SIAM Journal on Computing* 18.1 (1989), pp. 186–208.

[GMW86] Oded Goldreich, Silvio Micali, and Avi Wigderson. “Proofs that yield nothing but their validity and a methodology of cryptographic protocol design”. In: *Proceedings of the 27th Annual IEEE Symposium on Foundations of Computer Science*. 1986, pp. 174–187.

[GMW91] Oded Goldreich, Silvio Micali, and Avi Wigderson. “Proofs that yield nothing but their validity or all languages in NP have zero-knowledge proof systems”. In: *Journal of the ACM* 38.3 (1991), pp. 690–728.

[Gro10] Jens Groth. “Short pairing-based non-interactive zero-knowledge arguments”. In: *Proceedings of the 16th International Conference of Theory and Application of Cryptology and Information Security (ASIACRYPT ’10)*. 2010, pp. 321–340.

[GW11] Craig Gentry and Daniel Wichs. “Separating succinct non-interactive arguments from all falsifiable assumptions”. In: *Proceedings of the 43rd Annual Symposium on Theory of Computing (STOC ’11)*. 2011, pp. 99–108.

[HPS98] Jeffrey Hoffstein, Jill Pipher, and Joseph Silverman. “NTRU: A ring-based public key cryptosystem”. In: *Algorithmic Number Theory. Lecture Notes in Computer Science* 1423 (1998).

[JMV01] Don Johnson, Alfred Menezes, and Scott Vanstone. “The elliptic curve digital signature algorithm (ECDSA)”. In: *International Journal of Information Security* 1.1 (2001), pp. 36–63.

[Kil92] Joe Kilian. "A note on efficient zero-knowledge proofs and arguments". In: *Proceedings of the 24th Annual ACM Symposium on Theory of Computing (STOC '92)*. 1992, pp. 723–732.

[Leh14] Lewis Lehrman. "The federal reserve and the dollar". In: *Cato Journal* 34.2 (2014), pp. 417–433.

[Lip12] Helger Lipmaa. "Progression-free sets and sublinear pairing-based non-interactive zero-knowledge arguments". In: *Proceedings of the 9th Theory of Cryptography Conference (TCC '12)*. 2012, pp. 169–189.

[LSP82] Leslie Lamport, Robert Shostak, and Marshall Pease. "The Byzantine Generals problem". In: *ACM Transactions on Programming Languages and Systems* 4.3 (1982), pp. 382–401.

[LWW04] Joseph Liu, Victor Wei, and Duncan Wong. "Linkable spontaneous anonymous group signature for ad hoc groups". In: *Information Security and Privacy* (2004), pp. 325–335.

[Mic00] Silvio Micali. "Computationally sound proofs". In: *SIAM Journal on Computing* 30.4 (2000), pp. 1253–1298.

[MPJ+13] Sarah Meiklejohn, Marjori Pomarole, Grant Jordan, Kirill Levchenko, Damon McCoy, Geoffrey Voelker, and Stefan Savage. "A fistful of bitcoins: characterizing payments among men with no names". In: *Proceedings of the 2013 conference on Internet measurement conference (IMC '13)*. 2013, pp. 127–140.

[MR98] Dahlia Malkhi and Michael Reiter. "Byzantine quorum systems". In: *Distributed computing* 11.4 (1998), pp. 203–213.

[Muh16] Brian Muhs. *The Ancient Egyptian Economy: 3000–30 BCE*. Cambridge, United Kingdom: Cambridge University Press, 2016.

[Nak09] Satoshi Nakamoto. *Bitcoin: a peer-to-peer electronic cash system*. 2009. URL: `http://www.bitcoin.org/bitcoin.pdf`.

[NMT16] Shen Noether, Adam Mackenzie, and Monero Core Team. *Ring confidential transactions. Monero Research Lab (MRL-0005)*. 2016. URL: `https://lab.getmonero.org/pubs/MRL-0005.pdf`.

[Par15] Bryan Parno. "A note on the unsoundness of vnTinyRAM's SNARK". In: *Cryptology ePrint Archive* 2015/437 (2015).

[Ped92] Torben Pedersen. "Non-interactive and information-theoretic secure verifiable secret sharing". In: *Advances in Cryptology (CRYPTO '91) Proceedings*. 1992.

[PGH+13] Bryan Parno, Craig Gentry, Jon Howell, and Mariana Raykova. "Pinocchio: nearly practical verifiable computation". In: *Proceedings of the 34th IEEE Symposium on Security and Privacy (Oakland '13)*. 2013, pp. 238–252.

[RH11] Fergal Reid and Martin Harrigan. “An analysis of anonymity in the Bitcoin system”. In: *2011 IEEE Third International Conference on Privacy, Security, Risk and Trust and 2011 IEEE Third International Conference on Social Computing*. 2011, pp. 1318–1326.

[RSA78] Ron Rivest, Adi Shamir, and Leonard Adleman. “A method for obtaining digital signatures and public-key cryptosystems”. In: *Communications of the ACM* 21.2 (1978), pp. 120–126.

[RST01] Ron Rivest, Adi Shamir, and Yael Tauman. “How to leak a secret”. In: *Advances in Cryptology (ASIACRYPT 2001)*. 2001, pp. 552–565.

[RW16] Serena Randolph and Austin Williams. *bitcoin-closure*. 2016. URL: `https://github.com/sharkcrayon/bitcoin-closure` (visited on 02/17/2018).

[Sch04] David Schaps. *The invention of coinage and the monetization of Ancient Greece*. Ann Arbor, Michigan, United States: University of Michigan Press, 2004.

[Sha92] Adi Shamir. “IP = PSPACE”. In: *Journal of the ACM* 39.4 (1992), pp. 869–877.

[Val08] Paul Valiant. “Incrementally verifiable computation or proof of knowledge imply time/space efficiency”. In: *Proceedings of the 5th Theory of Cryptography Conference (TCC ’08)*. 2008, pp. 1–18.