Rational Zero: Economic Security for Zerocoin with Everlasting Anonymity

Christina Garman, Matthew Green, Ian Miers(✉), and Aviel D. Rubin

Department of Computer Science, The Johns Hopkins University, Baltimore, USA
{cgarman,mgreen,imiers,rubin}@cs.jhu.edu

Abstract. Zerocoin proposed adding decentralized cryptographically anonymous e-cash to Bitcoin. Given the increasing popularity of Bitcoin and its reliance on a distributed pseudononymous public ledger, this anonymity is important if only to provide the same minimal privacy protections from nosy neighbors offered by conventional banking. Unfortunately, at 25 KB, the non-interactive zero-knowledge proofs for spending a zerocoin are nearly prohibitively large. In this paper, we consider several improvements. First, we strengthen Zerocoin's anonymity guarantees, making them independent of the size of these proofs. Given this freedom, we explore several techniques for drastically reducing proof size while ensuring that forging a single zerocoin is more difficult than the block mining process used to maintain Bitcoin's distributed ledger. Provided a zerocoin is worth less than the reward for a Bitcoin block, forging a coin is not an economically rational action. Hence we preserve Zerocoin's absolute anonymity guarantees while achieving drastic reductions in proof size by limiting ourselves to security against rational attackers.

Keywords: Privacy · e-cash

1 Introduction

Bitcoin is an electronic currency built atop a distributed transaction ledger. While Bitcoin has achieved widespread success, it has significant weaknesses related to transaction privacy [16,21]. Zerocoin [17] attempts to address these issues by extending Bitcoin with a new form of anonymous electronic cash. To add privacy while retaining Bitcoin's decentralized nature, Zerocoin uses a novel construction based on digital commitments and efficient zero-knowledge proofs that a commitment is in a list of commitments. While this construction achieves strong anonymity and prevents double spending, it can incur significant costs. In particular, to achieve cryptographically strong protection against double spending, Zerocoin uses large "spend proofs" that grow rapidly as λ, the resistance of the proofs to forgery, increases. Even for the modest $\lambda = 80$ security level (ensuring forgery effort of 2^{80} operations), Zerocoin spend proofs exceed 25 KB. Since these proofs must be stored in the block chain, the large size of these proofs makes it challenging to deploy Zerocoin in practice.

In this work we explore extensions to Zerocoin that may substantially decrease the size of these proofs. Our key observation is a need for revised assumptions. Zerocoin was designed on the assumption that all proofs must by computationally infeasible to forge. We observe that this requirement is, in a certain sense, an anachronism of cryptographic formalism. For example, in the real world we do not require that physical money be impossible to forge, merely that it be impossible to forge while making a profit. Indeed this is already true of Zerocoin: the Bitcoin block chain, upon which Zerocoin's integrity depends, does not itself provide strong cryptographic guarantees against powerful attackers. Instead, the Bitcoin protocol depends on the weaker assumption that an attacker cannot amass more than 50 % of the Bitcoin network's computational power.[1] Thus in some sense, cryptographically unforgeable zerocoins are simply impossible: even if the Zerocoin primitives resist forgery, Bitcoin's block chain can be manipulated to provide the same effect. However, the standard game-based approaches of the type used in the original security analysis of Zerocoin do not provide us any insight into safely reducing the Zerocoin security parameter. Given that this would offer a substantial performance improvements, it is interesting to consider new methods of analysis.

A primary contribution of this paper is a new methodology for examining the computational cost of forging non-interactive zero-knowledge proofs relative to the computational costs of Bitcoin mining. Our main result is as follows: by using the payout from mining a new block as a baseline, we can actually quantify the cost of forging a non-interactive zero-knowledge proof. As a result, we are able to construct game theoretic arguments for Zerocoin's resistance to forgery assuming a rational actor who wishes to profit from forging such a coin.

In and of itself, unfortunately, this new perspective does not allow us to lower the security parameter λ as far as we would like nor, consequently, realize the full reduction in proof size and increase in proof performance. To fully realize these savings, we examine two different techniques for increasing the cost of coin forgery without raising Zerocoin's proof sizes. In our new model, the security parameters are chosen based on economic considerations — such as the value of a zerocoin.

An immediate concern with our new approach is that there exist other factors that cannot be priced as easily as coin forgery. One such factor is the user's *anonymity.* There are no known techniques for pricing the value of a user's long-term transaction privacy, since this price is subjective and may vary from user to user. Moreover, we cannot easily predict the future cost of de-anonymization attacks. Indeed, since Zerocoin transcripts may be retained for long periods of time, the cost of executing an offline attack on a user's anonymity may decrease enormously over time as new computational techniques (e.g., quantum computers) become available. We must be careful in our protocol changes, since even a minor weakening of the zero-knowledge characteristics of Zerocoin's proofs could have significant long-term impact on the anonymity of users. Thus a necessary

[1] Some recent results raise questions about this 50 % number [9].

prerequisite of our above analysis is an explicit separation of Zerocoin's security as a real-world currency from its anonymity as a "pure" cryptographic protocol.

Fortunately we are able to address this concern in our work. In fact, through some simple enhancements to the Zerocoin protocol, we are able to provide an even stronger guarantee than what is provided by the original Zerocoin paper. Specifically, our new construction ensures that proofs will provide long-term statistical zero-knowledge even when the hash function they are instantiated with proves to be non-ideal, i.e., it behaves very differently from a random oracle.[2] Not just does this provide stronger anonymity guarantees, it safely allows the use of the block hash as part of the zero-knowledge proof even though the block may have adversarially controlled input. This proves to be a crucial step to increasing the cost of forging a zerocoin.

Our analysis is somewhat unusual in that it applies only to the zero-knowledge property of the proofs; we continue to analyze the soundness of the proofs under the assumption of an ideal hash. The key benefit of our approach is that we are able to retain the efficiency of the original Fiat-Shamir proofs while ensuring that user anonymity is protected over long periods of time. This gives us everlasting anonymity in the common reference string model.

Finally, as an independent contribution, we outline a construction for divisible Zerocoin. The original Zerocoin protocol proposes a new form of electronic cash in which individual coins all have the same value. While the Bitcoin-equivalent value of each zerocoin can be adjusted by protocol convention (and multiple denominations of Zerocoin can be instantiated simultaneously), this property can still be quite restrictive. In this work we show how to modify the Zerocoin protocol to create *divisible* coins, such that every zerocoin can contain an arbitrary individual denomination which may subsequently be "subdivided" into new coins of arbitrary value.

2 Background

2.1 Bitcoin

Bitcoin is a distributed e-cash system that operates without trusted parties or signing authorities. Indeed, the only cryptographic keys necessary for the system to operate are held by individual users and used to authenticate fund transfers.

At a high level, Bitcoin is a set of transaction semantics built on top of a distributed ledger which is known as the block chain. The exact semantics of the transactions are irrelevant here, so for a more detailed discussion of them and the modifications necessary for Zerocoin, we direct the reader to the original Zerocoin paper by Miers et al. [17] or the original Bitcoin paper [19].

[2] Specifically, we are concerned with future vulnerabilities in hash functions such as SHA256 that might allow for practical attacks on the zero-knowledge property of Fiat-Shamir proofs. While this concern seems rarified, existing analyses do not allow us to rule out such attacks.

Of extreme importance to our proposed modifications to Zerocoin, however, is the mechanism by which Bitcoin's ledger is maintained. We detail it here.

Consider a version of Bitcoin where there were a fixed number of network nodes. In this case, we could simply have the nodes vote on the correct version of the ledger. Under the assumption that the majority of the nodes are honest, this results in a correct ledger and hence a valid currency system. Effectively, this is the consensus technique used in Byzantine systems. However, Bitcoin is not such a closed network: anyone can download the software, fire up an instance, and join the network. In particular, one individual can fire up numerous instances and mount a Sybil attack, effectively stuffing the ballot box.

Bitcoin's approach to solving this issue is perhaps most intuitively described as the one-CPU-cyle-one-vote approach. Instead of having each node vote, consider a version of Bitcoin that places a computational requirement on voting and updating consensus. Mounting a Sybil attack would be costly. Bitcoin takes this one step further and instead of voting, actually requires a computationally intensive process to propose an update and has updates accepted only if they add on to the maximally difficult set of updates. Under the assumption that the majority of the computational power of the network is held by honest nodes and the requirements that honest nodes only build updates on valid updates, the longest chain of updates will be the correct consensus value of the ledger. Bitcoin calls this process mining, and we describe it below.

In Bitcoin, each node competes to produce an update to the block chain, known as a block, containing new transactions. The block contains a partial hash collision over (1) the previous block hash (hence block chain), (2) the hash of the transactions, and (3) a nonce. This proof of work is $\mathcal{H}_b(data||nonce) < t$ where t is the difficulty target. The target is picked by the network every two weeks in order to cause the rate at which blocks are created to average 10 min given the network's current computational power. As of November 2013, the current difficulty is $609{,}482{,}679.89 \approx 2^{29}$. The number of expected hash calculations required to generate a block is given as $\mathit{difficulty} * 2^{32}$. As a result, it takes 2^{61} expected hash calls to generate a single Bitcoin block. Bitcoin uses the double application of SHA256 as its hash function $\mathcal{H}_b$.

Bitcoin, however, goes yet one step further to ensure block chain integrity: a block is not fully trusted until it has a certain number of confirmations (typically six), meaning that there are six blocks on top of it. As a result, the effort required to manipulate a block and completely ensure it stays on the block chain is at least $2^{61} * 6 \approx 2^{63}$ hash calls.

2.2 Zero-Knowledge Proofs

In a zero-knowledge protocol [11] a user (the prover) proves a statement to another party (the verifier) without revealing anything about the statement other than that it is true.

A three-round example of a zero-knowledge protocol is often referred to as a Sigma protocol because Σ represents the flow of the protocol. The three steps can be described in the following manner: (1) commitment, (2) challenge, and

(3) response. A popular and well-known example of this is the technique of Schnorr [22], used to prove knowledge of a discrete logarithm. The protocol works as follows (Fig. 1):

Given a cyclic group G of order q with generator g and $y = g^x$, prove knowledge of x.

Prover		Verifier
Choose $r \in_R \mathbb{Z}_q$ Calculate $t = g^r$		
	$\xrightarrow{\text{Send } t}$	
		Choose $c \in_R \mathbb{Z}_q$
	$\xleftarrow{\text{Send } c}$	
Calculate $s = xc + r \pmod q$		
	$\xrightarrow{\text{Send } s}$	
		Accept if $g^s = ty^c$

Fig. 1. Schnorr protocol for proving knowledge of a discrete logarithm.

While zero-knowledge protocols are normally viewed in the "general cheating verifier" setting, where no matter the strategy of the verifier he learns no additional information, we can also consider the "honest verifier" (or semi-honest verifier) setting. An honest verifier must follow the protocol specifications exactly but maintains the ability to keep a record of the entire interaction [12]. This is of use to us because the Fiat-Shamir heuristic [10] allows us to transform any three-round (Sigma) honest-verifier zero-knowledge protocol into a non-interactive (one-round) zero-knowledge proof of knowledge with the use of a hash function modeled as a random oracle. We demonstrate an example of the application of the Fiat-Shamir heuristic using the Schnorr protocol in Fig. 2 below:

Prover		Verifier
Choose $r \in_R \mathbb{Z}_q$ Calculate $t = g^r$ Compute $c = \mathcal{H}(t)$ Calculate $s = xc + r \pmod q$		
	$\xrightarrow{\text{Send } (t,s)}$	
		Compute $c = \mathcal{H}(t)$ Accept if $g^s = ty^c$

Fig. 2. The Fiat-Shamir heuristic as applied to the Schnorr protocol.

When referring to the aforementioned proofs we will use the notation of Camenisch and Stadler [7]. For instance, $\mathsf{NIZKPoK}\{(x, y) : h = g^x \ \wedge \ c = g^y\}$ denotes a non-interactive zero-knowledge proof of knowledge of the elements x and y that satisfy both $h = g^x$ and $c = g^y$. All values not enclosed in ()'s are assumed to be known to the verifier.

2.3 Zerocoin

The original Zerocoin protocol added anonymous currency to Bitcoin that was backed by bitcoins. A zerocoin was a commitment to a serial number S. Zerocoins were minted when a user submitted a transaction spending a fixed amount of bitcoins (e.g., 1 bitcoin) and outputting a new zerocoin. The bitcoins were placed in an escrow pool and the new zerocoin added to a list of all zerocoins. Zerocoins could be spent to withdraw the same fixed bitcoins from the escrow pool by revealing the serial number of the coin and proving it came from the list of coins. This proof was examined by the distributed network running Bitcoin and, if valid and the serial number unused, the correct amount of bitcoins were transferred. Specifically, the proof was a zero-knowledge proof that (1) some coin had that serial number and (2) that that coin was on the list of minted coins. Because the proof is zero-knowledge, any given coin spend cannot be traced to its withdrawal and hence is anonymous.

The naive version of this proof, instantiated as "either this coin, or this coin, or this coin, or ...", is of size $O(n)$. The principal cryptographic contribution of the original paper was finding a compact representation of the list of coins that still admitted a commitment scheme containing a serial number. Miers et al. accomplished this by using a cryptographic accumulator [3] to represent the list of coins as one group element, a proof due to Camenisch and Lysyanskaya [6] to prove that a committed value is accumulated, and finally a double discrete log proof [8] to prove that the committed value is actually a commitment to a serial number. This results in a proof that is constant size regardless of the number of coins on the list.

Unfortunately, the double discrete log proof is constructed using cut-and-choose methods which effectively repeat a single proof multiple times to decrease the probability of forgery. As a result, the proof is of size $\lambda \cdot 2k$ where k is the size of a single field element and λ is the soundness parameter of the proof. For 1024 bit commitments and an 80 bit security level, this results in a 20 KB double discrete log proof and a total proof size (including the accumulator proof) of 25 KB. Moreover, single threaded runtime for both verification and generation of the proof runs in $O(\lambda \cdot k)$.

Finally, as the proofs for spending a zerocoin need to be publicly verifiable to allow the withdrawal of bitcoins form the escrow pool, they must be non-interactive. To accomplish this, Zerocoin uses the Fiat-Shamir heuristic to transform the above interactive proofs into non-interactive ones. Moreover, the proof is actually used as a signature of knowledge, not just spending a coin, but also signing the Bitcoin address where the withdrawn bitcoins should be deposited.

3 Everlasting Anonymity

The original zero-knowledge proofs in Zerocoin were non-interactive Fiat-Shamir proofs where both the soundness and zero-knowledge property held only in the random oracle model. This is a rather large concern since, at some point in the future, it seems likely SHA256 will be broken in a way that makes it utterly unsuitable for instantiating a random oracle, just as MD5 and MD2 have been broken. Old Zerocoin proofs using that function will still be around, and their anonymity should be preserved if possible. Intuitively, this should not be an issue, however, absent further analysis, one cannot be sure anonymity is maintained.

More significantly, one of our proposed techniques for increasing the cost of forging a zerocoin depends on the prover interacting with the block chain to generate the proof. As the block chain can be adversarially controlled, we need to ensure the proof is still zero-knowledge even in the face of block chain manipulation.

We take the expedient of detailing a simple modification to the proofs that, while still only achieving soundness in the random oracle model, achieves at least statistical zero-knowledge in the common reference string model. In the original (non-interactive) proofs, the challenge (i.e., the second move in a standard three-way "sigma" interactive zero-knowledge proof) was obtained by hashing what would have been the first move in the interactive version. In the random oracle model, a simulator can program a hash function to output arbitrary results. Accordingly, such a simulator could induce a verifier to accept a "proof" even though the simulator knew no witness to the statement being proved. Thus the original proof was zero-knowledge. Obviously when instantiated with an actual hash function, this property no longer strictly holds.

To fix this we propose applying a standard modification for converting from (interactive) honest verifier zero-knowledge proofs to (interactive) non-honest verifier proofs before applying the Fiat-Shamir heuristic: instead of making the first move in the protocol public, first commit to it and then reveal the move only after the challenge is output. Specifically, instead of hashing the first move of the transcript to create a challenge value, we hash a commitment to (the hash of) the first move of the transcript. See Fig. 3 for an example using the Schnorr protocol. As a result, any simulator who can control the common reference string can construct the commitment scheme such that they can equivocate and decommit to a first move that satisfies the generated challenge. This is not a typical approach as Fiat-Shamir proofs rely on the random oracle model themselves. However, by using this approach we get proofs that are at least statistical zero-knowledge in the common reference string model, even if soundness still requires the random oracle model, i.e., from the point of view of a privacy critical system, the proofs fail safe.

Prover		Verifier
Choose $r \in_R \mathbb{Z}_q$		
Calculate $t = g^r$		
Compute $c' = \mathcal{H}(t)$		
Choose $r' \in_R \mathbb{Z}_q$		
Calculate $com = g^{c'} h^{r'} \pmod p$		
Compute $c = \mathcal{H}(com)$		
Calculate $s = xc + r \pmod q$		
	$\xrightarrow{\text{Send } (t, com, r', s)}$	
		Compute $c' = \mathcal{H}(t)$
		Compute $c = \mathcal{H}(com)$
		Accept if $g^s = ty^c$ and
		$com = g^{c'} h^{r'} \pmod p$

Fig. 3. Dishonest verifier Schnorr protocol with Fiat-Shamir.

4 Cost Effective Security Against Forgery and Double Spending

Conceptually, payment systems are subject to three types of attacks: theft of funds, forgery of funds, and double (or more) spending of legitimate attacker controlled funds. These are major issues for both theoretical and extent currency and payment systems, and there are a broad range of solutions which vary considerably in terms of both cost and effectiveness. On one end of the spectrum, e-cash schemes typically avoid all three attacks through the use of secure cryptographic primitives which require a staggeringly prohibitive amount of computational power to break. In contrast, on the decidedly low end of the spectrum, debit cards in the US provide little-to-no security against theft/cloning. Instead they leverage fraud detection and minimization procedures to get the costs of such attacks to acceptable levels without imposing too high an overhead on transactions (e.g., verifying multiple forms of ID for every single transaction).

Certainly, the cryptographic approach is superior provided it is achievable with little overhead. Unfortunately for Zerocoin, it is neither completely achievable nor cheap: as mentioned previously, spends for even modest security parameters reach 25 KB and take 0.5 seconds to verify. Moreover, even if Zerocoin was cryptographically secure against such attacks, Bitcoin, upon which it depends, is not. Both double spends and forgery of zerocoins can be accomplished by breaking Bitcoin and without ever touching Zerocoin's underlying cryptographic primitives.

However, the approaches used by centralized credit card companies are antithetical to the decentralized nature of Bitcoin. Moreover, we prefer not to incur the administrative overhead, merchant fees, and chargebacks inherent in the fraud-management approach used by debit cards. Instead we opt for a middle ground: we create cryptographic primitives that are not cost effective to break.

4.1 The Homo-Economicus Security Model

Homo-economicus is a species of rational and narrowly self-interested actors typically found in economic papers. Since our construction provides everlasting anonymity in the common reference string model, we can safely ignore the thorny question of placing a monetary value on privacy and hence safely consider theft, forgery, and double spending attacks under the assumption that our attacker is a member of the species homo-economicus. This leads to a simple security requirement: the expected return from stealing, forging, or double (or more) spending a zerocoin should be less than the expected cost of mounting the required attack. In general, while potentially promising, this model has some large drawbacks. Estimating the real cost of a cryptographic attack is prohibitively difficult, requiring both considerable work in the concrete security model and an accurate cost function for generic computation. The theoretically elegant and simple solution to our problem is not to alter the Zerocoin construction at all. Instead, we would construct a game that, given an attacker who can forge a zerocoin, extracts the computational effort required. One would then assign a monetary value to this work and ensure it is worth more than the resulting forged coin.

We make no such attempt here. Instead, we model our construction only in the expected number of calls an attacker must make to a hash oracle and use the reward for mining a Bitcoin block to establish the market value of computation. While this approach is inherently linked to Bitcoin, it serves our limited purposes well.

Of course, such a model discounts the possibility of someone who is not financially motivated (e.g., a government) wanting to destroy the currency. While this may be a legitimate concern, we note that an attacker who merely wants to disrupt Zerocoin could also easily attack/block the underlying Bitcoin network and likely at far lower cost.

4.2 Zerocoin Attack Surface

We examine how the choice of various security parameters interacts with attacks on Zerocoin and how to minimize these parameters in light of that. Again, due to everlasting anonymity, we neglect attacks on Zerocoin's anonymity properties.

Theft. Actually stealing a user's zerocoin entails spending a coin with the same serial number. Since the Pedersen commitment containing a serial number (i.e., the coin) is information theoretically hiding, an attacker who cannot compromise a user's computer and wallet can only guess blindly. This is a very low probability event and can be made arbitrarily small by increasing the serial number length. If as an absolute minimal bound we assume 512 bit commitments, then we can have 512 bit serial numbers or, in the case of divisible coins, $512 - 64 = 448$ bit serial numbers. A theft probability of 1 in 2^{448} is too small to consider practically and hence we discount theft as a worry.

A second technical consideration for Zerocoin is that proof forgeries can deplete the escrow pool of bitcoins that zerocoins are exchanged for. This would effectively steal someone's coins. A simple solution to this is to operate with no explicit escrow pool, opting instead to destroy bitcoins when minting a zerocoin and create fresh bitcoins when spending one. As a result, forgery of a zerocoin results only in inflation. If forgery is very rare, this is a manageable problem.

Forgery. Factoring the accumulator's RSA modulus allows an attacker to forge the coin membership proof and hence forge an unlimited number of coins. This is perhaps the single biggest target in Zerocoin. As a result, we have little choice but to recommend a large modulus, say 3072 bits.

A second avenue for forging a coin is to forge the zero-knowledge proof in a spend. Each such forgery results in one and only one forged coin (since even a forged proof has a unique serial number). As such, we want to make the cost of conducting n forgeries more than the value of n coins. The bulk of the remaining portion of this section will focus on techniques to accomplish this.

Double Spending. To double spend a coin, one must assign the coin two different serial numbers. This is equivalent to causing the commitment to open to two separate values. Unfortunately, for simple Pedersen commitments, computing a single discrete log value — $\log_g(h)$ or $\log_h(g)$ — allows this to be done an infinite number of times, again giving us a single point of failure. We will discuss a modification to Pedersen commitments that makes this attack more expensive per instance, though does not eliminate entirely the aggregate effect.

4.3 Raising the Cost of Proof Forgeries

Forging a zero-knowledge proof implies guessing the challenge value prior to starting the protocol. For Fiat-Shamir based non-interaction zero-knowledge proofs, where the challenge is provided by the hash of the first move of the protocol, the only way to do this — assuming the hash function is a random oracle — is to repeatedly query the hash function until you get lucky. If the challenge value has length λ then the probability of forging the proof is $P(f) = 2^{-\lambda}$. Normally for zero-knowledge proofs we choose λ such that $P(f)$ is negligible, and hence, even with a concerted offline attack, a forgery is not feasible.

Suppose it takes b expected evaluations of $\mathcal{H}_B$ to mine a Bitcoin block. If v is the value of a coin and p is the payout from mining a block in terms of reward and collected transaction fees, then we need it to take q expected queries of $\mathcal{H}_B$ to forge the proof such that:

$$\frac{p}{b} > \frac{v}{q}$$

I.e., it pays more per hash calculation to try and mine a block than "mine" a proof forgery. Unfortunately, this analysis yields only a small reduction in the security parameter. The payout for mining a block in terms of transactions fees and the reward is roughly 2^4.[3] Mining such a block at current difficulty levels

[3] This is discounted to allow for lower payouts from, e.g., a mining cartel's cut.

takes 2^{61} calls to $\mathcal{H}_B$. Assuming a zerocoin is worth one bitcoin, solving the above equation gives us $q = 2^{57}$ and hence $\lambda = 57$.

Proof of Work. Instead of a simple query to $\mathcal{H}_B$, we can make a single instance of the zero-knowledge proof hash function make a tunable number of calls w to $\mathcal{H}_B$ in much the same manner as PBKDF2. Thus it takes $q = 2^{\lambda}w$ expected queries to $\mathcal{H}_B$ to forge a proof.

As a result, we end up with a different boundary condition for forgery unprofitability:

$$\frac{(2^{\lambda}w)p}{b} > v$$

Again assuming the current reward of 25 bitcoins per block plus transaction fees, 2^{61} invocations of $\mathcal{H}_B$ to find a block, and $\lambda = 40$ bit proofs, we end up with approximately $\frac{(2^{40}w)2^{4}}{2^{61}} > v$. If zerocoins are each worth one bitcoin, this necessitates a value of w of roughly 2^{17} or about 130 thousand hash calls. Since $\mathcal{H}_B$ is the double SHA256 computation used by Bitcoin, we can use the extensive comparisons of Bitcoin mining power across hardware to estimate the cost of this approach. A low end Intel core i3 can compute 1.8 million hashes a second, a now more than a decade old Pentium IV can compute between 0.85 and 1.29 depending on the model, and an AMR Cortex A-9 such as found in the Samsung Galaxy SII can do 1.3 million hashes a second [1]. As such, this approach is surprisingly viable even for very modest hardware.

This approach has one major limitation: it gets worse as mining difficulty increases, and mining difficulty has been increasing very rapidly as application specific integrated circuits (ASIC) mining hardware comes online. Although one could easily (and should) exclude ASICs from forging proofs via trivial changes to the hash function (e.g., changing the padding or using triple SHA256) that invalidate the ASICs but do not affect hash throughput on a general purpose computer, this does not solve the problem. We can do nothing to address the drop in payout per hash that ASICs introduce by upping the number of hashes needed to mine a block but not changing the reward.[4] Thus we would still eventually have to increase w beyond levels feasible on non specialized hardware.

Since the first move in the proof reveals nothing and our proofs allow for dishonest verifiers, this computation can be outsourced. However, paying for that outsourcing represents a catch-22: how do you anonymously pay to spend anonymous currency? While there are potential solutions to this involving small anonymous face-to-face Bitcoin transactions as a bootstrapping mechanism, they are less than ideal.

Rate-Limiting Forgeries. A second option that does not place a computational or financial burden on individuals is to rate limit the proof's hash function. To do this, we split the proof over $n + 1$ blocks. The first block encodes the first

[4] Recall that the difficulty of mining a block adjusts to keep blocks spaced at 10 min intervals. Hence greater hashing power necessitates more hashes needed per block.

moves of the protocol. The n^{th} block encodes responses to the challenge value. The λ bit challenge value is generated by taking the first $\frac{\lambda}{n}$ bits from each block of the $1, \ldots, n$ blocks and hashing them to produce a challenge. For an honest prover, this entails no additional work (unlike the proof of work system) as they can satisfy the proof for any challenge and thus must merely wait for the block chain to advance before computing the proof. A dishonest prover, on the other hand, must get a specific challenge. As such, they must either mount many parallel attempts each with a different guess at the challenge value or control the block hashes and hence the challenge. The former can be prevented by merely limiting the number of transactions in a block (Bitcoin already effectively does this by limiting the size of a block).

The likelihood that a challenge value is the one guessed is still $2^{-\lambda}$. However, assuming a maximum of 1000 Zerocoin transactions per block, attempts can only be made every half second. If we assume 40 bit security levels for the proofs, we need an expected 2^{40} hash calls and thus making a single forged zerocoin would take 2^{39} seconds or roughly seventeen thousand years. Even at Bitcoin's current unrealized theoretical maximum transaction throughput of seven transactions a second [14] this would still take over 2400 years. This seems both a prohibitive amount of time for mounting an attack and, as a practical matter, an acceptable rate of coin forgery.

Manipulating the block chain to produce the correct challenge is even more difficult. An attacker must generate far more than n blocks in order to get the correct challenge. They must first generate all n blocks, complete with proof of work for each, and extract the challenge. The overwhelmingly likely case is that the challenge is wrong, and they must repeat the process. If this was done for $n = 2$ blocks and all bits were extracted only from the last block, this would require the attacker to compute 2^{λ} expected blocks to get the right challenge and hence make $2^{\lambda+61}$ calls to $\mathcal{H}_B$ at Bitcoin's current difficulty. The situation, however, is actually worse than that since the last block only contributes $\frac{\lambda}{n}$ bits as input to the hash function the attacker is trying to get to output the guessed challenge value. Thus the attacker cannot merely generate 2^{λ} fresh n^{th} blocks knowing that by the pigeonhole principle one of those will result in the right challenge. Instead, they must actually start with a fresh first block and generate the entire sequence before checking if it works.[5] Not just does this increase the difficulty of mounting such an attack substantially, but because each block depends on the previous one, it adds in a sequential bottleneck that prevents fully parallelizing the attack process. Recall that six blocks is the threshold for normal Bitcoin transactions to be considered confirmed and as such the mere ability to compute six blocks efficiently, let alone $2^{\lambda} \cdot 6$ blocks, constitutes a massive attack on Bitcoin.

[5] It is possible to prune some of this work by checking if given, e.g., the first two of n blocks, any assignment of the remaining bits would hash to the correct challenge. We leave to future work the analysis of this strategy along with the best way to skew the sampling of bits from the n blocks to minimize it.

We stress that the above approach is not safe on its own: without the changes described in Sect. 3, adversarial manipulation of the block chain can result in a complete loss of anonymity.

4.4 Raising the Cost of Double Spends

In the original Zerocoin construction of Miers et al., computing a single discrete log of $log_g(h)$ or $log_h(g)$ broke the binding property of Pedersen commitments completely and allowed arbitrary double spends. This is undesirable since a single 1024 bit discrete log instance may be in the range of things solvable by a well-funded organization in six months to a year. We wish to avoid such an attack without using larger moduli.

Instead of using a fixed $g, h \in G$ for our commitment group, we hash the serial number into G to select g, h at random using two different hash functions, $\mathcal{H}, \mathcal{H}'$. When spending a coin, we provide these bases in the proof and then the verifier both checks the proof and that the bases result from the hash of the serial number. As a result, assuming $\mathcal{H}, \mathcal{H}'$ are collision resistant, double spends occur exactly once for any given discrete log computation.

We accomplish this by using the hash of the coin serial number S to select g and h at random. This is enforced at verification time by the verifier simply checking that $g = \mathcal{H}(S)$ and $h = \mathcal{H}'(S)$ for the provided public proof inputs. We briefly outline why this modification preserves both the blinding and binding properties of a Pedersen commitment.

Pedersen commitments are information theoretically blinding because for a fixed commitment c and any given value x, there is randomness r that opens the commitment to that value and all such r values are equally likely, i.e., for a given g^x, there exists an r such that $c = g^x h^r \mod p$. If we replace h with $h' = \mathcal{H}'(x||pad)$, then we merely shift the randomness r by $log_{h'}(h)$ and do not change the distribution on r. Hence this still holds.

Pedersen commitments are computationally binding if the discrete log problem is hard. Given a commitment c that opens to two different values x, x' with randomness r, r', one can compute the discrete log of h with respect to g by substituting in $g^l = h$ and solving $x + lr = x' + lr'$ since $g^x g^{lr} = g^{x'} g^{lr'} = g^{x+lr} = g^{x'+lr'}$. Since g and h are no longer fixed public parameters in our case, we cannot use a single violation of the blinding property to break an instance of the discrete log problem in G. It is probably possible to construct a security proof based on the assumption that the hash function is collision resistant and the discrete log problem is hard. As the rest of our constructions depend on the random oracle model for soundness, we take the expedient of programing the hash function to output the appropriate generators. This is sufficient for our purposes.

Of course, solving l discrete logs in a *fixed* $\mathbb{G}$ is not as hard as solving l discrete logs in *distinct* $\mathbb{G}_1, \ldots, \mathbb{G}_l$. The exact security of this appears not to have been well studied. Some preliminary results indicate that for Pollard's Rho algorithm, the difficulty of computing $l < \epsilon\sqrt[3]{N}$ discrete logs is approximately $\sqrt{2NL}$ where N is the order of the group and $0 < \epsilon < 1$ [2]. The far faster class of index calculus methods are still sub-exponential when run on a fixed group. Specifically, they run in $L_p(\frac{1}{2}, \frac{1}{2})$ instead of $L_p(1, \frac{1}{2})$ with a sub-exponential space requirement $L_p(\frac{1}{2}, \frac{1}{2})$ [18]. What this means in practice is an interesting question. We note that both SSH and the Internet Key Exchange protocol used in IPv6 use groups for Diffie-Hellman that are fixed for far longer timespans than we are contemplating.

5 Divisible Cash

The original Zerocoin construction did not make particularly efficient use of the fact that coins are an information theoretically blinding and computationally binding commitment that can contain arbitrary data. These commitments were merely used as a container for a serial number. Yet there are a whole number of techniques for proving far more interesting statements about commitments. These techniques allow us to construct divisible coins. We are aware of an unpublished result that makes this observation in the context of a different Zerocoin construction entirely. Our purpose in this document is not to introduce divisibility but to point out how it can be achieved using the existing cryptographic construction.

Intuition. Instead of a coin being a commitment to a serial number, we propose committing to a serial number S and a balance B. The coin owner can divide the balance B_0 in an existing coin c_0 into two new coins c_1 and c_2 with balances B_1 and B_2 respectively. She does so by creating two new coins, proving that $B_0 = B_1 + B_2$, and revealing the serial number S_0 of the divided coin c_0. Note that because we do not reveal the balance of any coin in this construction and by the original Zerocoin construction the spends for the resulting c_1 and c_2 are unlinkable to their minting, we lose nothing by explicitly identifying the original coin c_0. As such, we do not need to provide the expensive proof used for a spend, we can just identify the coin outright. This results in a highly efficient proof.

The technical question left to answer is how do we encode both the balance and the serial number in the coin? There are two possible constructions:

- We use multi-message commitments where one message is the serial number and one is the balance.
- We encode both the balance and serial number in one value in the commitment.

While conceptually elegant, multi-message commitments are problematic. In the case of Pedersen commitments [20], a commitment to a vector $\boldsymbol{m}$ of messages n is $(\prod_{i=1}^{n} g_i^{m_i})h^r$. Since the coin is then $g_1^{m_1} g_2^{m_2} h^r$, the double discrete log proof used for a coin spend must prove knowledge of three exponents instead of two. This adds approximately 10 KB to the proof. With the encoding case, we can encode the balance as the l low order bits of the original serial number and use the high $2^{l-\epsilon}$ as the actual serial number. We merely open the coin using the existing spend proof, reveal the encoded value, and then anyone can extract out the serial number and balance.

Dividing a coin c_0 is not as straightforward. We must prove that $B_0 = B_1 + B_2$ and reveal the existing coin's serial number S_0 without revealing anything about the serial numbers for the new coins. We do this as follows:

$$\pi = \mathsf{NIZKPoK}\{(S_1, S_2, r_0, r_1, r_2, B_0, B_1, B_2) : \\ (B_0 = B_1 + B_2) \wedge_{i=0}^{2} (c_i = g^{B_i + 2^{l+\epsilon} S_i} h^{r_i} \ \wedge 0 \leq B_i < 2^l \ \wedge \ 0 \leq S_i < 2^l)\}$$

This proof can be accomplished with a variety of standard techniques for efficiently proving range restrictions [4,5,13,15]. The granularity of the ranges these techniques admit vary and will define both the size l of the serial number and balance and space ϵ between the two values.

6 Conclusion

We demonstrate several useful extensions to Zerocoin. First, by removing the random oracle assumption for the zero-knowledge property of the proofs, we get everlasting security in the common reference string model. Second, and most importantly, we provide a means to model the cost of forging a coin and hence allow for cryptographic parameters to be picked to make such forgery uneconomic. As a result, we argue that one can safely reduce the soundness of the proofs from 80 bits to 40, reducing proof size from 25 KB to 10 KB and nearly halving proof generation and verification time on a single threaded implementation (or increasing throughput on a multithreaded one). The techniques used to accomplish this are specific both to Bitcoin and certain instantiations of hash functions for Fiat-Shamir proofs. We are hopeful future work will provide a general model for game-theoretic security for e-cash.

References

1. Mining hardware comparison. https://en.bitcoin.it/wiki/Mining_hardware_comparison. Accessed 23 Nov 2013
2. Kuhn, F., Struik, R.: Random walks revisited: extensions of pollard's rho algorithm for computing multiple discrete logarithms. In: Vaudenay, S., Youssef, A.M. (eds.) SAC 2001. LNCS, vol. 2259, pp. 212–229. Springer, Heidelberg (2001)
3. Benaloh, J.C., de Mare, M.: One-way accumulators: a decentralized alternative to digital signatures. In: Helleseth, T. (ed.) EUROCRYPT 1993. LNCS, vol. 765, pp. 274–285. Springer, Heidelberg (1994)
4. Boudot, F.: Efficient proofs that a committed number lies in an interval. In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, pp. 431–444. Springer, Heidelberg (2000)
5. Camenisch, J.L., Chaabouni, R., Shelat, A.: Efficient protocols for set membership and range proofs. In: Pieprzyk, J. (ed.) ASIACRYPT 2008. LNCS, vol. 5350, pp. 234–252. Springer, Heidelberg (2008)
6. Camenisch, J.L., Lysyanskaya, A.: An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In: Pfitzmann, B. (ed.) EUROCRYPT 2001. LNCS, vol. 2045, p. 93. Springer, Heidelberg (2001)

7. Camenisch, J.L., Stadler, M.A.: Efficient group signature schemes for large groups. In: Kaliski Jr., B.S. (ed.) CRYPTO 1997. LNCS, vol. 1294, pp. 410–424. Springer, Heidelberg (1997)
8. Camenisch, J.L.: Group signature schemes and payment systems based on the discrete logarithm problem. Ph.D. thesis, ETH Zürich (1998)
9. Eyal, I., Sirer, E.G.: Majority is not enough: bitcoin mining is vulnerable (2013)
10. Fiat, A., Shamir, A.: How to prove yourself: practical solutions to identification and signature problems. In: Odlyzko, A.M. (ed.) CRYPTO 1986. LNCS, vol. 263, pp. 186–194. Springer, Heidelberg (1987)
11. Goldreich, O., Micali, S., Wigderson, A.: Proofs that yield nothing but their validity and a methodology of cryptographic protocol design. In: FOCS (1986)
12. Goldreich, O.: A short tutorial of zero-knowledge (2010)
13. Groth, J.: Non-interactive zero-knowledge arguments for voting. In: Ioannidis, J., Keromytis, A.D., Yung, M. (eds.) ACNS 2005. LNCS, vol. 3531, pp. 467–482. Springer, Heidelberg (2005)
14. Lee, T.B.: Bitcoin needs to scale by a factor of 1000 to compete with Visa. Here's how to do it, November 2013. http://www.washingtonpost.com
15. Lipmaa, H.: On diophantine complexity and statistical zero-knowledge arguments. In: Laih, C.-S. (ed.) ASIACRYPT 2003. LNCS, vol. 2894, pp. 398–415. Springer, Heidelberg (2003)
16. Meiklejohn, S., Pomarole, M., Jordan, G., Levchenko, K., McCoy, D., Voelker, G.M., Savage, S.: A fistful of bitcoins: characterizing payments among men with no names. In: Internet Measurement Conference (2013)
17. Miers, I., Garman, C., Green, M., Rubin, A.D.: Zerocoin: anonymous distributed e-cash from bitcoin. In: IEEE Symposium on Security and Privacy (2013)
18. Mihalcik, J.: An analysis of algorithms for solving discrete logarithms in fixed groups. Master's thesis, Navel Post Graduate School (March 2010)
19. Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system, 2009 (2012). http://www.bitcoin.org/bitcoin.pdf
20. Pedersen, T.P.: Non-interactive and information-theoretic secure verifiable secret sharing. In: Feigenbaum, J. (ed.) CRYPTO 1991. LNCS, vol. 576, pp. 129–140. Springer, Heidelberg (1992)
21. Reid, F., Harrigan, M.: An analysis of anonymity in the Bitcoin system. In: Security and Privacy in Social Networks (SOCIALCOM) (2011)
22. Schnorr, C.P.: Efficient signature generation for smart cards. J. Cryptol. **4**(3), 239–252 (1991)