2013 IEEE Symposium on Security and Privacy

Zerocoin: Anonymous Distributed E-Cash from Bitcoin

Ian Miers, Christina Garman, Matthew Green, Aviel D. Rubin
The Johns Hopkins University Department of Computer Science, Baltimore, USA
{*imiers, cgarman, mgreen, rubin*}*@cs.jhu.edu*

***Abstract*—Bitcoin is the first e-cash system to see widespread adoption. While Bitcoin offers the potential for new types of financial interaction, it has significant limitations regarding privacy. Specifically, because the Bitcoin transaction log is completely public, users' privacy is protected only through the use of pseudonyms. In this paper we propose Zerocoin, a cryptographic extension to Bitcoin that augments the protocol to allow for fully anonymous currency transactions. Our system uses standard cryptographic assumptions and does not introduce new trusted parties or otherwise change the security model of Bitcoin. We detail Zerocoin's cryptographic construction, its integration into Bitcoin, and examine its performance both in terms of computation and impact on the Bitcoin protocol.**

I. Introduction

Digital currencies have a long academic pedigree. As of yet, however, no system from the academic literature has seen widespread use. Bitcoin, on the other hand, is a viable digital currency with a market capitalization valued at more than $100 million [1] and between $2 and $5 million USD in transactions a day [2]. Unlike many proposed digital currencies, Bitcoin is fully decentralized and requires no central bank or authority. Instead, its security depends on a distributed architecture and two assumptions: that a majority of its nodes are honest and that a substantive proof-of-work can deter Sybil attacks. As a consequence, Bitcoin requires neither legal mechanisms to detect and punish double spending nor trusted parties to be chosen, monitored, or policed. This decentralized design is likely responsible for Bitcoin's success, but it comes at a price: all transactions are public and conducted between cryptographically binding pseudonyms.

While relatively few academic works have considered the privacy implications of Bitcoin's design [2, 3], the preliminary results are not encouraging. In one example, researchers were able to trace the spending of 25,000 bitcoins that were allegedly stolen in 2011 [3, 4]. Although tracking stolen coins may seem harmless, we note that similar techniques could also be applied to trace sensitive transactions, thus violating users' privacy. Moreover, there is reason to believe that sophisticated results from other domains (e.g., efforts to de-anonymize social network data using network topology [5]) will soon be applied to the Bitcoin transaction graph.

Since all Bitcoin transactions are public, anonymous transactions are necessary to avoid tracking by third parties even if we do not wish to provide the absolute anonymity typically associated with e-cash schemes. On top of such transactions, one could build mechanisms to partially or explicitly identify participants to authorized parties (e.g., law enforcement). However, to limit this information to authorized parties, we must first anonymize the underlying public transactions.

The Bitcoin community generally acknowledges the privacy weaknesses of the currency. Unfortunately, the available mitigations are quite limited. The most common recommendation is to employ a *laundry service* which exchanges different users' bitcoins. Several of these are in commercial operation today [6, 7]. These services, however, have severe limitations: operators can steal funds, track coins, or simply go out of business, taking users' funds with them. Perhaps in recognition of these risks, many services offer short laundering periods, which lead to minimal transaction volumes and hence to limited anonymity.

Our contribution. In this paper we describe Zerocoin, a distributed e-cash system that uses cryptographic techniques to break the link between individual Bitcoin transactions *without* adding trusted parties. To do this, we first define the abstract functionality and security requirements of a new primitive that we call a *decentralized e-cash* scheme. We next propose a concrete instantiation and prove it secure under standard cryptographic assumptions. Finally, we describe the specific extensions required to integrate our protocol into the Bitcoin system and evaluate the performance of a prototype implementation derived from the original open-source `bitcoind` client.

We are not the first to propose e-cash techniques for solving Bitcoin's privacy problems. However, a common problem with many e-cash protocols is that they rely fundamentally on a trusted currency issuer or "bank," who creates electronic "coins" using a blind signature scheme. One solution (attempted unsuccessfully with Bitcoin [8]) is to simply appoint such a party. Alternatively, one can distribute the responsibility among a quorum of nodes using threshold cryptography. Unfortunately, both of these solutions introduce points of failure and seem inconsistent with the Bitcoin network model, which consists of many untrusted nodes that routinely enter and exit the network. Moreover, the problem of choosing long-term trusted parties, especially in the legal and regulatory grey area Bitcoin operates in, seems like a major impediment to adoption. Zerocoin eliminates

1081-6011/13 $26.00 © 2013 IEEE
DOI 10.1109/SP.2013.34

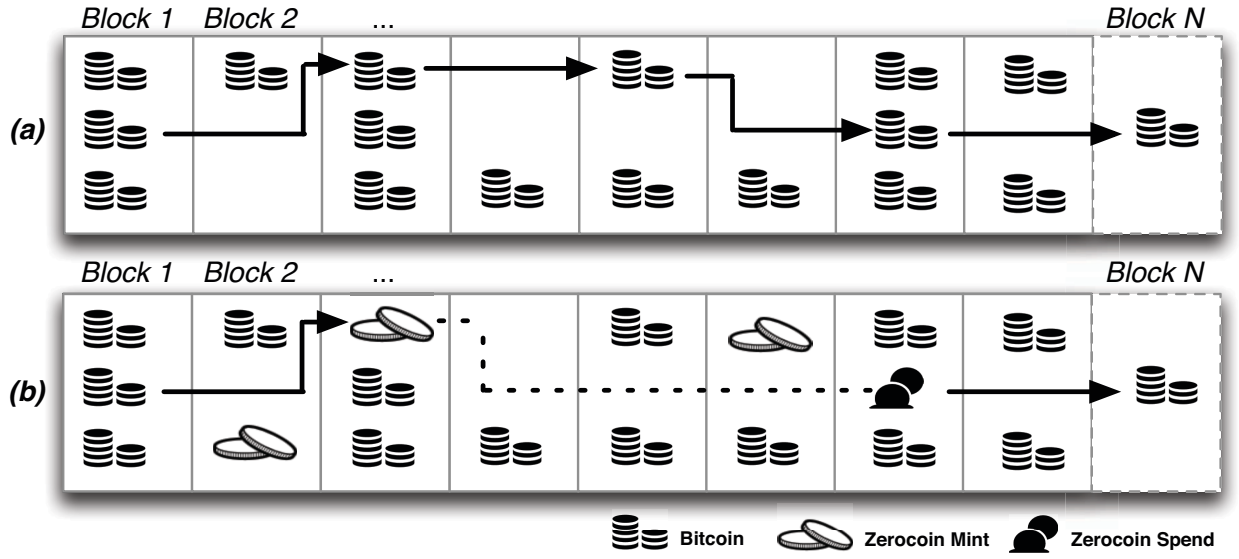


Figure 1: Two example block chains. Chain (a) illustrates a normal Bitcoin transaction history, with each transaction linked to a preceding transaction. Chain (b) illustrates a Zerocoin chain. The linkage between mint and spend (dotted line) cannot be determined from the block chain data.

the need for such coin issuers by allowing individual Bitcoin clients to generate their own coins — provided that they have sufficient classical bitcoins to do so.

Intuition behind our construction. To understand the intuition behind Zerocoin, consider the following "pencil and paper" protocol example. Imagine that all users share access to a physical bulletin board. To mint a zerocoin of fixed denomination \$1, a user Alice first generates a random coin serial number S, then commits to S using a secure digital commitment scheme. The resulting commitment is a coin, denoted C, which can only be opened by a random number r to reveal the serial number S. Alice pins C to the public bulletin board, along with \$1 of physical currency. All users will accept C provided it is correctly structured and carries the correct sum of currency.

To redeem her coin C, Alice first scans the bulletin board to obtain the set of valid commitments $(C_1, \ldots, C_N)$ that have thus far been posted by *all* users in the system. She next produces a non-interactive zero-knowledge proof π for the following two statements: (1) she knows a $C \in (C_1, \ldots, C_N)$ and (2) she knows a hidden value r such that the commitment C opens to S. In full view of the others, Alice, using a disguise to hide her identity,[1] posts a "spend" transaction containing (S, π). The remaining users verify the proof π and check that S has not previously appeared in any other spend transaction. If these conditions are met, the users allow Alice to collect \$1 from *any* location on the bulletin board; otherwise they reject her transaction and prevent her from collecting the currency.

This simple protocol achieves some important aims. First, Alice's minted coin cannot be linked to her retrieved funds: in order to link the coin C to the the serial number S used in her withdrawal, one must either know r or directly know which coin Alice proved knowledge of, neither of which are revealed by the proof. Thus, even if the original dollar bill is recognizably tainted (e.g., it was used in a controversial transaction), it cannot be linked to Alice's new dollar bill. At the same time, if the commitment and zero-knowledge proof are secure, then Alice cannot double-spend any coin without re-using the serial number S and thus being detected by the network participants.

Of course, the above protocol is not workable: bulletin boards are a poor place to store money and critical information. Currency might be stolen or serial numbers removed to allow double spends. More importantly, to conduct this protocol over a network, Alice requires a distributed digital backing currency.[2]

The first and most basic contribution of our work is to recognize that Bitcoin answers all of these concerns, providing us with a backing currency, a bulletin board, and a conditional currency redemption mechanism. Indeed, the core of the Bitcoin protocol is the decentralized calculation

[1]Of course, in the real protocol Alice will emulate this by using an anonymity network such as Tor [9].

[2]One could easily imagine a solution based on existing payment networks, e.g., Visa or Paypal. However, this would introduce the need for trusted parties or exchanges.

of a *block chain* which acts as a trusted, append-only bulletin board that can both store information and process financial transactions. Alice can add her commitments *and* escrow funds by placing them in the block chain while being assured that strict protocol conditions (and not her colleagues' scruples) determine when her committed funds may be accessed.

Of course, even when integrated with the Bitcoin block chain, the protocol above has another practical challenge. Specifically, it is difficult to *efficiently* prove that a commitment C is in the set $(C_1, \ldots, C_N)$. The naive solution is to prove the disjunction $(C = C_1) \vee (C = C_2) \vee \ldots \vee (C = C_N)$. Unfortunately such "OR proofs" have size $O(N)$, which renders them impractical for all but small values of N.

Our second contribution is to solve this problem, producing a new construction with proofs that do not grow linearly as N increases. Rather than specifying an expensive OR proof, we employ a "public" one-way accumulator to reduce the size of this proof. One-way accumulators [10, 11, 12, 13, 14], first proposed by Benaloh and de Mare [10], allow parties to combine many elements into a constant-sized data structure, while efficiently *proving* that one specific value is contained within the set. In our construction, the Bitcoin network computes an accumulator A over the commitments $(C_1, \ldots, C_N)$, along with the appropriate membership witnesses for each item in the set. The spender need only prove knowledge of one such witness. In practice, this can reduce the cost of the spender's proof to $O(log\ N)$ or even constant size.

Our application requires specific properties from the accumulator. With no trusted parties, the accumulator and its associated witnesses must be *publicly* computable and verifiable (though we are willing to relax this requirement to include a single, trusted *setup* phase in which parameters are generated). Moreover, the accumulator must bind even the computing party to the values in the set. Lastly, the accumulator must support an efficient non-interactive witness-indistinguishable or zero-knowledge proof of set membership. Fortunately such accumulators do exist. In our concrete proposal of Section IV we use a construction based on the Strong RSA accumulator of Camenisch and Lysyanskaya [12], which is in turn based on an accumulator of Baric and Pfitzmann [11] and Benaloh and de Mare [10].

Outline of this work. The rest of this paper proceeds as follows. In Section II we provide a brief technical overview of the Bitcoin protocol. In Section III we formally define the notion of *decentralized e-cash* and provide correctness and security requirements for such a system. In Section IV we give a concrete realization of our scheme based on standard cryptographic hardness assumptions including the Discrete Logarithm problem and Strong RSA. Finally, in Sections V, VI, and VII, we describe how we integrate our e-cash construction into the Bitcoin protocol, discuss the security and anonymity provided, and detail experimental results showing that our solution is practical.

II. Overview of Bitcoin

In this section we provide a short overview of the Bitcoin protocol. For a more detailed explanation, we refer the reader to the original specification of Nakamoto [15] or to the summary of Barber *et al.* [2].

The Bitcoin network. Bitcoin is a peer-to-peer network of nodes that distribute and record transactions, and clients used to interact with the network. The heart of Bitcoin is the *block chain*, which serves as an append-only bulletin board maintained in a distributed fashion by the Bitcoin peers. The block chain consists of a series of blocks connected in a hash chain.[3] Every Bitcoin block memorializes a set of transactions that are collected from the Bitcoin broadcast network.

Bitcoin peers compete to determine which node will generate the next canonical block. This competition requires each node to solve a proof of work based on identifying specific SHA-256 preimages, specifically a block B such that $\mathsf{SHA256}(\mathsf{SHA256}(B)) = (0^\ell || \{0,1\}^{256-\ell})$.[4] The value ℓ is selected by a periodic network vote to ensure that on average a block is created every 10 minutes. When a peer generates a valid solution, a process known as *mining*, it broadcasts the new block to all nodes in the system. If the block is valid (i.e., all transactions validate and a valid proof of work links the block to the chain thus far), then the new block is accepted as the head of the block chain. The process then repeats.

Bitcoin provides two separate incentives to peers that mine new blocks. First, successfully mining a new block (which requires a non-trivial computational investment) entitles the creator to a reward, currently set at 25 BTC.[5] Second, nodes who mine blocks are entitled to collect *transaction fees* from every transaction they include. The fee paid by a given transaction is determined by its author (though miners may exclude transactions with insufficient fees or prioritize high fee transactions).

Bitcoin transactions. A Bitcoin transaction consists of a set of outputs and inputs. Each output is described by the tuple (a, V) where a is the amount, denominated in Satoshi (one bitcoin = 10^9 Satoshi), and V is a specification of who is authorized to spend that output. This specification, denoted $\mathsf{scriptPubKey}$, is given in *Bitcoin script*, a stack-based non-Turing-complete language similar to Forth. Transaction inputs

[3] For efficiency reasons, this chain is actually constructed using a hash tree, but we use the simpler description for this overview.

[4] Each block includes a counter value that may be incremented until the hash satisfies these requirements.

[5] The Bitcoin specification holds that this reward should be reduced every few years, eventually being eliminated altogether.

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

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

Figure 2: Example Bitcoin transaction. The output script specifies that the redeeming party provide a public key that hashes to the given value and that the transaction be signed with the corresponding private key.

are simply a reference to a previous transaction output,[6] as well as a second script, scriptSig, with code and data that when combined with scriptPubKey evaluates to true. Coinbase transactions, which start off every block and pay its creator, do not include a transaction input.

To send d bitcoins to Bob, Alice embeds the hash[7] of Bob's ECDSA public key pk_b, the amount d, and some script instructions in scriptPubKey as one output of a transaction whose referenced inputs total at least d bitcoins (see Figure 2). Since any excess input is paid as a transaction fee to the node who includes it in a block, Alice typically adds a second output paying the surplus change back to herself. Once the transaction is broadcasted to the network and included in a block, the bitcoins belong to Bob. However, Bob should only consider the coins his once at least five subsequent blocks reference this block.[8] Bob can spend these coins in a transaction by referencing it as an input and including in scriptSig a signature on the claiming transaction under sk_b and the public key pk_b.

Anonymity. Anonymity was not one of the design goals of Bitcoin [3, 15, 17]. Bitcoin provides only *pseudonymity* through the use of Bitcoin identities (public keys or their hashes), of which a Bitcoin user can generate an unlimited number. Indeed, many Bitcoin clients routinely generate new identities in an effort to preserve the user's privacy.

Regardless of Bitcoin design goals, Bitcoin's user base seems willing to go through considerable effort to maintain their anonymity — including risking their money and paying transaction fees. One illustration of this is the existence of laundries that (for a fee) will mix together different users' funds in the hopes that shuffling makes them difficult to trace [2, 6, 7]. Because such systems require the users to trust the laundry to both (a) not record how the mixing is done and (b) give the users back the money they put in to the pot, use of these systems involves a fair amount of risk.

[6]This reference consists of a transaction hash identifier as well as an index into the transaction's output list.

[7]A 34 character hash that contains the double SHA-256 hash of the key and some checksum data.

[8]Individual recipients are free to disregard this advice. However, this could make them vulnerable to double-spending attacks as described by Karame *et al.* [16].

III. Decentralized E-Cash

Our approach to anonymizing the Bitcoin network uses a form of cryptographic e-cash. Since our construction does not require a central coin issuer, we refer to it as a *decentralized e-cash scheme*. In this section we define the algorithms that make up a decentralized e-cash scheme and describe the correctness and security properties required of such a system.

Notation. Let λ represent an adjustable security parameter, let $poly(\cdot)$ represent some polynomial function, and let $\nu(\cdot)$ represent a negligible function. We use $\mathcal{C}$ to indicate the set of allowable coin values.

Definition 3.1 (Decentralized E-Cash Scheme): A decentralized e-cash scheme consists of a tuple of possibly randomized algorithms $(\mathsf{Setup}, \mathsf{Mint}, \mathsf{Spend}, \mathsf{Verify})$.

- $\mathsf{Setup}(1^\lambda) \to params$. On input a security parameter, output a set of global public parameters $params$ and a description of the set $\mathcal{C}$.
- $\mathsf{Mint}(params) \to (c, skc)$. On input parameters $params$, output a coin $c \in \mathcal{C}$, as well as a trapdoor skc.
- $\mathsf{Spend}(params, c, skc, R, \mathbf{C}) \to (\pi, S)$. Given $params$, a coin c, its trapdoor skc, some transaction string $R \in \{0,1\}^*$, and an arbitrary set of coins $\mathbf{C}$, output a coin spend transaction consisting of a proof π and serial number S if $c \in \mathbf{C} \subseteq \mathcal{C}$. Otherwise output $\perp$.
- $\mathsf{Verify}(params, \pi, S, R, \mathbf{C}) \to \{0,1\}$. Given $params$, a proof π, a serial number S, transaction information R, and a set of coins $\mathbf{C}$, output 1 if $\mathbf{C} \subseteq \mathcal{C}$ and (π, S, R) is valid. Otherwise output 0.

We note that the Setup routine may be executed by a trusted party. Since this setup occurs only once and does not produce any corresponding secret values, we believe that this relaxation is acceptable for real-world applications. Some concrete instantiations may use different assumptions.

Each coin is generated using a randomized minting algorithm. The *serial number* S is a unique value released during the spending of a coin and is designed to prevent any user from spending the same coin twice. We will now formalize the correctness and security properties of a decentralized e-cash scheme. Each call to the Spend algorithm can include an arbitrary string R, which is intended to store transaction-specific information (e.g., the identity of a transaction recipient).

Correctness. Every decentralized e-cash scheme must satisfy the following correctness requirement. Let $params \leftarrow \mathsf{Setup}(1^\lambda)$ and $(c, skc) \leftarrow \mathsf{Mint}(params)$. Let $\mathbf{C} \subseteq \mathcal{C}$ be any valid set of coins, where $|\mathbf{C}| \leq poly(\lambda)$, and

assign $(\pi, S) \leftarrow \mathsf{Spend}(params, c, skc, R, \mathbf{C})$. The scheme is *correct* if, over all $\mathbf{C}$, R, and random coins used in the above algorithms, the following equality holds with probability $1 - \nu(\lambda)$:

$$\mathsf{Verify}(params, \pi, S, R, \mathbf{C} \cup \{c\}) = 1$$

Security. The security of a decentralized e-cash system is defined by the following two games: Anonymity and Balance. We first describe the Anonymity experiment, which ensures that the adversary cannot link a given coin spend transaction (π, S) to the coin associated with it, even when the attacker provides many of the coins used in generating the spend transaction.

Definition 3.2 (Anonymity): A decentralized e-cash scheme $\Pi = (\mathsf{Setup}, \mathsf{Mint}, \mathsf{Spend}, \mathsf{Verify})$ satisfies the *Anonymity* requirement if every *probabilistic polynomial-time (p.p.t.)* adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ has negligible advantage in the following experiment.

$\mathsf{Anonymity}(\Pi, \mathcal{A}, \lambda)$
$params \leftarrow \mathsf{Setup}(1^\lambda)$
For $i \in \{0,1\}$: $(c_i, skc_i) \leftarrow \mathsf{Mint}(params)$
$(\mathbf{C}, R, z) \leftarrow \mathcal{A}_1(params, c_0, c_1)$; $b \leftarrow \{0,1\}$
$(\pi, S) \leftarrow \mathsf{Spend}(params, c_b, skc_b, R, \mathbf{C} \cup \{c_0, c_1\})$
Output: $b' \leftarrow \mathcal{A}_2(z, \pi, S)$

We define $\mathcal{A}$'s advantage in the above game as $|\mathbf{Pr}[\, b = b' \,] - 1/2|$.

The Balance property requires more consideration. Intuitively, we wish to ensure that an attacker cannot *spend* more coins than she mints, even when she has access to coins and spend transactions produced by honest parties. Note that to strengthen our definition, we also capture the property that an attacker might alter valid coins, e.g., by modifying their transaction information string R.

Our definition is reminiscent of the "one-more forgery" definition commonly used for blind signatures. We provide the attacker with a collection of valid coins and an oracle $\mathcal{O}_{spend}$ that she may use to spend any of them.[9] Ultimately $\mathcal{A}$ must produce m coins and $m+1$ valid spend transactions such that no transaction duplicates a serial number or modifies a transaction produced by the honest oracle.

Definition 3.3 (Balance): A decentralized e-cash scheme $\Pi = (\mathsf{Setup}, \mathsf{Mint}, \mathsf{Spend}, \mathsf{Verify})$ satisfies the *Balance* property if $\forall N \leq \mathrm{poly}(\lambda)$ every *p.p.t.* adversary $\mathcal{A}$ has negligible advantage in the following experiment.

$\mathsf{Balance}(\Pi, \mathcal{A}, N, \lambda)$
$params \leftarrow \mathsf{Setup}(1^\lambda)$
For $i = 1$ to N: $(c_i, skc_i) \leftarrow \mathsf{Mint}(params)$
Output: $(c'_1, \ldots, c'_m, \mathcal{S}_1, \ldots, \mathcal{S}_m, \mathcal{S}_{m+1})$
$\leftarrow \mathcal{A}^{\mathcal{O}_{spend}(\cdot,\cdot,\cdot)}(params, c_1, \ldots, c_N)$

[9]We provide this functionality as an oracle to capture the possibility that the attacker can specify *arbitrary* input for the value $\mathbf{C}$.

The oracle $\mathcal{O}_{spend}$ operates as follows: on the j_{th} query $\mathcal{O}_{spend}(c_j, \mathbf{C}_j, R_j)$, the oracle outputs $\perp$ if $c_j \notin \{c_1, \ldots, c_N\}$. Otherwise it returns $(\pi_j, S_j) \leftarrow \mathsf{Spend}(params, c_j, skc_j, R_j, \mathbf{C}_j)$ to $\mathcal{A}$ and records (S_j, R_j) in the set $\mathcal{T}$.

We say that $\mathcal{A}$ wins (i.e., she produces more spends than minted coins) if $\forall \mathfrak{s} \in \{\mathcal{S}_1, \ldots, \mathcal{S}_m, \mathcal{S}_{m+1}\}$ where $\mathfrak{s} = (\pi', S', R', \mathbf{C}')$:

- $\mathsf{Verify}(params, \pi', S', R', \mathbf{C}') = 1$.
- $\mathbf{C}' \subseteq \{c_1, \ldots, c_N, c'_1, \ldots, c'_m\}$.
- $(S', R') \notin \mathcal{T}$.
- S' appears in only one tuple from $\{\mathcal{S}_1, \ldots, \mathcal{S}_m, \mathcal{S}_{m+1}\}$.

We define $\mathcal{A}$'s advantage as the probability that $\mathcal{A}$ wins the above game.

IV. Decentralized E-Cash from Strong RSA

In this section we describe a *concrete* instantiation of a decentralized e-cash scheme. We first define the necessary cryptographic ingredients.

A. Cryptographic Building Blocks

Zero-knowledge proofs and signatures of knowledge. Our protocols use zero-knowledge proofs that can be instantiated using the technique of Schnorr [18], with extensions due to e.g., [19, 20, 21, 22]. We convert these into *non-interactive* proofs by applying the Fiat-Shamir heuristic [23]. In the latter case, we refer to the resulting non-interactive proofs as *signatures of knowledge* as defined in [24].

When referring to these proofs we will use the notation of Camenisch and Stadler [25]. 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. Similarly, the extension $\mathsf{ZKSoK}[m]\{(x, y) : h = g^x \wedge c = g^y\}$ indicates a *signature of knowledge* on message m.

Accumulators. Our construction uses an accumulator based on the Strong RSA assumption. The accumulator we use was first proposed by Benaloh and de Mare [10] and later improved by Baric and Pfitzmann [11] and Camenisch and Lysyanskaya [12]. We describe the accumulator using the following algorithms:

- $\mathsf{AccumSetup}(\lambda) \rightarrow params$. On input a security parameter, sample primes p, q (with polynomial dependence on the security parameter), compute $N = pq$, and sample a seed value $u \in QR_N, u \neq 1$. Output (N, u) as $params$.
- $\mathsf{Accumulate}(params, \mathbf{C}) \rightarrow A$. On input $params$ (N, u) and a set of prime numbers $\mathbf{C} = \{c_1, \ldots, c_i \mid c \in [\mathcal{A}, \mathcal{B}]\}$,[10] compute the accumulator A as $u^{c_1 c_2 \cdots c_n} \bmod N$.

[10]See Appendix A for a more precise description.

- $\mathsf{GenWitness}(params, v, \mathbf{C}) \rightarrow w$. On input $params$ (N, u), a set of prime numbers $\mathbf{C}$ as described above, and a value $v \in \mathbf{C}$, the witness w is the accumulation of all the values in $\mathbf{C}$ besides v, i.e., $w = \mathsf{Accumulate}(params, \mathbf{C} \setminus \{v\})$.
- $\mathsf{AccVerify}(params, A, v, \omega) \rightarrow \{0, 1\}$. On input $params$ (N, u), an element v, and witness ω, compute $A' \equiv \omega^v \mod N$ and output 1 if and only if $A' = A$, v is prime, and $v \in [\mathcal{A}, \mathcal{B}]$ as defined previously.

For simplicity, the description above uses the full calculation of A. Camenisch and Lysyanskaya [12] observe that the accumulator may also be *incrementally* updated, i.e., given an existing accumulator A_n it is possible to add an element x and produce a new accumulator value A_{n+1} by computing $A_{n+1} = A_n^x \mod N$. We make extensive use of this optimization in our practical implementation.

Camenisch and Lysyanskaya [12] show that the accumulator satisfies a strong *collision-resistance* property if the Strong RSA assumption is hard. Informally, this ensures that no *p.p.t.* adversary can produce a pair (v, ω) such that $v \notin \mathbf{C}$ and yet AccVerify is satisfied. Additionally, they describe an efficient zero-knowledge proof of knowledge that a committed value is in an accumulator. We convert this into a non-interactive proof using the Fiat-Shamir transform and refer to the resulting proof using the following notation:

$$\mathsf{NIZKPoK}\{(v, \omega) : \mathsf{AccVerify}((N, u), A, v, \omega) = 1\}.$$

B. Our Construction

We now describe a concrete decentralized e-cash scheme. Our scheme is secure assuming the hardness of the Strong RSA and Discrete Logarithm assumptions, and the existence of a zero-knowledge proof system.

We now describe the algorithms:

- $\mathsf{Setup}(1^\lambda) \rightarrow params$. On input a security parameter, run $\mathsf{AccumSetup}(1^\lambda)$ to obtain the values (N, u). Next generate primes p, q such that $p = 2^w q + 1$ for $w \geq 1$. Select random generators g, h such that $\mathbb{G} = \langle g \rangle = \langle h \rangle$ and $\mathbb{G}$ is a subgroup of $\mathbb{Z}_q^*$. Output $params = (N, u, p, q, g, h)$.
- $\mathsf{Mint}(params) \rightarrow (c, skc)$. Select $S, r \leftarrow \mathbb{Z}_q^*$ and compute $c \leftarrow g^S h^r \bmod p$ such that $\{c \text{ prime} \mid c \in [\mathcal{A}, \mathcal{B}]\}$.[11] Set $skc = (S, r)$ and output (c, skc).
- $\mathsf{Spend}(params, c, skc, R, \mathbf{C}) \rightarrow (\pi, S)$. If $c \notin \mathbf{C}$ output $\perp$. Compute $A \leftarrow \mathsf{Accumulate}((N, u), \mathbf{C})$ and $\omega \leftarrow \mathsf{GenWitness}((N, u), c, \mathbf{C})$. Output (π, S) where π comprises the following signature of knowledge:[12]

$$\pi = \mathsf{ZKSoK}[R]\{(c, w, r) : \mathsf{AccVerify}((N, u), A, c, w) = 1 \ \wedge \ c = g^S h^r\}$$

- $\mathsf{Verify}(params, \pi, S, R, \mathbf{C}) \rightarrow \{0, 1\}$. Given a proof π, a serial number S, and a set of coins $\mathbf{C}$, first compute $A \leftarrow \mathsf{Accumulate}((N, u), \mathbf{C})$. Next verify that π is the aforementioned signature of knowledge on R using the known public values. If the proof verifies successfully, output 1, otherwise output 0.

Our protocol assumes a trusted setup process for generating the parameters. We stress that the accumulator trapdoor (p, q) is not used subsequent to the Setup procedure and can therefore be destroyed immediately after the parameters are generated. Alternatively, implementers can use the technique of Sander for generating so-called RSA UFOs for accumulator parameters without a trapdoor [26].

C. Security Analysis

We now consider the security of our construction.

Theorem 4.1: If the zero-knowledge signature of knowledge is computationally zero-knowledge in the random oracle model, then $\Pi = (\mathsf{Setup}, \mathsf{Mint}, \mathsf{Spend}, \mathsf{Verify})$ satisfies the Anonymity property.

We provide a proof sketch for Theorem 4.1 in Appendix A. Intuitively, the security of our construction stems from the fact that the coin commitment C is a perfectly-hiding commitment and the signature proof π is at least computationally zero-knowledge. These two facts ensure that the adversary has at most negligible advantage in guessing which coin was spent.

Theorem 4.2: If the signature proof π is *sound* in the random oracle model, the Strong RSA problem is hard, and the Discrete Logarithm problem is hard in $\mathbb{G}$, then $\Pi = (\mathsf{Setup}, \mathsf{Mint}, \mathsf{Spend}, \mathsf{Verify})$ satisfies the Balance property.

A proof of Theorem 4.1 is included in Appendix A. Briefly, this proof relies on the binding properties of the coin commitment, as well as the soundness and unforgeability of the ZKSoK and collision-resistance of the accumulator. We show that an adversary who wins the Balance game with non-negligible advantage can be used to *either* find a collision in the commitment scheme (allowing us to solve the Discrete Logarithm problem) *or* find a collision in the accumulator (which leads to a solution for Strong RSA).

V. Integrating with Bitcoin

While the construction of the previous section gives an overview of our approach, we have yet to describe how our techniques integrate with Bitcoin. In this section we address the specific challenges that come up when we combine a decentralized e-cash scheme with the Bitcoin protocol.

The general overview of our approach is straightforward. To mint a zerocoin c of denomination d, Alice runs $\mathsf{Mint}(params) \rightarrow (c, skc)$ and stores skc securely.[13] She then embeds c in the output of a Bitcoin transaction that spends $d + fees$ classical bitcoins. Once a mint transaction has been accepted into the block chain, c is included in the

[11] See Appendix A for a more precise description.

[12] See Appendix B for the construction of the ZKSoK.

[13] In our implementation all bitcoins have a single fixed value. However, we can support multiple values by running distinct Zerocoin instantiations simultaneously, all sharing the same set of public parameters.

global accumulator A, and the currency cannot be accessed except through a Zerocoin spend, i.e., it is essentially placed into escrow.

To spend c with Bob, Alice first constructs a partial transaction ptx that references an unclaimed mint transaction as input and includes Bob's public key as output. She then traverses all valid mint transactions in the block chain, assembles the set of minted coins $\mathbf{C}$, and runs $\mathsf{Spend}(params, c, skc, hash(ptx), \mathbf{C}) \rightarrow (\pi, S)$. Finally, she completes the transaction by embedding (π, S) in the scriptSig of the input of ptx. The output of this transaction could also be a further Zerocoin mint transaction — a feature that may be useful to transfer value between multiple Zerocoin instances (i.e., of different denomination) running in the same block chain.

When this transaction appears on the network, nodes check that $\mathsf{Verify}(params, \pi, S, hash(ptx), \mathbf{C}) = 1$ and check that S does not appear in any previous transaction. If these condition hold and the referenced mint transaction is not claimed as an input into a different transaction, the network accepts the spend as valid and allows Alice to redeem d bitcoins.

Computing the accumulator. A naive implementation of the construction in Section IV requires that the verifier re-compute the accumulator A with each call to $\mathsf{Verify}(\ldots)$. In practice, the cost can be substantially reduced.

First, recall that the accumulator in our construction can be computed *incrementally*, hence nodes can add new coins to the accumulation when they arrive. To exploit this, we require any node mining a new block to add the zerocoins in that block to the previous block's accumulator and store the resulting new accumulator value in the *coinbase transaction* at the start of the new block.[14] We call this an *accumulator checkpoint*. Peer nodes validate this computation before accepting the new block into the blockchain. Provided that this verification occurs routinely when blocks are added to the chain, some clients may choose to trust the accumulator in older (confirmed) blocks rather than re-compute it from scratch.

With this optimization, Alice need no longer compute the accumulator A and the full witness w for c. Instead she can merely reference the current block's *accumulator checkpoint* and compute the witness starting from the checkpoint preceding her mint (instead of starting at T_0), since computing the witness is equivalent to accumulating $\mathbf{C} \setminus \{c\}$.

New transaction types. Bitcoin transactions use a flexible scripting language to determine the validity of each transaction. Unfortunately, Bitcoin script is (by design) *not* Turing-complete. Moreover, large segments of the already-limited script functionality have been disabled in the Bitcoin production network due to security concerns. Hence, the existing script language cannot be used for sophisticated calculations such as verifying zero-knowledge proofs. Fortunately for our purposes, the Bitcoin designers chose to reserve several script operations for future expansion.

We extend Bitcoin by adding a new instruction: ZEROCOIN_MINT. Minting a zerocoin constructs a transaction with an output whose scriptPubKey contains this instruction and a coin c. Nodes who receive this transaction should validate that c is a well-formed coin. To spend a zerocoin, Alice constructs a new transaction that claims as input some Zerocoin mint transaction and has a scriptSig field containing (π, S) and a reference to the block containing the accumulator used in π. A verifier extracts the accumulator from the referenced block and, using it, validates the spend as described earlier.

Finally, we note that transactions must be signed to prevent an attacker from simply changing who the transaction is payed to. Normal Bitcoin transactions include an ECDSA signature by the key specified in the scriptPubKey of the referenced input. However, for a spend transaction on an *arbitrary* zerocoin, there is no ECDSA public key. Instead, we use the ZKSoK π to sign the transaction hash that normally would be signed using ECDSA.[15]

Statekeeping and side effects. Validating a zerocoin changes Bitcoin's semantics: currently, Bitcoin's persistent state is defined solely in terms of transactions and blocks of transactions. Furthermore, access to this state is done via explicit reference by hash. Zerocoin, on the other hand, because of its strong anonymity requirement, deals with existentials: the coin is in the set of thus-far-minted coins and its serial number is *not* yet in the set of spent serial numbers. To enable these type of qualifiers, we introduce side effects into Bitcoin transaction handling. Processing a mint transaction causes a coin to be accumulated as a side effect. Processing a spend transaction causes the coin serial number to be added to a list of spent serial numbers held by the client.

For coin serial numbers, we have little choice but to keep a full list of them per client and incur the (small) overhead of storing that list and the larger engineering overhead of handling all possible ways a transaction can enter a client. The accumulator state is maintained within the accumulator checkpoints, which the client verifies for each received block.

Proof optimizations. For reasonable parameter sizes, the proofs produced by $\mathsf{Spend}(\ldots)$ exceed Bitcoin's 10KB transaction size limits. Although we can simply increase this limit, doing so has two drawbacks: (1) it drastically increases the storage requirements for Bitcoin since current transactions

[14]The coinbase transaction format already allows for the inclusion of arbitrary data, so this requires no fundamental changes to the Bitcoin protocol.

[15]In practice, this modification simply requires us to include the transaction digest in the hash computation of the challenge for the Fiat-Shamir proofs. See Appendix A for details.

are between 1 and 2 KB and (2) it may increase memory pressure on clients that store transactions in memory.[16]

In our prototype implementation we store our proofs in a separate, well-known location (a simple server). A full implementation could use a Distributed Hash Table or non block-chain backed storage in Bitcoin. While we recommend storing proofs in the block chain, these alternatives do not increase the storage required for the block chain.[17]

A. Suggestions for Optimizing Proof Verification

The complexity of the proofs will also lead to longer verification times than expected with a standard Bitcoin transaction. This is magnified by the fact that a Bitcoin transaction is verified once when it is included by a block and again by every node when that block is accepted into the block chain. Although the former cost can be accounted for by charging transaction fees, it would obviously be ideal for these costs to be as low as possible.

One approach is to distribute the cost of verification over the entire network and not make each node verify the entire proof. Because the ZKSoK we use utilizes cut-and-choose techniques, it essentially consists of n repeated iterations of the same proof (reducing the probability of forgery to roughly 2^{-n}). We can simply have nodes *randomly* select which iterations of the proofs they verify. By distributing this process across the network, we should achieve approximately the same security with less duplication of effort.

This optimization involves a time-space tradeoff, since the existing proof is verified by computing a series of (at a minimum) 1024 bit values $T_1, \ldots, T_n$ and hashing the result. A naive implementation would require us to send $T_1, \ldots, T_n$ fully computed — greatly increasing the size of the proof – since the client will only compute some of them but needs all of them to verify the hash. We can avoid this issue by replacing the standard hash with a Merkel tree where the leaves are the hashed T_i values and the root is the challenge hash used in the proof. We can then send the 160 bit or 256 bit intermediate nodes instead of the 1024 bit T_i values, allowing the verifier to compute *only* a subset of the T_i values and yet still validate the proof against the challenge without drastically increasing the proof size.

B. Limited Anonymity and Forward Security

A serious concern in the Bitcoin community is the loss of wallets due to poor endpoint security. In traditional Bitcoin, this results in the theft of coins [4]. However, in the Zerocoin setting it may also allow an attacker to *de-anonymize* Zerocoin transactions using the stored skc. The obvious solution is to securely delete skc immediately after a coin is spent. Unfortunately, this provides no protection if skc is stolen at some earlier point.

One solution is to generate the spend transaction immediately (or shortly after) the coin is minted, possibly using an earlier checkpoint for calculating $\mathbf{C}$. This greatly reduces the user's anonymity by decreasing the number of coins in $\mathbf{C}$ and leaking some information about when the coin was minted. However, no attacker who compromises the wallet can link any zerocoins in it to their mint transactions.

C. Code Changes

For our implementation, we chose to modify `bitcoind`, the original open-source Bitcoin C++ client. This required several modifications. First, we added instructions to the Bitcoin script for minting and spending zerocoins. Next, we added transaction types and code for handling these new instructions, as well as maintaining the list of spent serial numbers and the accumulator. We used the Charm cryptographic framework [27] to implement the cryptographic constructions in Python, and we used Boost's Python utilities to call that code from within `bitcoind`. This introduces some performance overhead, but it allowed us to rapidly prototype and leave room for implementing future constructions as well.

D. Incremental Deployment

As described above, Zerocoin requires changes to the Bitcoin protocol that must happen globally: while transactions containing the new instructions will be validated by updated servers, they will fail validation on older nodes, potentially causing the network to split when a block is produced that validates for some, but not all, nodes. Although this is not the first time Bitcoin has faced this problem, and there is precedent for a flag day type upgrade strategy [28], it is not clear how willing the Bitcoin community is to repeat it. As such, we consider the possibility of an incremental deployment.

One way to accomplish this is to embed the above protocol as comments in standard Bitcoin scripts. For non Zerocoin aware nodes, this data is effectively inert, and we can use Bitcoin's n of k signature support to specify that such comment embedded zerocoins are valid only if signed by some subset of the Zerocoin processing nodes. Such Zerocoin aware nodes can parse the comments and charge transaction fees for validation according to the proofs embedded in the comments, thus providing an incentive for more nodes to provide such services. Since this only changes the validation mechanism for Zerocoin, the Anonymity property holds as does the Balance property if no more than $n - 1$ Zerocoin nodes are malicious.

Some care must be taken when electing these nodes to prevent a Sybil attack. Thankfully, if we require that such a node also produce blocks in the Bitcoin block chain, we have

[16]The reference `bitcoind` client stores transactions as STL Vectors, which require contiguous segments of memory. As such, storing Zerocoin proofs in the transaction might cause memory issues far faster than expected.

[17]Furthermore, this solution allows for the intriguing possibility that proofs be allowed to vanish after they have been sufficiently verified by the network and entombed in the block chain. However, it is not clear how this interacts with Bitcoin in theory or practice.

a decent deterrent. Furthermore, because any malfeasance of these nodes is readily detectable (since they signed an invalid Zerocoin transaction), third parties can audit these nodes and potentially hold funds in escrow to deter fraud.

VI. Real World Security and Parameter Choice

A. Anonymity of Zerocoin

Definition 3.2 states that given two Zerocoin mints and one spend, one cannot do much better than guess which minted coin was spent. Put differently, an attacker learns no more from our scheme than they would from observing the mints and spends of some ideal scheme. However, even an ideal scheme imposes limitations. For example, consider a case where N coins are minted, then all N coins are subsequently spent. If another coin is minted after this point, the size of the anonymity set for the next spend is $k = 1$, not $k = 11$, since it is clear to all observers that the previous coins have been used. We also stress that — as in many anonymity systems — privacy may be compromised by an attacker who mints a large fraction of the active coins. Hence, a lower bound on the anonymity provided is the number of coins minted by honest parties between a coin's mint and its spend. An upper bound is the total set of minted coins.

We also note that Zerocoin reveals the number of minted and spent coins to all users of the system, which provides a potential source of information to attackers. This is in contrast to many previous e-cash schemes which reveal this information primarily to merchants and the bank. However, we believe this may be an advantage rather than a loss, since the bank is generally considered an adversarial party in most e-cash security models. The public model of Zerocoin actually removes an information asymmetry by allowing users to determine when such conditions might pose a problem.

Lastly, Zerocoin does not hide the denominations used in a transaction. In practice, this problem can be avoided by simply fixing one or a small set of coin denominations and exchanging coins until one has those denominations, or by simply using Zerocoin to anonymize bitcoins.

B. Parameters

Generally, cryptographers specify security in terms of a single, adjustable security parameter λ. Indeed, we have used this notation throughout the previous sections. In reality, however, there are three distinct security choices for Zerocoin which affect either the system's anonymity, its resilience to counterfeiting, or both. These are:

1) The size of the Schnorr group used in the coin commitments.
2) The size of the RSA modulus used in the accumulator.
3) λ_{zkp}, the security of the zero-knowledge proofs.

Commitments. Because Pedersen commitments are information theoretically hiding for *any* Schnorr group whose order is large enough to fit the committed values, the size of the group used does not affect the long term anonymity of Zerocoin. The security of the commitment scheme does, however, affect counterfeiting: an attacker who can break the binding property of the commitment scheme can mint a zerocoin that opens to at least two different serial numbers, resulting in a double spend. As a result, the Schnorr group must be large enough that such an attack cannot be feasibly mounted in the lifetime of a coin. On the other hand, the size of the signature of knowledge π used in coin spends increases linearly with the size of the Schnorr group.

One solution is to minimize the group size by announcing fresh parameters for the commitment scheme periodically and forcing old zerocoins to expire unless exchanged for new zerocoins minted under the fresh parameters.[18] Since all coins being spent on the network at time t are spent with the current parameters and all previous coins can be converted to fresh ones, this does not decrease the anonymity of the system. It does, however, require users to convert old zerocoins to fresh ones before the old parameters expire. For our prototype implementation, we chose to use 1024 bit parameters on the assumption that commitment parameters could be regenerated periodically. We explore the possibility of extensions to Zerocoin that might enable smaller groups in Section IX.

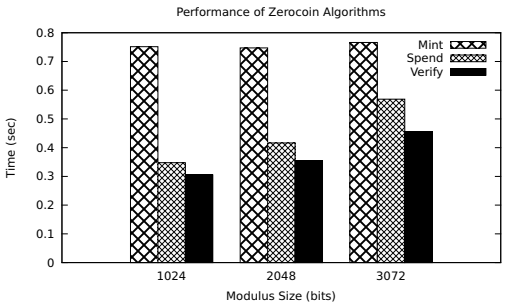
Accumulator RSA key. Because generating a new accumulator requires either a new trusted setup phase or generating a new RSA UFO [26], we cannot re-key very frequently. As a result, the accumulator is long lived, and thus we truly need long term security. Therefore we currently propose an RSA key of at least 3072 bits. We note that this does not greatly affect the size of the coins themselves, and, because the proof of accumulator membership is efficient, this does not have a large adverse effect on the overall coin spend proof size. Moreover, although re-keying the accumulator is expensive, it need not reduce the anonymity of the system since the new parameters can be used to re-accumulate the existing coin set and hence anonymize spends over that whole history.

Zero-knowledge proof security λ_{zkp}. This parameter affects the anonymity and security of the zero-knowledge proof. It also greatly affects the size of the spend proof. Thankfully, since each proof is independent, it applies per proof and therefore per spend. As such, a dishonest party would have to expend roughly $2^{\lambda_{zkp}}$ effort to forge a *single* coin or could link a *single* coin mint to a spend with probability roughly $\frac{1}{2^{\lambda_{zkp}}}$. As such we pick $\lambda_{zkp} = 80$ bits.

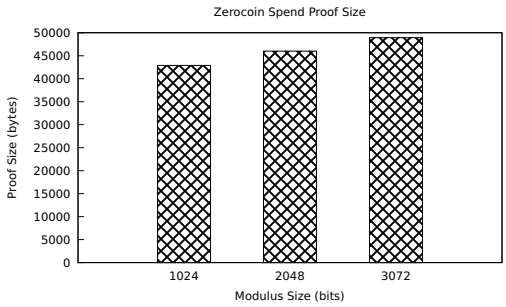
VII. Performance

To validate our results, we conducted several experiments using the modified `bitcoind` implementation described in Section V. We ran our experiments with three different

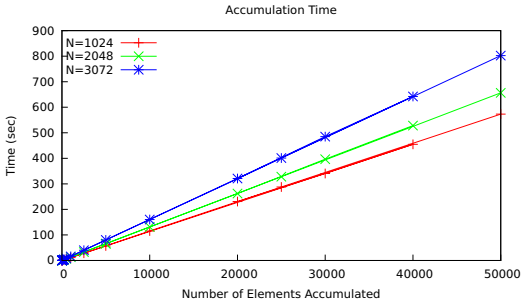
[18] Note that this conversion need not involve a full spend of the coins. The user may simply reveal the trapdoor for the old coin, since the new zerocoin will still be unlinkable when properly spent.



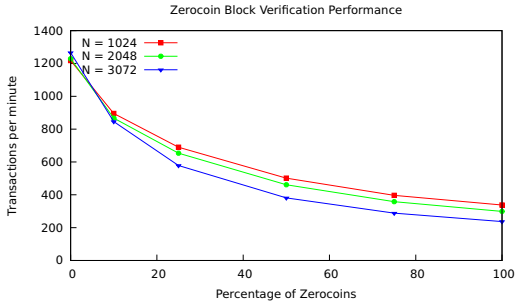
(a) Times for a single Zerocoin operation measured in seconds. These operations do not include the time required to compute the accumulator.



(b) Zerocoin proof sizes measured in bytes as a function of RSA modulus size.



(c) Time required to accumulate x elements. Note, this cost is amortized when computing the global accumulator.



(d) Transaction verifications per minute as a function of the percentage of Zerocoin transactions in the network (where half are mints and half are spends). Note, since we plot the reciprocal of transaction time, this graph appears logarithmic even though Zerocoin scales linearly.

Figure 3: Zerocoin performance as a function of parameter size.

parameter sizes, where each corresponds to a length of the RSA modulus N: 1024 bits, 2048 bits, and 3072 bits.[19]

We conducted two types of experiments: (1) *microbenchmarks* that measure the performance of our cryptographic constructions and (2) tests of our whole modified Bitcoin client measuring the time to verify Zerocoin carrying blocks. The former gives us a reasonable estimate of the cost of minting a single zerocoin, spending it, and verifying the resulting transaction. The latter gives us an estimate of Zerocoin's impact on the existing Bitcoin network and the computational cost that will be born by each node that verifies Zerocoin transactions.

All of our experiments were conducted on an Intel Xeon E3-1270 V2 (3.50GHz quad-core processor with hyper-threading) with 16GB of RAM, running 64-bit Ubuntu Server 11.04 with Linux kernel 2.6.38.

[19]These sizes can be viewed as *roughly* corresponding to a discrete logarithm/factorization security level of 2^{80}, 2^{112}, and 2^{128} respectively. Note that the choice of N determines the size of the parameter p. We select $|q|$ to be roughly twice the estimated security level.

A. Microbenchmarks

To evaluate the performance of our Mint, Spend, and Verify algorithms in isolation, we conducted a series of microbenchmarks using the Charm (Python) implementation. Our goal in these experiments was to provide a direct estimate of the performance of our cryptographic primitives.

Experimental setup. One challenge in conducting our microbenchmarks is the accumulation of coins in $\mathbf{C}$ for the witness in Spend$(\ldots)$ or for the global accumulator in both Spend$(\ldots)$ and Verify$(\ldots)$. This is problematic for two reasons. First, we do not know how large $\mathbf{C}$ will be in practice. Second, in our implementation accumulations are incremental. To address these issues we chose to break our microbenchmarks into two separate experiments. The first experiment simply computes the accumulator for a number of possible sizes of $\mathbf{C}$, ranging from 1 to 50,000 elements. The second experiment measures the runtime of the Spend$(\ldots)$ and Verify$(\ldots)$ routines with a precomputed accumulator and witness (A, ω).

We conducted our experiments on a single thread of the processor, using all three parameter sizes. All experiments

were performed 500 times, and the results given represent the average of these times. Figure 3a shows the measured times for computing the coin operations, Figure 3b shows the resulting proof sizes for each security parameter, and Figure 3c shows the resulting times for computing the accumulator. We stress that accumulation in our system is *incremental*, typically over at most the $200-500$ transactions in a block (which takes at worst eight seconds), and hence the cost of computing the global accumulator is therefore amortized. The only time one might accumulate 50,000 coins at one time would be when generating the witness for a very old zerocoin.

B. Block Verification

How Zerocoin affects network transaction processing determines its practicality and scalability. Like all transactions, Zerocoin spends must be verified first by the miner to make sure he is not including invalid transactions in a block and then again by the network to make sure it is not including an invalid block in the block chain. In both cases, this entails checking that $\mathsf{Verify}(\ldots) = 1$ for each Zerocoin transaction and computing the accumulator checkpoint.

We need to know the impact of this for two reasons. First, the Bitcoin protocol specifies that a new block should be created on average once every 10 minutes.[20] If verification takes longer than 10 minutes for blocks with a reasonable number of zerocoins, then the network cannot function.[21] Second, while the cost of generating these blocks and verifying their transactions can be offset by transaction fees and coin mining, the cost of verifying blocks prior to appending them to the block chain is only offset for mining nodes (who can view it as part of the cost of mining a new block). This leaves anyone else verifying the block chain with an uncompensated computational cost.

Experimental setup. To measure the effect of Zerocoin on block verification time, we measure how long it takes our modified `bitcoind` client to verify externally loaded test blocks containing 200, 400, and 800 transactions where 0, 10, 25, 75, or 100 percent of the transactions are Zerocoin transactions (half of which are mints and half are spends). We repeat this experiment for all three security parameters.

Our test data consists of two blocks. The first contains z Zerocoin mints that must exist for any spends to occur. The second block is our actual test vector. It contains, in a random order, z Zerocoin spends of the coins in the previous block, z Zerocoin mints, and s standard Bitcoin sendToAddress transactions. We measure how long the `processblock` call of the `bitcoind` client takes to verify the second block containing the mix of Zerocoin and classical Bitcoin transactions. For accuracy, we repeat these measurements 100 times and average the results. The results are presented in Figure 3d.

C. Discussion

Our results show that Zerocoin scales beyond current Bitcoin transaction volumes. Though we require significant computational effort, verification does not fundamentally threaten the operation of the network: even with a block containing 800 Zerocoin transactions — roughly double the average size of a Bitcoin block currently — verification takes less than five minutes. This is under the unreasonable assumption that all Bitcoin transactions are supplanted by Zerocoin transactions.[22] In fact, we can scale well beyond Bitcoin's current average of between 200 and 400 transactions per block [29] if Zerocoin transactions are not the majority of transactions on the network. If, as the graph suggests, we assume that verification scales linearly, then we can support a 50% transaction mix out to 350 transactions per minute (3,500 transactions per block) and a 10% mixture out to 800 transactions per minute (8,000 per block).

One remaining question is at what point we start running a risk of coin serial number collisions causing erroneous double spends. Even for our smallest serial numbers — 160 bits — the collision probability is small, and for the 256 bit serial numbers used with the 3072 bit accumulator, our collision probability is at worst equal to the odds of a collision on a normal Bitcoin transaction which uses SHA-256 hashes.

We stress several caveats about the above data. First, our prototype system does not exploit any parallelism either for verifying multiple Zerocoin transactions or in validating an individual proof. Since the only serial dependency for either of these tasks is the (fast) duplicate serial number check, this offers the opportunity for substantial improvement.

Second, the above data is not an accurate estimate of the financial cost of Zerocoin for the network: (a) it is an *overestimate* of a mining node's extra effort when verifying proposed blocks since in practice many transactions in a received block will already have been received and validated by the node as it attempts to construct its own contribution to the block chain; (b) execution time is a poor metric in the context of Bitcoin, since miners are concerned with actual monetary operating cost; (c) since mining is typically performed using GPUs and to a lesser extent FPGAs and ASICs, which are far more efficient at computing hash collisions, the CPU cost measured here is likely insignificant.

Finally, our experiment neglects the load on a node both from processing incoming transactions and from solving the proof of work. Again, we contend that most nodes will probably use GPUs for mining, and as such the latter is not an issue. The former, however, remains an unknown. At

[20]This rate is maintained by a periodic network vote that adjusts the difficulty of the Bitcoin proof of work.

[21]For blocks with unreasonable numbers of Zerocoin transaction we can simply extend `bitcoind`'s existing anti-DoS mechanisms to reject the block and blacklist its origin.

[22]In practice we believe Zerocoin will be used to anonymize bitcoins that will then be spent in actual transactions, resulting in far lower transaction volumes.

the very least it seems unlikely to disproportionately affect Zerocoin performance.

VIII. Previous Work

A. E-Cash and Bitcoin

Electronic cash has long been a research topic for cryptographers. Many cryptographic e-cash systems focus on user privacy and typically assume the existence of a semi-trusted coin issuer or bank. E-cash schemes largely break down into *online* schemes where users have contact with a bank or registry and *offline* schemes where spending can occur even without a network connection. Chaum introduced the first online cryptographic e-cash system [30] based on RSA signatures, later extending this work to the offline setting [31] by de-anonymizing users who double-spent. Many subsequent works improved upon these techniques while maintaining the requirement of a trusted bank: for example, by making coins divisible [32, 33] and reducing wallet size [34]. One exception to the rule above comes from Sander and Ta-Shma [35] who presciently developed an alternative model that is reminiscent of our proposal: the central bank is replaced with a hash chain and signatures with accumulators. Unfortunately the accumulator was not practical, a central party was still required, and no real-world system existed to compute the chain.

Bitcoin's primary goal, on the other hand, is not anonymity. It has its roots in a non-academic proposal by Wei Dai for a distributed currency based on solving computational problems [36]. In Dai's original proposal anyone could create currency, but all transactions had to be broadcast to all clients. A second variant limited currency generation and transaction broadcast to a set of servers, which is effectively the approach Bitcoin takes. This is a marked distinction from most, if not all, other e-cash systems since there is no need to select one or more trusted parties. There is a general assumption that a majority of the Bitcoin nodes are honest, but anyone can join a node to the Bitcoin network, and anyone can get the entire transaction graph. An overview of Bitcoin and some of its shortcomings was presented by Barber *et. al.* in [2].

B. Anonymity

Numerous works have shown that "pseudonymized" graphs can be re-identified even under passive analysis. Narayanan and Shmatikov [5] showed that real world social networks can be passively de-anonymized. Similarly, Backstrom *et al.* [37] constructed targeted attacks against anonymized social networks to test for relationships between vertices. Previously, Narayanan and Shmatikov de-anonymized users in the Netflix prize data set by correlating data from IMDB [38].

Bitcoin itself came into existence in 2009 and is now beginning to receive scrutiny from privacy researchers. De-anonymization techniques were applied effectively to Bitcoin even at its relatively small 2011 size by Reid and Harrigan [3]. Ron and Shamir examined the general structure of the Bitcoin network graph [1] after its nearly 3-fold expansion. Finally, we have been made privately aware of two other early-stage efforts to examine Bitcoin anonymity.

IX. Conclusion and Future Work

Zerocoin is a distributed e-cash scheme that provides strong user anonymity and coin security under the assumption that there is a distributed, online, append-only transaction store. We use Bitcoin to provide such a store and the backing currency for our scheme. After providing general definitions, we proposed a concrete realization based on RSA accumulators and non-interactive zero-knowledge signatures of knowledge. Finally, we integrated our construction into Bitcoin and measured its performance.

Our work leaves several open problems. First, although our scheme is workable, the need for a double-discrete logarithm proof leads to large proof sizes and verification times. We would prefer a scheme with both smaller proofs and greater speed. This is particularly important when it comes to reducing the cost of third-party verification of Zerocoin transactions. There are several promising constructions in the cryptographic literature, e.g., bilinear accumulators, mercurial commitments [13, 39]. While we were not able to find an analogue of our scheme using alternative components, it is possible that further research will lead to other solutions. Ideally such an improvement could produce a drop-in replacement for our existing implementation.

Second, Zerocoin currently derives both its anonymity and security against counterfeiting from strong cryptographic assumptions at the cost of substantially increased computational complexity and size. As discussed in section VI-B, anonymity is relatively cheap, and this cost is principally driven by the anti-counterfeiting requirement, manifesting itself through the size of the coins and the proofs used.

In Bitcoin, counterfeiting a coin is not computationally prohibitive, it is merely computationally costly, requiring the user to obtain control of at least 51% of the network. This provides a possible alternative to our standard cryptographic assumptions: rather than the strong assumption that computing discrete logs is infeasible, we might construct our scheme on the weak assumption that there is no financial incentive to break our construction as the cost of computing a discrete log exceeds the value of the resulting counterfeit coins.

For example, if we require spends to prove that fresh and random bases were used in the commitments for the corresponding mint transaction (e.g., by selecting the bases for the commitment from the hash of the coin serial number and proving that the serial number is fresh), then it appears that an attacker can only forge a *single* zerocoin per discrete log computation. Provided the cost of computing such a discrete log is greater than the value of a zerocoin, forging a coin is not profitable. How small this allows us to make

the coins is an open question. There is relatively little work comparing the asymptotic difficulty of solving multiple distinct discrete logs in a fixed group,[23] and it is not clear how theory translates into practice. We leave these questions, along with the security of the above proposed construction, as issues for future work.

Finally, we believe that further research could lead to different tradeoffs between security, accountability, and anonymity. A common objection to Bitcoin is that it can facilitate money laundering by circumventing legally binding financial reporting requirements. We propose that additional protocol modifications (e.g., the use of anonymous credentials [40]) might allow users to maintain their anonymity while demonstrating compliance with reporting requirements.

Acknowledgements. We thank Stephen Checkoway, George Danezis, and the anonymous reviewers for their helpful comments. The research in this paper was supported in part by the Office of Naval Research under contract N00014-11-1-0470, and DARPA and the Air Force Research Laboratory (AFRL) under contract FA8750-11-2-0211.

REFERENCES

[1] D. Ron and A. Shamir, "Quantitative Analysis of the Full Bitcoin Transaction Graph," Cryptology ePrint Archive, Report 2012/584, 2012, http://eprint.iacr.org/.

[2] S. Barber, X. Boyen, E. Shi, and E. Uzun, "Bitter to better – how to make bitcoin a better currency," in *Financial Cryptography 2012*, vol. 7397 of LNCS, 2012, pp. 399–414.

[3] F. Reid and M. Harrigan, "An analysis of anonymity in the Bitcoin system," in *Privacy, security, risk and trust (PASSAT), 2011 IEEE Third Internatiojn Conference on Social Computing (SOCIALCOM)*. IEEE, 2011, pp. 1318–1326.

[4] T. B. Lee, "A risky currency? Alleged $500,000 Bitcoin heist raises questions," Available at http://arstechnica.com/, June 2011.

[5] A. Narayanan and V. Shmatikov, "De-anonymizing social networks," in *Security and Privacy, 2009 30th IEEE Symposium on*. IEEE, 2009, pp. 173–187.

[6] "Bitcoin fog company," http://www.bitcoinfog.com/.

[7] "The Bitcoin Laundry," http://www.bitcoinlaundry.com/.

[8] "Blind Bitcoin," Information at https://en.bitcoin.it/wiki/Blind_Bitcoin_Transfers.

[9] [Online]. Available: https://www.torproject.org/

[10] J. Benaloh and M. de Mare, "One-way accumulators: a decentralized alternative to digital signatures," in *EUROCRYPT '93*, vol. 765 of LNCS, 1994, pp. 274–285.

[11] N. Barić and B. Pfitzmann, "Collision-free accumulators and fail-stop signature schemes without trees," in *EUROCRYPT '97*, vol. 1233 of LNCS, 1997, pp. 480–494.

[23] We note that both SSH and the Internet Key Exchange protocol used in IPv6 use fixed Diffie-Hellman parameters.

[12] J. Camenisch and A. Lysyanskaya, "Dynamic accumulators and application to efficient revocation of anonymous credentials," in *CRYPTO '02*, 2002, pp. 61–76.

[13] L. Nguyen, "Accumulators from bilinear pairings and applications," in *Topics in Cryptology – CT-RSA 2005*, 2005, vol. 3376 LNCS, pp. 275–292.

[14] J. Camenisch, M. Kohlweiss, and C. Soriente, "An accumulator based on bilinear maps and efficient revocation for anonymous credentials," in *PKC '09*, vol. 5443 of LNCS, 2009, pp. 481–500.

[15] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system, 2009," 2012. [Online]. Available: http://www.bitcoin.org/bitcoin.pdf

[16] G. O. Karame, E. Androulaki, and S. Capkun, "Two bitcoins at the price of one? double-spending attacks on fast payments in bitcoin," Cryptology ePrint Archive, Report 2012/248, 2012, http://eprint.iacr.org/.

[17] European Central Bank, "Virtual currency schemes," Available at http://www.ecb.europa.eu/pub/pdf/other/virtualcurrencyschemes201210en.pdf, October 2012.

[18] C.-P. Schnorr, "Efficient signature generation for smart cards," *Journal of Cryptology*, vol. 4, no. 3, pp. 239–252, 1991.

[19] R. Cramer, I. Damgård, and B. Schoenmakers, "Proofs of partial knowledge and simplified design of witness hiding protocols," in *CRYPTO '94*, vol. 839 of LNCS, 1994, pp. 174–187.

[20] J. Camenisch and M. Michels, "Proving in zero-knowledge that a number n is the product of two safe primes," in *EUROCRYPT '99*, vol. 1592 of LNCS, 1999, pp. 107–122.

[21] J. L. Camenisch, "Group signature schemes and payment systems based on the discrete logarithm problem," Ph.D. dissertation, ETH Zürich, 1998.

[22] S. Brands, "Rapid demonstration of linear relations connected by boolean operators," in *EUROCRYPT '97*, vol. 1233 of LNCS, 1997, pp. 318–333.

[23] A. Fiat and A. Shamir, "How to prove yourself: Practical solutions to identification and signature problems," in *CRYPTO '86*, vol. 263 of LNCS, 1986, pp. 186–194.

[24] M. Chase and A. Lysyanskaya, "On signatures of knowledge," in *CRYPTO'06*, vol. 4117 of LNCS, 2006, pp. 78–96.

[25] J. Camenisch and M. Stadler, "Efficient group signature schemes for large groups," in *CRYPTO '97*, vol. 1296 of LNCS, 1997, pp. 410–424.

[26] T. Sander, "Efficient accumulators without trapdoor extended abstract," in *Information and Communication Security*, vol. 1726 of LNCS, 1999, pp. 252–262.

[27] J. A. Akinyele, C. Garman, I. Miers, M. W. Pagano, M. Rushanan, M. Green, and A. D. Rubin, "Charm: A framework for rapidly prototyping cryptosystems," *To appear, Journal of Cryptographic Engineering*, 2013. [Online]. Available: http://dx.doi.org/10.1007/s13389-013-0057-3

[28] [Online]. Available: https://en.bitcoin.it/wiki/BIP_0016

[29] [Online]. Available: http://blockchain.info/charts/n-transactions-per-block

[30] D. Chaum, "Blind signatures for untraceable payments," in *CRYPTO '82*. Plenum Press, 1982, pp. 199–203.

[31] D. Chaum, A. Fiat, and M. Naor, "Untraceable electronic cash," in *CRYPTO 88*, 1990, vol. 403 of LNCS, pp. 319–327.

[32] T. Okamoto and K. Ohta, "Universal electronic cash," in *CRYPTO 91*, 1992, vol. 576 of LNCS, pp. 324–337.

[33] T. Okamoto, "An efficient divisible electronic cash scheme," in *Crypt '95*, 1995, vol. 963 of LNCS, pp. 438–451.

[34] J. Camenisch, S. Hohenberger, and A. Lysyanskaya, "Compact e-cash," in *EUROCRYPT '05*, 2005, vol. 3494 of LNCS, pp. 566–566.

[35] T. Sander and A. Ta-Shma, "Auditable, anonymous electronic cash (extended abstract)," in *CRYPTO '99*, vol. 1666 of LNCS, 1999, pp. 555–572.

[36] W. Dai. B-money proposal. [Online]. Available: http://www.weidai.com/bmoney.txt

[37] L. Backstrom, C. Dwork, and J. Kleinberg, "Wherefore art thou r3579x?: Anonymized social networks, hidden patterns, and structural steganography," in *Proceedings of the 16th international conference on World Wide Web*, ser. WWW '07. New York, NY, USA: ACM, 2007, pp. 181–190.

[38] A. Narayanan and V. Shmatikov, "Robust de-anonymization of large sparse datasets," in *IEEE Symposium on Security and Privacy*. IEEE, 2008, pp. 111–125.

[39] M. Chase, A. Healy, A. Lysyanskaya, T. Malkin, and L. Reyzin, "Mercurial commitments with applications to zero-knowledge sets," in *EUROCRYPT '05*, vol. 3494, 2005, pp. 422–439.

[40] J. Camenisch and A. Lysyanskaya, "An efficient system for non-transferable anonymous credentials with optional anonymity revocation," in *EUROCRYPT '01*, vol. 2045 of LCNS, 2001, pp. 93–118.

[41] ——, "Dynamic accumulators and application to efficient revocation of anonymous credentials," in *CRYPTO '02*, 2002, extended Abstract. [Online]. Available: http://cs.brown.edu/~anna/papers/camlys02.pdf

[42] D. Pointcheval and J. Stern, "Provably secure blind signature schemes," in *ASIACRYPT '96*, vol. 1163 of LNCS, 1996, pp. 252–265.

APPENDIX A. SECURITY PROOFS

A. Proof Sketch of Theorem 4.1

Proof sketch. Consider the following simulation. First, the simulation generates $params \leftarrow \mathsf{Setup}(1^\lambda)$ and two primes C_0, C_1 that are uniformly sampled from the set of prime numbers in the range $[\mathcal{A}, \mathcal{B}]$.[24] $\mathcal{A}_1$ takes these values as input and outputs a set $\mathbf{C}$ and transaction string R using any strategy it wishes. Next the simulation runs $\mathcal{A}_2$ with a simulated[25] zero-knowledge signature of knowledge π and a random coin serial number S sampled from $\mathbb{Z}_q^*$. Note that if π is at least computationally zero-knowledge then with all but negligible probability, all values provided to $\mathcal{A}$ are distributed as in the real protocol. Moreover, all are independent of the bit b. By implication, $\mathbf{Pr}[\, b = b' \,] = 1/2 + \nu(\lambda)$ and $\mathcal{A}$'s advantage is negligible. □

[24] "Where $\mathcal{A}$ and $\mathcal{B}$ can be chosen with arbitrary polynomial dependence on the security parameter, as long as $2 < \mathcal{A}$ and $\mathcal{B} < \mathcal{A}^2$." [41] For a full description, see [41, §3.2 and §3.3].

B. Proof of Theorem 4.2

Proof: Let $\mathcal{A}$ be an adversary that wins the Balance game with non-negligible advantage ϵ. We construct an algorithm $\mathcal{B}$ that takes input (p, q, g, h), where $\mathbb{G} = \langle g \rangle = \langle h \rangle$ is a subgroup of $\mathbb{Z}_p^*$ of order q, and outputs $x \in \mathbb{Z}_q$ such that $g^x \equiv h \pmod{p}$. $\mathcal{B}$ works as follows:

On input (p, q, g, h), first generate accumulator parameters N, u as in the Setup routine and set $params \leftarrow (N, u, p, q, g, h)$. For $i = 1$ to K, compute $(c_i, skc_i) \leftarrow \mathsf{Mint}(params)$, where $skc_i = (S_i, r_i)$, and run $\mathcal{A}(params, c_1, \ldots, c_K)$. Answer each of $\mathcal{A}$'s queries to $\mathcal{O}_{spend}$ using the appropriate trapdoor information. Let $(S_1, R_1), \ldots, (S_l, R_l)$ be the set of values recorded by the oracle.

At the conclusion of the game, $\mathcal{A}$ outputs a set of M coins $(c'_1, \ldots, c'_M)$ and a corresponding set of $M + 1$ valid tuples $(\pi'_i, S'_i, R'_i, \mathbf{C}'_i)$. For $j = 1$ to $M+1$, apply the ZKSoK extractor to the j^{th} zero-knowledge proof π'_j to extract the values (c^*_j, r^*_j) and perform the following steps:

1) If the extractor fails, abort and signal $\text{EVENT}_{\text{EXT}}$.
2) If $c^*_j \notin \mathbf{C}'_j$, abort and signal $\text{EVENT}_{\text{ACC}}$.
3) If $c^*_j \in \{c_1, \ldots, c_K\}$:
   a) If for some i, $(S'_j, r^*_j) = (S_i, r_i)$ and $R'_j \neq R_i$, abort and signal $\text{EVENT}_{\text{FORGE}}$.
   b) Otherwise if for some i, $(S'_j, r^*_j) = (S_i, r_i)$, abort and signal $\text{EVENT}_{\text{COL}}$.
   c) Otherwise set $(a, b) = (S_i, r_i)$.
4) If for some i, $c^*_j = c^*_i$, set $(a, b) = (S'_i, r^*_i)$.

If the simulation did not abort, we now have $(c^*_j, r^*_j, S'_j, a, b)$ where (by the soundness of π) we know that $c^*_j \equiv g^{S'_j} h^{r^*_j} \equiv g^a h^b \pmod{p}$. To solve for $\log_g h$, output $(S'_j - a) \cdot (b - r'_j)^{-1} \bmod q$.

Analysis. Let us briefly explain the conditions behind this proof. When the simulation does *not* abort, we are able to extract $(c^*_1, \ldots, c^*_{M+1})$ where the win conditions enforce that $\forall j \in [1, M+1]$, $c^*_j \in \mathbf{C}'_j \in \{c_1, \ldots, c_K, c'_1, \ldots, c'_M\}$ and each S'_j is distinct (and does not match any serial number output by $\mathcal{O}_{spend}$). Since $\mathcal{A}$ has produced M coins and yet spent $M + 1$, there are only two possibilities:

1) *$\mathcal{A}$ has spent one of the challenger's coins but has provided a new serial number for it.* For some (i, j),

[25] Our proofs assume the existence of an efficient simulator and extractor for the ZKSoK. See Appendix B.

$c_j^* = c_i \in \{c_1, \ldots, c_K\}$. Observe that in cases where the simulation does not abort, the logic of the simulation always results in a pair $(a, b) = (S_i, r_i)$ where $g^a h^b \equiv g^{S'_j} h^{r^*_j} \equiv c_j^* \pmod p$ and $(a, b) \neq (S'_j, r^*_j)$.

2) $\mathcal{A}$ *has spent the same coin twice.* For some (i, j), $c_j^* = c_i^*$ and yet $(S'_j \neq S'_i)$. Thus again we identify a pair $(a, b) = (S'_i, r^*_i)$ that satisfies $g^a h^b \equiv c_j^* \pmod p$ where $(a, b) \neq (S'_j, r^*_j)$.

Finally, we observe that given any such pair (a, b) we can solve for $x = \log_g h$ using the equation above.

Abort probability. It remains only to consider the probability that the simulation aborts. Let $\nu_1(\lambda)$ be the (negligible) probability that the extractor fails on input π. By summation, $\mathbf{Pr}[\text{EVENT}_{\text{EXT}}] \leq (M+1)\nu_1(\lambda)$. Next consider the probability of $\text{EVENT}_{\text{COL}}$. This implies that for some i, $\mathcal{A}$ has produced a pair $(S'_j, r^*_j) = (S_i, r_i)$ where S'_j has *not* been produced by $\mathcal{O}_{spend}$. Observe that there are l distinct pairs (S, r) that satisfy $c_j^* = g^S h^r \bmod p$ and $\mathcal{A}$'s view is independent of the specific pair chosen. Thus $\mathbf{Pr}[\text{EVENT}_{\text{COL}}] \leq 1/l$.

Next, we argue that under the Strong RSA and Discrete Log assumptions, $\mathbf{Pr}[\text{EVENT}_{\text{ACC}}] \leq \nu_2(\lambda)$ and $\mathbf{Pr}[\text{EVENT}_{\text{FORGE}}] \leq \nu_3(\lambda)$. We show this in Lemmas A.1 and A.2 below. If $\mathcal{A}$ succeeds with advantage ϵ, then by summing the above probabilities we show that $\mathcal{B}$ succeeds with probability $\geq \epsilon - ((M+1)\nu_1(\lambda) + \nu_2(\lambda) + \nu_3(\lambda) + 1/l)$. We conclude with the remaining Lemmas.

Lemma A.1: Under the Strong RSA assumption, $\mathbf{Pr}[\text{EVENT}_{\text{ACC}}] \leq \nu_2(\lambda)$.

Proof sketch. The basic idea of this proof is that an $\mathcal{A}'$ who induces $\text{EVENT}_{\text{ACC}}$ with non-negligible probability can be used to find a witness ω to the presence of a non-member in a given accumulator. Given this value, we apply the technique of [12, §3] to solve the Strong RSA problem. For the complete details we refer the reader to [12, §3] and simply outline the remaining details of the simulation.

Let $\mathcal{A}'$ be an adversary that induces $\text{EVENT}_{\text{ACC}}$ with non-negligible probability ϵ' in the simulation above. We use $\mathcal{A}'$ to construct a Strong RSA solver $\mathcal{B}'$ that succeeds with non-negligible probability. On input a Strong RSA instance (N, u), $\mathcal{B}'$ selects (p, q, g, h) as in Setup and sets $params = (N, u, p, q, g, h)$. It generates $(c_1, \ldots, c_K)$ as in the previous simulation and runs $\mathcal{A}'$. To induce $\text{EVENT}_{\text{ACC}}$, $\mathcal{A}'$ produces valid output $(\pi', \mathbf{C}')$ and (by extraction from π') a $c^* \notin \mathbf{C}'$. $\mathcal{B}'$ now extracts ω^* from π' using the technique described in [12, §3] and uses the resulting value to compute a solution to the Strong RSA instance. □

Lemma A.2: Under the Discrete Logarithm assumption, $\mathbf{Pr}[\text{EVENT}_{\text{FORGE}}] \leq \nu_3(\lambda)$.

Proof sketch. We leave a proof for the full version of this paper, but it is similar to those used by earlier schemes, e.g., [25]. Let $\mathcal{A}'$ be an adversary that induces $\text{EVENT}_{\text{FORGE}}$ with non-negligible probability ϵ' in the simulation above. On input a discrete logarithm instance, we run $\mathcal{A}'$ as in the main simulation except that we do not use the trapdoor information to answer $\mathcal{A}'$'s oracle queries. Instead we select random serial numbers and simulate the ZKSoK responses to $\mathcal{A}'$ by programming the random oracle. When $\mathcal{A}'$ outputs a forgery on a repeated serial number but a different string R' than used in any previous proof, we rewind $\mathcal{A}'$ to extract the pair (S'_j, r^*_j) and solve for the discrete logarithm as in the main simulation. □

■

APPENDIX B.
ZERO-KNOWLEDGE PROOF CONSTRUCTION

The signature of knowledge

$$\pi = \mathsf{ZKSoK}[R]\{(c, w, r) : \mathsf{AccVerify}((N, u), A, c, w) = 1 \ \wedge \ c = g^S h^r\}$$

is composed of two proofs that (1) a committed value c is accumulated and (2) that c is a commitment to S. The former proof is detailed in [41, §3.3 and Appendix A]. The latter is a double discrete log signature of knowledge that, although related to previous work [21, §5.3.3], is new (at least to us). A proof of its security can be found in the full version of this paper. It is constructed as follows:

Given $y_1 = g^{a^x b^z} h^w$.
Let $l \leq k$ be two security parameters and $H : \{0,1\}^* \to \{0,1\}^k$ be a cryptographic hash function. Generate $2l$ random numbers $r_1, \ldots, r_l$ and $v_1, \ldots, v_l$. Compute, for $1 \leq i \leq l$, $t_i = g^{a^x b^{r_i}} h^{v_i}$. The signature of knowledge on the message m is $(c, s_1, s_2, \ldots, s_l, s'_1, s'_2, \ldots, s'_l)$, where:

$$c = H(m \| y_1 \| a \| b \| g \| h \| x \| t_1 \| \ldots \| t_l)$$

and

$$\textbf{if } c[i] = 0 \textbf{ then } s_i = r_i, s'_i = v_i; \quad \textbf{else } s_i = r_i - z, s'_i = v_i - w b^{r_i - z};$$

To verify the signature it is sufficient to compute:

$$c' = H(m \| y_1 \| a \| b \| g \| h \| x \| \bar{t_1} \| \ldots \| \bar{t_l})$$

with

$$\textbf{if } c[i] = 0 \textbf{ then } \bar{t_i} = g^{a^x b^{s_i}} h^{s'_i}; \quad \textbf{else } \bar{t_i} = y_1^{b^{s_i}} h^{s'_i};$$

and check whether $c = c'$.

Simulating and extracting. Our proofs in Appendix A assume the existence of an efficient simulator and extractor for the signature of knowledge. These may be constructed using well-understood results in the random oracle model, e.g., [25, 42]. We provide further details in the full version of this work.