Decentralized Anonymous Payments

by

Ian Miers

A dissertation submitted to The Johns Hopkins University in conformity with the requirements for the degree of Doctor of Philosophy.

Baltimore, Maryland

August, 2017

Abstract

Decentralized payment systems such as Bitcoin record monetary transactions between pseudonyms in an append-only ledger known as a blockchain. Because the ledger is public, permanent, and readable by anyone, a user's privacy depends solely on the difficulty of linking pseudonymous transactions either to each other or to real identities. Both academic work and commercial services have shown that such linking is, in fact, very easy. Anyone at any point in the future can download a user's transaction history and analyze it. In this work, we propose and implement privacy preserving coins, payments, and payment channels that can be built atop a ledger.

In particular we propose:

Zerocoin A blockchain based protocol for breaking the link between a transaction that receives non-anonymous funds and the subsequent transaction that spends it.

Zerocash The successor to Zerocoin, a blockchain based payment system supporting anonymous payments of arbitrary hidden value to other parties. While payments are recorded publicly in the blockchain, they reveal almost nothing else: the

recipient learns only the amount paid but not the source and anyone else learns only that a payment of some value to someone took place.

Bolt A payment channel protocol that allows two parties to anonymously and securely make many unlinkable payments while only posting two messages to the blockchain. This protocol provides for instant payments while providing drastically improved scalability as every transaction is no longer recorded in the blockchain.

Primary Reader: Matthew Green

Secondary Readers: Abhishek Jain, Aviel Rubin

Acknowledgments

This work would not have been possible without the help of a number of people: friends, family, and collaborators. I would like to thank Avi Rubin for his support and mentorship especially when I started. I would also like to thank my fellow graduate students, Christina Garman, Michael Rushanan, and Gabriel Kaptchuk, both for their ideas and making the lab a home for the past 6 years. I would particularly like to thank my advisor Matthew Green, for getting me started in this, for his advice, his contributions, and above all never being too busy to take the time for us to throw things at a white board. Some of them actually stuck.

Contents

List of Figures

Chapter 1

Introduction

Bitcoin has rapidly advanced from a toy system to one worth 35 billion dollars as of May 2017. Regardless of what happens to Bitcoin or crypto-currencies in general, the technical mechanisms underpinning it are worthy of study, not because of its success in and of itself, but because that success indicates that, somewhat surprisingly, the underlying paradigm is viable. Unlike almost all previous proposals for electronic cash, and indeed much of cryptography and computer security, Bitcoin does not depend on a central trusted authority. Instead of a trusted central bank or server that manages funds, every transaction is contained in a public, append-only, auditable ledger known as a blockchain. This blockchain is maintained by a peer-to-peer network under the assumption that some super-majority of its computational power is honest. This lack of trust enabled Bitcoin to grow organically without having to find a party or parties to trust with 35 billion dollars.

This approach need not be limited to payments. It can be used to manage identities and provide other services in peer-to-peer networks and other settings where there is no trusted party or finding and selecting such a party is challenging. Even where there is such a party, a blockchain can be used to minimize the security requirements for that party and make it auditable. Indeed, even if Bitcoin ultimately fails, basing systems on a public append-only log is an approach clearly worth exploring. However, this very feature—that all transactions are public and auditable—is also the

greatest limitation to Bitcoin and blockchains in general.

Since all transactions are recorded in a public ledger between pseudonyms, data mining can reveal users' spending habits to anyone by linking pseudonymous transactions together and to a user's real identity. In addition to many academic papers on the topic (e.g. [1, 2]), there are several companies in the business of providing such analysis [3, 4]. Deanonymization is a problem not just for individuals who value their privacy but for businesses who wish to keep suppliers, employee salaries, and revenue confidential from their competitors. Beyond being a privacy impediment, deanonymization is also an economic hindrance. Because payments have a public history, they are not necessarily fungible, i.e., one unit of currency is not necessarily interchangeable with another. For example, funds which have in the past been associated with questionable activity may not be exchangeable with funds which have no such history. While this limitation may seem like a good thing, commerce is based on the fundamental assumption that money—if legitimately obtained—is worth its face value. When coins have a readily available history, the face value is not the only information that will be used in determining the value of the coin.

The lack of privacy and confidentiality is an inherent limitation of Bitcoin and any blockchain based approach which does not select completely trusted peers. Because the ad hoc peer-to-peer network needs to be tolerant of churn, it must be open for others to join. As a result, what is recorded in the blockchain is fundamentally public. Even if the records were somehow encrypted, the peers would need access to cleartext records. Since anyone can join the network, such protections become meaningless. Even if we discard the peer-to-peer requirement, anyone who wishes to audit the blockchain needs to read it. Thus a fully private blockchain is in most contexts no blockchain at all since it lacks many of the features that enabled Bitcoin's success. This lack of privacy is a fundamental consequence of recording everything in an auditable ledger.

Privacy and confidentiality, however, are not the only limitations to Bitcoin. A second major issue is latency. The ledger is updated in chunks called a block, and blocks must be confirmed by the network. In the case of Bitcoin, it takes 10 minutes to create a block and 60 minutes to

fully confirm one. While there are no doubt many performance improvements to be had and other crypto-currencies offer better performance, the very nature of a wide area peer-to-peer network makes it unlikely payments will ever be confirmed fast enough for most in-person transactions.

Finally, recording every transaction in a public ledger entails large scalability issues. As of summer 2017, limitations on the size of blocks in Bitcoin lead to a large backlog of nearly 165,000 transactions [5]. This block size limitation restricts Bitcoin to a maximum of 7 transactions per second with an average of 3 to 4 [6]. Of course, different crypto-currencies may have different performance characteristics, and there are a variety of proposals for at least alleviating Bitcoin's current scaling issues, including simply increasing the block size. However, large scale usage is still a major issue for all existing crypto-currencies. Visa's payment network, for example, handles an average of nearly 4500 transactions per second with peak capacity of 65,000. [6]. Because broadcasting and recording all transactions globally is an intrinsically expensive operation, achieving this kind of scale with conventional techniques is a challenge, particularly given the latency requirement for in-person payments.

Payment channels [7, 8] are a novel approach that sidesteps scaling the blockchain itself. Instead of using the blockchain to record every transaction, payment channels use the blockchain merely to resolve disputes. When two parties want to open a channel, they escrow funds on the blockchain. Those parties can then, without interacting with the blockchain, make payments between themselves simply by updating their split of the escrowed funds. They do so by invalidating the current split of funds, agreeing on a new split, and creating a transaction that pays each party the agreed split of the escrowed funds. The blockchain only needs to ensure that any transaction spending the escrowed funds is 1) signed by both parties; and 2) not based on an invalid split. As a result, individual payments are near instantaneous and do not take up space on the blockchain.

Unfortunately, payment channels do not solve Bitcoin's privacy issues and, in certain cases, exacerbate them. Payment channels still leak the relationship between participants to any observer

of the blockchain, and channels frequently leak the amount of money exchanged over the channel[1]. The only thing payment channels do obscure is the exact value and timing of individual payments. However, in many cases (e.g. payments to an oncologist) the existence of a set of payments and their aggregate amount is far more sensitive than the particular amount or timing of any individual payment.

Even worse, channels provide more information to intermediaries who are party to the channel. Because transactions between long-lived pseudonyms are visible to participants in the channel, patterns can be seen in repeated interactions that might ordinarily be obscured by the use of new pseudonymous addresses. As a result, channels offer slightly increased privacy from third party observers but potentially much less privacy from payment processors and merchants. Since channels use repeated interactions with fixed pseudonyms (the identities used to establish the channel), they present a privacy problem even if the underlying ledger is somehow made private. Consequently, payment channels require special privacy solutions.

Our Contribution In this work we propose and implement protocols for privacy preserving decentralized coins, payments, and payment channels which allow us to build decentralized anonymous payments. These techniques maintain Bitcoin's core identity as a public auditable ledger while resolving both the fundamental privacy and scaling issues.

1.1 Background and Related Work

1.1.1 Bitcoin, crypto-currencies, and blockchains

Bitcoin is a decentralized payment protocol and crypto-currency, the development of which dates to at least 2009 by a person or persons using the pseudonym Satoshi Nakamoto [9]. Bitcoin's core technical achievement is a public log of transactions maintained by an ad hoc peer-to-peer

[1]Many payment flows are unidirectional. Since closure is public, the rate of which a channel between two parties is closed and a new one established reveals the total cash flow.

network. This log is called the blockchain because it chains blocks of data together: each block contains the hash of the previous block along with a set of transactions moving funds between public keys known as addresses. The longest sequence of blocks containing valid transactions and a valid proof of work is considered the authoritative state of the blockchain. Since honest peers will only add onto valid chains, the combination of these two rules effectively ensures that if some super-majority[2] of peers in the network are honest, then the blockchain is correct. In Bitcoin, the proof of work is to find a nonce such that $\mathcal{H}(data||nonce) < T$ where $\mathcal{H}$ is a hash function and T is some target difficulty. Assuming the proper choice of $\mathcal{H}$, the only way to do this is by trial and error. Thus more computational power implies a greater chance of generating a block.

Although it was the first, Bitcoin is not the only crypto-currency. Many others exist that use different software, protocols, and proofs of work. Even a brief discussion of these is beyond the scope of this work.

1.1.2 E-cash

E-cash, first introduced by Chaum [11] and extended in subsequent work, e.g. [12, 13, 14], allows for private transactions between a customer, a merchant, and a trusted bank. The customer withdraws cryptographic tokens, often called *coins*, signed by the bank and later spends them with a merchant. E-cash guarantees that the customer cannot forge tokens and that the merchant, even if she colludes with the bank, cannot link withdrawal of the coins to when they were spent. Since the customer identifies themselves when withdrawing the funds from their account (but need not identify themselves to conduct the transaction), anonymity is ensured.

Even if the customer cannot forge a coin, they can copy it and try to spend it twice. This strategy is known as a double spend attack. To prevent it, e-cash schemes typically embed a unique serial number into each coin. The serial number is recorded when the coin is spent, and spending a

[2]The exact threshold is an open question. Originally it was believed to be anything over 50%. However, there is evidence that this threshold is not the case [10], and at this time determining the exact threshold is an active research area.

coin whose serial number is already recorded is prohibited.

The challenge in using e-cash with Bitcoin is that no trusted party is available to store the signing key needed to issue coins. Consequently, we must take a completely different approach.

1.2 Intuition: From public to unlinkable and then anonymous payments and channels[3]

Consider a physical "pencil and paper" analog to a blockchain, which we call bbcoin (bulletin board coin). All users share access to a physical bulletin board to which they can post messages. No user can delete, reorder, or overwrite a message posted to the bulletin board. Messages that are posted to the bulletin board must be valid according to some set of rules. These rules are enforced by the users who will remove anything that is invalid.

The goal of the various forms of bbcoin is to allow Alice to use the bulletin board to make private payments. In the "pencil and paper" model, we assume there is an underlying physical currency and that this currency is traceable. For example, assume that the underlying currency is denominated in 1 USD notes that can only be transferred by posting a message to the bulletin board specifying the recipient and affixing a dollar note. Because the bills have a unique serial number, anyone who watches the bulletin board can trace payments.

We now show, in a series of five steps, how to move from this system where payments are completely public, through one where the links between payments can be broken, to one where payments can be made that reveal neither the amount, the sender, nor the recipient.

Version 1: coins with hidden origin. In this version of bbcoin, we want to break the linkage between the funds Alice is paid and the funds she makes payments with. To do this, we allow Alice to exchange currency associated with her for special cryptographic tokens called coins that are not linked to her.

[3] A condensed version of this appeared in [15]. What follows here is an expanded intuitive version intended for a broader audience.

To create a coin, Alice generates a random coin serial number sn and commits to it with randomness r. She posts the resulting commitment cm to the bulletin board along with \$1 of the underlying currency. A commitment is the cryptographic equivalent of a sealed envelope. No one can see what is inside it, and Alice cannot change its contents once she seals the envelope. Later, using r, Alice can reveal the contents. In effect, she has committed in advance to a serial number she can later reveal. All users accept the posted transaction if cm is well formed and the attached funds are genuine. Alice has in effect exchanged a unit of traceable currency for a cryptographic token.

To complete the exchange and redeem the token, Alice first assembles a list CMList of all such commitments $\mathsf{cm}_0, \ldots, \mathsf{cm}_n$ that have been posted to the bulletin board by her or anyone else. She then generates a non-interactive zero-knowledge proof π that:

1. She knows randomness r opening some commitment cm to a serial number sn
2. $\mathsf{cm} \in \mathsf{CMList}$

While this proof intentionally reveals sn, it reveals nothing else and hides Alice's commitment in the set of all commitments. Next, under a new pseudonym and disguised, Alice posts the proof π to the bulletin board. If the proof is valid and the serial number sn has never appeared before on the bulletin board, Alice is allowed to take one unit of currency off the bulletin board. Crucially, that unit of currency need not be the same unit of currency Alice originally posted. In fact, Alice should choose the unit of currency she takes at random.

Although this protocol may seem rudimentary, it allows Alice to regain her privacy: she has replaced a dollar linked to her with one chosen at random for one of the other participants in the protocol. Because the zero-knowledge proof does not reveal which coin commitment she used, no one can link her redemption of the coin to the currency originally used to fund its creation. At the same time, because the proof is sound and the commitment scheme is binding, Alice cannot make coins out of thin air; she must always pay for them. Finally, because the serial number can only be used once, Alice cannot make multiple proofs using the same coin.

Version 2: coins with efficient expenditure. a.k.a Zerocoin. A naive proof that $\mathsf{cm} \in \mathsf{CMList} = \{\mathsf{cm}_0, \ldots, \mathsf{cm}_n\}$ takes the form of an "OR" proof that $\mathsf{cm} = \mathsf{cm}_1$ OR $\mathsf{cm} = \mathsf{cm}_2$ OR .... Such a proof's generation time and size grows linearly with CMList. Since coin commitments cannot be "dropped" from CMList because they are, by necessity, never identified when they are spent, this list will be very large.

What we need is a compact proof of set membership which is zero-knowledge. The general approach is to combine some set membership mechanism with a zero-knowledge proof system. The challenge is finding an efficient combination given the practical limitations of zero-knowledge proof techniques. There are at least two ways to make this compact set membership proof, and the choice of approach marks the first difference between Zerocoin and follow up work Zerocash.

In Zerocoin, compactness is achieved with a cryptographic accumulator [16] $A = u_0^{\mathsf{cm}_1 \cdot \mathsf{cm}_2 \cdot \ldots \cdot \mathsf{cm}_n} \mod N$ where N is an RSA modulus. The proof itself is constant size but requires linear computation. With careful optimization, most of this computation can be computed by the network.[4] When instantiated using Schnorr proofs [17], the proof is computationally practical. However, it is $\sim 25KB$ for an optimized implementation. This size is a major limitation.

The second approach, used by Zerocash, relies on a Merkle tree for a compact representation of CMList and efficient membership proofs whose size and generation time is logarithmic in the size of CMList. Thus, unlike Zerocoin, spending a coin does not require knowing the entire ledger's contents, only the appropriate path from the leaf of the Merkle tree to the root. We denote the Merkle tree's root $\mathsf{rt}(\mathsf{CMList})$ and the collision-resistant hash function used in the tree as CRH. The set-membership portion of π now consists of showing that cm is in the Merkle tree with root $\mathsf{rt}(\mathsf{CMList})$.

Using standard zero-knowledge proofs, this proof would be exceedingly large due the necessity of proofs over hash functions. However, advances in zero-knowledge proofs, specifically zk-SNARKs [18], enable a compact and efficiently verifiable proof.

Version 3: from coins to transfers. So far, bbcoin only allows Alice to exchange her own

[4] Users need only compute over the list of all coins added after the one they are spending.

currency. If she wishes to make a payment to Bob, she must then construct a separate non-anonymous transaction that pays him with the underlying currency. While Alice and Bob can both use fresh pseudonyms for this payment, it is still public and the value of the transaction is leaked.

Is it possible to skip this step? Instead of claiming the funds herself, can Alice give Bob the information necessary to construct the proof and redeem the currency himself? While this strategy is the right approach, we cannot use the existing scheme for two reason.

First, because Alice created the coin and knows its contents, she can spend it herself. Thus she could give the coin to Bob in exchange for goods or services and then spend the coin before Bob does. Second, since Alice knows the serial number of the coin, she can see when Bob spends it.

To solve the ownership issue, we include the identity of the recipient in the coin commitment. Each user gets a public *address*, $a_{\mathsf{pk}} := \mathsf{PRF}^{\mathrm{addr}}_{a_{\mathsf{sk}}}(0)$, where a_{sk} is the corresponding private key.[5] Coins destined to a user contain the user's address in the commitment in addition to the value. To redeem the coin, the recipient must prove he knows the private key a_{sk} corresponding to the public key included in the payment. This proof prevents Alice from stealing back the coin.

To solve the serial number privacy issue, coins no longer have an explicit serial number precomputed by their creator. Instead, the serial number needs to be computed by the recipient in a deterministic way that the creator of the coin cannot predict. We accomplish this by generating the serial number from a pseudorandom function keyed off the recipient's private key but evaluated on randomness ρ provided by the sender. This computation is denoted as $\mathsf{sn} := \mathsf{PRF}^{\mathrm{sn}}_{a_{\mathsf{sk}}}(\rho)$. Since only the recipient knows the proof key, the resulting serial number cannot be predicted by anyone else even if they know ρ. On the other hand, the process is deterministic, so the recipient cannot double spend the same coin by causing it to have two different serial numbers.

With these two changes, Bob can now safely receive payments from Alice and be assured that the money can neither be stolen or traced. He is then free to redeem the coin for a unit of currency himself.

[5] Users can have as many addresses as they want.

Version 4: from money orders to payments. While bbcoin now supports transfers between parties, we are still operating in a model where Alice converts one unit of currency to a coin, pays the coin to Bob, and then Bob converts it back to the underlying currency. This process is akin to money-orders where funds are converted back to the base currency at either end of the transaction.

When Bob wishes to make a payment to Charlie using the funds he received from Alice, he can simply skip the intermediary step. He posts the proof claiming the coin Alice gave him and simultaneously posts a new coin addressed to Charlie. Bob does not remove a unit of the underlying currency from the bulletin board since he is, in effect, paying it to Charlie. The other users need merely verify this proof and that the new coin is correctly formatted because they are already assured that Bob has "claimed" one unit of currency and immediately paid it to someone else.

Eventually some user, if they want, can take their coin and convert it to the underlying currency. However, there is no immediate need to do so. Coins now effectively function like currency backed by precious metal: they can be redeemed for backing value if needed, but it is not necessary or even practical to do so on a day-to-day basis. Of course, coins still need to be created initially with some underlying value, but that process only needs to be done once for each coin. The exact method by which it is done is an economic question, not a technical one.

Version 4: payments with arbitrary values. Instead of fixed value coins, the coin commitment cm can store both the serial number sn and the coin's value v. Alice can post the commitment along with v units of currency to the bulletin board. She must prove that the value inside the commitment corresponds to the value of the posted currency.

Naturally, if the coin is ever redeemed, the owner reveals the value v and takes the corresponding equivalent in base currency. Payments now can be made for arbitrary values.

This step introduces one additional issue: if Alice has a coin for \$100 and wishes to send \$25 to Bob, what happens to the remaining \$75? Alice seemingly must pay Bob the whole \$100. To fix this problem, we change the proof used to make a payment to one that produces two coins instead of one and prove that the sum of the two new coins does not exceed the sum of the coin being spent.

Thus Alice can post two coins and a proof to the bulletin board. The first coin pays \$75 to Alice as "change;" while the second pays Bob \$25.

With these modifications, a coin is now a tuple $\mathbf{c} := (a_{\mathsf{pk}}, v, \rho, r, s, \mathsf{cm})$. Knowledge of this tuple and address secret key a_{sk} is sufficient to claim the coin with the appropriate proof. The final proof $\pi_{\texttt{POUR}}$ posted to the bulletin board is for the following statement:

"Given the Merkle-tree root $\mathsf{rt}(\mathsf{CMList})$, serial number sn, and coin commitments $\mathsf{cm}_1, \mathsf{cm}_2$, one knows coins $\mathbf{c}, \mathbf{c}_1, \mathbf{c}_2$ and address secret key a_{sk} such that:

- The coins are well-formed: for $\mathbf{c}$ it holds that $k = \mathsf{COMM}_r(a_{\mathsf{pk}} \| \rho)$ and $\mathsf{cm} = \mathsf{COMM}_s(v \| k)$; and similarly for $\mathbf{c}_1$ and $\mathbf{c}_2$.
- The address secret key matches the public key: $a_{\mathsf{pk}} = \mathsf{PRF}^{\mathrm{addr}}_{a_{\mathsf{sk}}}(0)$.
- The serial number is computed correctly: $\mathsf{sn} := \mathsf{PRF}^{\mathrm{sn}}_{a_{\mathsf{sk}}}(\rho)$.
- The coin commitment cm appears as a leaf of a Merkle-tree with root $\mathsf{rt}(\mathsf{CMList})$.
- The values add up: $v_1 + v_2 + v_{\mathsf{pub}} = v$."

The resulting transaction $\mathsf{tx}_{\mathsf{Pour}} := (\mathsf{rt}, \mathsf{sn}, \mathsf{cm}_1, \mathsf{cm}_2, \pi_{\texttt{POUR}})$ is appended to the bulletin board. (As before, the transaction is rejected if the serial number sn appears in a previous transaction or the proof is invalid.) In the final construction, detailed later, we allow for coins to be merged as well as split. In fact, the number of input and output coins can be fixed at arbitrary values. For simplicity, we use 2 coins in 2 coins out.

Version 5: direct anonymous payments. To spend his freshly received payment of \$25, Bob needs to know the opening to the commitment. Alice could send this to Bob out of band. However, doing so requires Bob to be online and reachable via an anonymous communication channel.

To avoid that requirement, we allow Alice to use the bulletin board to send this information to Bob. We give each user a key-pair, $(\mathsf{pk}_{\mathsf{enc}}, \mathsf{sk}_{\mathsf{enc}})$ for an encryption scheme. When Alice pays Bob, she encrypts the necessary information for Bob to claim the coin. Bob scans the bulletin board and attempts to decrypt every message with his key. When he successfully decrypts a message, he stores it and adds the coins to his wallet.

However, a simple public key encryption scheme would be problematic since the ciphertext can leak what public key it is encrypted under and thus the identity of the recipient. As a result, multiple payments to the same party can be linked together.

1.2.1 Augmenting direct anonymous payments with anonymous channels

While the above system is completely private, it requires that every transaction be posted to the bulletin board. In our toy example this system may scale, but in a real world system scalability and latency are real issues.

Payment channels are a novel solution to this problem because they only use the blockchain for dispute resolution. In bbcoin we can think of channels as IOUs that are enforced by the rest of the room. To open a channel with Bob for $100, Alice and Bob agree on a unique channel ID and then both sign an IOU for "$100 to Alice from channel ID". Alice then posts a message declaring a channel with Bob under that ID and attaches $100. The channel can only be closed by the IOU signed by both Alice and Bob. If Alice wishes to pay Bob $5, she and Bob sign a new IOU for "$95 to Alice, $5 to Bob from channel ID". They then destroy the old IOU. Alice can close the channel by posting the most recent IOU for that channel.

We cannot merely compose Zerocash with existing payment channels, however, and get privacy. The channel itself identifies the participants. Thus multiple payments made on the same channel can be linked together. Bob always knows he is being paid by Alice. Even if Alice uses a pseudonym when establishing the channel, all payments on that channel are linked together via the pseudonym.

A fully private payment channel would prevent that linkage, allowing a customer and a merchant to exchange payments while obscuring the customer's identity from the merchant. Fundamentally, however, one party is always linked to the transactions. Such linkage is implicit in the notion of a channel: it has to be open with someone. If the customer initiates the channel, then she knows who she is paying every time she uses the channel. The merchant, on the other hand, will

only learn they are interacting with one of the people with whom they have an open channel. This situation is similar to a cash transaction at a store: the customer knows who they are paying, but the merchant does not know the customer. We assume merchants have well known identities and hence chose the terminology accordingly.[6]

Unidirectional payment channels Consider the following straw-man private channel protocol. To establish a channel, Alice withdraws classical e-cash coins issued by Bob. These tokens would be tied to a channel Alice opens on the bulletin board. Payments would take place as in standard e-cash with Bob playing the role of both the bank and the merchant: i.e. to pay Bob \$2 from a \$10 channel, Alice would send two \$1 coins to Bob. The channel would be closed by Alice "paying" her remaining 8 unused coins to herself and posting the resulting transactions to the bulletin board. Bob could close the channel by posting a notice that forces Alice to either post her unspent coins within some time period or forfeit the channel balance.

Despite the fact that Bob plays the role of the issuing "bank," no party is trusted in this setup. If Alice attempted to claim coins that she already paid to Bob, it would result in a double spend. Provided the e-cash scheme allows third parties (i.e. other observers of the bulletin board) to detect double spends, then Alice's claim will be denied. Thus Bob will always be paid what he is owed. If the scheme provides strong exculpability, then Bob (in his role as the "bank") cannot frame Alice for double spends. As a result, Bob cannot claim more of the escrowed funds than he has been paid even if he issues himself coins.

This scheme, however, is not succinct: Alice must post all of her unspent coins to close the channel. In order to realize the scalability promises of channels, we must somehow remove this requirement. Otherwise, the cost of this protocol, at least when transactions are disputed, is the same as just making n payments.

To get succinctness, Alice must be able to close her channel without posting every unspent coin. To accomplish that, Alice generates ciphertexts $c_0, \ldots, c_n$ one for each coin in a channel with

[6] It is of course possible to reverse the roles, but the asymmetry remains just in the other direction.

balance n. c_i contains the transaction for spending coin i and the encryption key for ciphertext c_{i+1}. Alice signs each of these and gives them to Bob. To close the channel and claim her $n - i$ renaming funds, she posts the key to ciphertext i, allowing Bob to decrypt all subsequent unspent coins. If any of decrypted coins are double spent, Bob can post the first spent transaction along with the decrypted one as evidence of double spending. He need only do this for one transaction to invalidate Alice's claim.

Bidirectional payment channels The above scheme inherits the limitation of classical e-cash: it can only make fixed value payments. As a consequence, it also cannot make payments from the merchant to the customer, i.e. the channel is unidirectional. Unfortunately, unlike our transformation from Zerocoin to Zerocash, we cannot merely include a hidden value in each coin to get variable value payments. The above approach to unidirectional payment channels works by, in essence, declaring the value of each transaction in advance and giving each one encrypted to Bob. We cannot meaningfully do that with variable valued payments since we would have to know all the values in advance.

To build variable valued payments, we need to allow interaction between the two parties. In essence, the customer and merchant will, during each payment, jointly agree and update the channel state. If, for example, Alice and Bob have a channel with \$10 owed to Alice and the remainder owed to Bob, updating the channel from \$10 to \$6 pays Bob \$4. The first challenge is that we need to update the channel state while hiding from the merchant which channel is being updated and what the balance is. These requirements can be met with zero-knowledge proofs over signatures and commitments: the customer proves that they have a signed balance in a commitment and then asks for a signature on a new committed balance which differs by δ from the payment amount.

The challenge with this approach is that we seemingly need to atomically 1) invalidate the old signed channel state; and 2) issue the new state. If we fail to invalidate the old channel state before signing the new one, Alice can spend out her channel but then use the still valid old channel state to claim back all of her money. On the other hand, if we invalidate the old channel state before

getting the new channel state, then Alice cannot claim any of her funds. However, it is not clear how to do that, and the most obvious approach, a fair exchange of signatures (i.e. one on the new channel state by the merchant and one invalidating the old channel state by the customer) is known to be impossible. [19]

Since we cannot atomically transition between the new and old channel states, we add an intermediate state that only allows for channel closure but not subsequent payments. First, Alice reveals an identifier for her current channel state to prevent her from replaying it. She then gets a token that allows her to close the channel with the new (but still hidden by a commitment) balance. Crucially, since this token cannot be used to make further payments, Alice has no opportunity for fraud but is assured she can close her channel. Now Alice can safely revoke her channel state by signing the revocation statement containing the old channel state identifier. When that is done, Bob can safely issue a new channel state.

The downside to this protocol is that it is not completely anonymous: if Bob is malicious he can force Alice to close the channel with a token linked to the current transaction. For this reason, we require that the channel itself be established anonymously via Zerocash.

Organization In the next three chapters, we present Zerocoin, Zerocash, and Bolt. Each chapter is drawn from the corresponding publication with little modification or additional content. Interested readers should refer to the full versions of these papers which may contain updates or corrections.

A note on definitions and proofs The definitions for both Zerocoin and Zerocash are game based. Although the definitions are different, they both take the same basic approach with one game ensuring anonymity and one ensuring balance, i.e. ensuring that a party cannot forge coins. This approach is, in fact, incomplete. Neither definition ensures that a party can spend the money they are paid.

In particular, it is possible in Zerocash for Alice to pay Bob two payments with the same serial number since they contain the same payer provided randomness and recipient address.

The security definitions do not prohibit such attacks. We would like to thank Zooko Wilcox in 2015 for the observation and attack details. This issue is mostly a definitional problem, as the actual attack is readily fixed.

The simple solution to the attack is for Bob to check for duplicate serial numbers. Another solution is to force payer provided randomness to be chosen deterministically as a function of the serial number of the coins being spent. This solution avoids the recipient needing to check for duplicate serial numbers.

Serial number attacks also apply to Zerocoin. An attacker, on observing a legitimate spend by a user of a coin with serial number x, can drop that transaction, insert their own coin with serial number x into the blockchain, and then after that coin is put in a block, spend it. This tactic blocks the spending of the original coin. The solution here is to make the serial number a public key and then require a signature under that key to spend. We would like to thank Tim Ruffing for independently pointing this out.

The choice of complete definitions is a challenging and open problem. One option, as shown in [20] is to use ideal functionalities and assume simulation soundness. One could go further and provide UC secure ideal functionalities [21], but that provision comes at the cost of succinctness.

The most promising approach is given in Appendix B of [22], which is similar to [20], but far easier to work with. It provides for an idealized ledger that is operated by a trusted party who maintains a version of the ledger in the clear and enforces consistency checks. At every step, the party extracts and enforces these checks against the in-clear ledger. This approach greatly reduces the complexity of extending the definitions. However, practical complete definitions are the subject of ongoing work. Therefore, we opt to retain the original definitions and proofs for Zerocoin and Zerocash here and simply note the modifications necessary to prevent these attacks.

Funding In addition to the various funding agencies and grants listed in each chapter that supported my research or my co-authors throughout my studies, the preparation of this thesis itself was supported

by the NSF under CNS 1228443.

Chapter 2

Zerocoin

This chapter is based on joint work with Christina Garman, Matthew Green, and Aviel Rubin titled "Zerocoin: Anonymous Distributed E-Cash from Bitcoin," appearing in the 2013 IEEE Symposium on Security and Privacy. [23]

2.1 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 [9] or to the summary of Barber *et al.* [24].

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.[1] 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.

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

This competition requires each node to solve a proof of work based on identifying specific SHA-256 preimages, specifically a block B such that $\mathsf{SHA256}(\mathsf{SHA256}(B)) < T$.[2] The value T 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 12.5 BTC.[3] 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 are simply a reference to a previous transaction output,[4] as well as a second script, $\mathsf{scriptSig}$, with code and data that when combined with $\mathsf{scriptPubKey}$ 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[5] of Bob's ECDSA public key pk_b, the amount d, and some script instructions in $\mathsf{scriptPubKey}$ as one output of a transaction whose referenced

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

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

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

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

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

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

Figure 2.1: 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.

inputs total at least d bitcoins (see Figure 2.1). 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.[6] 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 [9, 1, 26]. 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 [24, 27, 28]. 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] Individual recipients are free to disregard this advice. However, this could make them vulnerable to double-spending attacks as described by Karame *et al.* [25].

2.2 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 $\text{negl}(\cdot)$ represent a negligible function. We use $\mathcal{C}$ to indicate the set of allowable coin values.

Definition 2.2.1 (Decentralized E-Cash Scheme)**.** *A decentralized e-cash scheme consists of a tuple of possibly randomized algorithms* $(\mathsf{Setup}, \mathsf{Mint}, \mathsf{Spend}, \mathsf{Verify})$.

- $\mathsf{Setup}(1^\lambda) \to \mathsf{pp}$. *On input a security parameter, output a set of global public parameters* pp *and a description of the set* $\mathcal{C}$.
- $\mathsf{Mint}(\mathsf{pp}) \to (c, skc)$. *On input parameters* pp*, output a coin* $c \in \mathcal{C}$*, as well as a trapdoor* skc.
- $\mathsf{Spend}(\mathsf{pp}, c, skc, \mathsf{info}, \mathbf{C}) \to (\pi, S)$. *Given* pp*, a coin* c*, its trapdoor* skc*, some transaction string* $\mathsf{info} \in \{0,1\}^*$*, 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* $\perp$.
- $\mathsf{Verify}(\mathsf{pp}, \pi, S, \mathsf{info}, \mathbf{C}) \to \{0,1\}$. *Given* pp*, a proof* π*, a serial number* S*, transaction information* info*, and a set of coins* $\mathbf{C}$*, output* 1 *if* $\mathbf{C} \subseteq \mathcal{C}$ *and* (π, S, info) *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 info, 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 $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$ and $(c, skc) \leftarrow \mathsf{Mint}(\mathsf{pp})$. Let $\mathbf{C} \subseteq \mathcal{C}$ be any valid set of coins, where $|\mathbf{C}| \leq poly(\lambda)$, and assign $(\pi, S) \leftarrow \mathsf{Spend}(\mathsf{pp}, c, skc, \mathsf{info}, \mathbf{C})$. The scheme is *correct* if, over all $\mathbf{C}$, R, and random coins used in the above algorithms, the following equality holds with probability $1 - \mathrm{negl}(\lambda)$:

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

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 2.2.2 (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.*

$Anonymity(\Pi, \mathcal{A}, \lambda)$

$\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$

For $i \in \{0, 1\}$*:* $(c_i, skc_i) \leftarrow \mathsf{Mint}(\mathsf{pp})$

$(\mathbf{C}, \mathsf{info}, z) \leftarrow \mathcal{A}_1(\mathsf{pp}, c_0, c_1)$*;* $b \leftarrow \{0, 1\}$

$(\pi, S) \leftarrow \mathsf{Spend}(\mathsf{pp}, c_b, skc_b, \mathsf{info}, \mathbf{C} \cup \{c_0, c_1\})$

Output: $b' \leftarrow \mathcal{A}_2(z, \pi, S)$

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

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 info.

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.[7] 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 2.2.3 (Balance)**.** *A decentralized e-cash scheme* $\Pi = (\mathsf{Setup}, \mathsf{Mint}, \mathsf{Spend}, \mathsf{Verify})$ *satisfies the* Balance *property if* $\forall N \leq \text{poly}(\lambda)$ *every* p.p.t. *adversary* $\mathcal{A}$ *has negligible advantage in the following experiment.*

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

The oracle $\mathcal{O}_{spend}$ operates as follows: on the j_{th} query $\mathcal{O}_{spend}(c_j, \mathbf{C}_j, \mathsf{info}_j)$, the oracle outputs $\perp$ if $c_j \notin \{c_1, \ldots, c_N\}$. Otherwise it returns $(\pi_j, S_j) \leftarrow \mathsf{Spend}(\mathsf{pp}, c_j, skc_j, \mathsf{info}_j, \mathbf{C}_j)$ to $\mathcal{A}$ and records (S_j, info_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 $\mathfrak{s} = (\pi', S', \mathsf{info}', \mathbf{C}')$:

- $\mathsf{Verify}(\mathsf{pp}, \pi', S', \mathsf{info}', \mathbf{C}') = 1$.

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

- $\mathbf{C}' \subseteq \{c_1, \ldots, c_N, c'_1, \ldots, c'_m\}$.
- $(S', \mathsf{info}') \notin \mathcal{T}$.
- 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.

2.3 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.

2.3.1 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 [17], with extensions due to e.g., [29, 30, 31, 13]. We convert these into *non-interactive* proofs by applying the Fiat-Shamir heuristic [32]. In the latter case, we refer to the resulting non-interactive proofs as *signatures of knowledge* as defined in [33].

When referring to these proofs we will use the notation of Camenisch and Stadler [34]. 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 [16] and later improved by Baric and Pfitzmann [35] and Camenisch and Lysyanskaya [36]. We describe the accumulator using the following algorithms:

- $\mathsf{AccumSetup}(\lambda) \rightarrow \mathsf{pp}$. On input a security parameter, sample primes $\hat{p}, \hat{q}$ (with polynomial dependence on the security parameter), compute $N = \hat{p}\hat{q}$, and sample a seed value $u \in QR_N, u \neq 1$. Output (N, u) as pp.

- $\mathsf{Accumulate}(\mathsf{pp}, \mathbf{C}) \rightarrow A$. On input pp (N, u) and a set of prime numbers $\mathbf{C} = \{c_1, \ldots, c_i \mid c \in [A, B]\}$,[8] compute the accumulator A as $u^{c_1 c_2 \cdots c_n} \ mod \ N$.

- $\mathsf{GenWitness}(\mathsf{pp}, v, \mathbf{C}) \rightarrow w$. On input pp (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., $w = \mathsf{Accumulate}(\mathsf{pp}, \mathbf{C} \setminus \{v\})$.

- $\mathsf{AccVerify}(\mathsf{pp}, A, v, \omega) \rightarrow \{0, 1\}$. On input pp (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 $v \in [A, B]$ as defined previously.

For simplicity, the description above uses the full calculation of A. Camenisch and Lysyanskaya [36] 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 [36] 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 $\mathsf{AccVerify}$ 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:

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

[8] Specifically, the values A, B must be chosen such that $2 < A < B < A^2$ as described in [36].

2.3.2 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:

- $\mathsf{Setup}(1^\lambda) \to \mathsf{pp}$. On input a security parameter, run $\mathsf{AccumSetup}(1^\lambda)$ to obtain the values (N, u). Next generate primes p, q such that $p = 2wq + 1$ 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 $\mathsf{pp} = (N, u, p, q, g, h)$.

- $\mathsf{Mint}(\mathsf{pp}) \to (c, skc)$. Select $S, r \leftarrow \mathbb{Z}_q^*$ and compute $c \leftarrow g^S h^r \bmod p$ such that $\{c \text{ prime} \mid c \in [A, B]\}$. Set $skc = (S, r)$ and output (c, skc).

- $\mathsf{Spend}(\mathsf{pp}, c, skc, \mathsf{info}, \mathbf{C}) \to (\pi, S)$. 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:[9]

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

- $\mathsf{Verify}(\mathsf{pp}, \pi, S, \mathsf{info}, \mathbf{C}) \to \{0, 1\}$. 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 info 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 $(\hat{p}, \hat{q})$ is not used subsequent to the Setup procedure and can therefore be destroyed immediately after the parameters are generated. Alternatively, implementers can use

[9] See Appendix A.2 for the construction of the ZKSoK.

the technique of Sander for generating so-called RSA UFOs for accumulator parameters without a trapdoor [37].

2.3.3 Security Analysis

We now consider the security of our construction.

Theorem 2.3.1. *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 2.3.1 in Appendix A.1. 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 2.3.2. *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 2.3.1 is included in Appendix A.1. 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).

2.4 Real World Security and Parameter Choice

2.4.1 Anonymity of Zerocoin

Definition 2.2.2 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.

2.4.2 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.[10] 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 2.8.

Accumulator RSA key. Because generating a new accumulator requires either a new trusted setup phase or generating a new RSA UFO [37], we cannot re-key very frequently. As a result, the

[10]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.

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.

2.5 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 $\mathsf{Mint}(\mathsf{pp}) \rightarrow (c, skc)$ and stores skc securely.[11] 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 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

[11] 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.

$\mathsf{Spend}(\mathsf{pp}, c, skc, hash(ptx), \mathbf{C}) \to (\pi, S)$. 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 $\mathsf{Verify}(\mathsf{pp}, \pi, S, hash(ptx), \mathbf{C}) = 1$ 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 2.3 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.[12] 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

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

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.[13]

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.

[13] 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.1 for details.

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 Spend(...) 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 are between 1 and 2 KB and (2) it may increase memory pressure on clients that store transactions in memory.[14]

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.[15]

2.5.1 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

[14] 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.

[15] 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.

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.

2.5.2 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 [38]. 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.

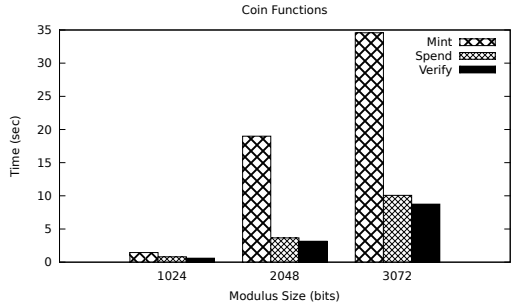
2.5.3 Code Changes

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 [39] 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.

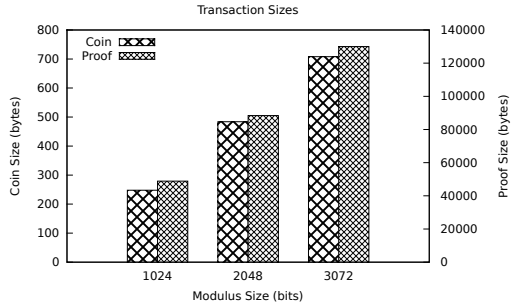
2.5.4 Incremental Deployment

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 [40], it is not clear how willing the Bitcoin community is to repeat it. As such, we consider the possibility of an incremental deployment.

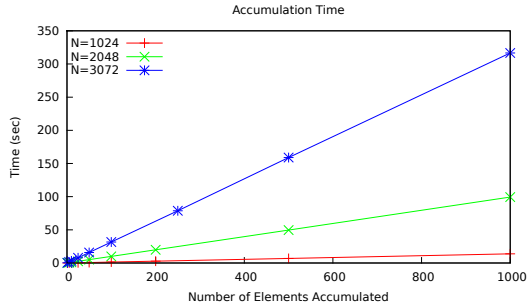
One way to accomplish this is to embed the above protocol as comments in standard Bitcoin scripts. For non Zerocoin aware nodes, this data is effectively inert, and we can use Bitcoin's n of k signature support to specify that such comment embedded zerocoins are valid only if signed by some subset of 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.



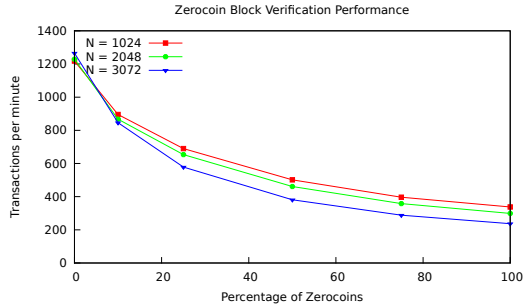
(a) Times for a single Zerocoin operation measured in seconds. These operations do not include the time required to compute the accumulator.



(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 2.2: Zerocoin performance as a function of parameter size.

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 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.

2.6 Performance

To validate our results, we conducted several experiments using the modified `bitcoind` implementation described in Section 2.5. We ran our experiments with three different parameter sizes, where each corresponds to a length of the RSA modulus N: 1024 bits, 2048 bits, and 3072 bits.[16]

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.

2.6.1 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(...) or for the global accumulator in both Spend(...) and Verify(...). 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

[16] 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.

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 $\mathsf{Spend}(\ldots)$ and $\mathsf{Verify}(\ldots)$ 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 2.2a shows the measured times for computing the coin operations, Figure 2.2b shows the resulting proof sizes for each security parameter, and Figure 2.2c 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.

2.6.2 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.[17] If verification takes longer than 10 minutes for blocks with a reasonable number of zerocoins, then the network cannot function.[18] 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).

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

[18] 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.

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 2.2d.

2.6.3 Discussion

Our results show that Zerocoin scales beyond Bitcoin transaction volumes as of 2013. 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 four times the number of transactions in a Bitcoin block as of 2013 — verification takes less than five minutes. This is under the unreasonable assumption that all Bitcoin transactions are supplanted by Zerocoin transactions.[19] In fact, we can scale well beyond Bitcoin's average of between 200 and 400 transactions per block [41] 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).

[19] 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.

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 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 ASICs for mining, and as such the latter is not an issue. The former, however, remains an unknown. At the very least it seems unlikely to disproportionately affect Zerocoin performance.

2.7 Previous Work

2.7.1 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 [11] based on RSA signatures, later extending this work to the offline setting [12] 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 [42, 43] and reducing wallet size [14]. One exception to the rule above comes from Sander and Ta-Shma [44] 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 [45]. 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 [24].

2.7.2 Anonymity

Numerous works have shown that "pseudonymized" graphs can be re-identified even under passive analysis. Narayanan and Shmatikov [46] showed that real world social networks can be passively de-anonymized. Similarly, Backstrom *et al.* [47] 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 [48].

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 [1]. Ron and Shamir examined the general structure of the Bitcoin network graph [49] after its nearly 3-fold expansion. Finally, we have been made privately aware of two other early-stage efforts to examine Bitcoin anonymity.

2.8 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 [50, 51]. 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 2.4.2, 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,[20] 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 [52]) 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.

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

Chapter 3

Zerocash

This chaper is based of joint work with Eli Ben-Sasson , Alessandro Chiesa , Christina Garman, Matthew Green, Eran Tromer, and Madars Virza titled "Zerocash: Decentralized Anonymous Payments from Bitcoin," appearing in the 2014 IEEE Symposium on Security and Privacy. [23]

3.1 Introduction

(1) We introduce the notion of a *decentralized anonymous payment scheme*, which formally captures the functionality and security guarantees of a full-fledged decentralized electronic currency with strong anonymity guarantees. We provide a construction of this primitive and prove its security under specific cryptographic assumptions. The construction leverages recent advances in the area of zero-knowledge proofs. Specifically, it uses *zero-knowledge Succinct Non-interactive ARguments of Knowledge* (zk-SNARKs) [53, 54, 55, 56, 57, 18, 58, 59].

(2) We implement the above primitive, via a system that we call **Zerocash**. Our system (at 128 bits of security):

- reduces the size of transactions spending a coin to under 1 kB (an improvement of over 97.7%);
- reduces the spend-transaction verification time to under 6 ms (an improvement of over 98.6%);

- allows for anonymous transactions of variable amounts;
- hides transaction amounts and the values of coins held by users; and
- allows for payments to be made directly to a user's fixed address (without user interaction).

To validate our system, we measured its performance and established feasibility by conducting experiments in a test network of 1000 nodes (approximately $\frac{1}{16}$ of the unique IPs in the Bitcoin network and $\frac{1}{3}$ of the nodes reachable at any given time [60]). This inspires confidence that Zerocash can be deployed as a fork of Bitcoin and operate at the same scale. Thus, due to its substantially improved functionality and performance, Zerocash makes it possible to entirely replace traditional Bitcoin payments with anonymous alternatives.

Concurrent work. The idea of using zk-SNARKs in the Bitcoin setting was first presented by one of the authors at Bitcoin 2013 [61]. In concurrent work, Danezis et al. [62] suggest using zk-SNARKs to reduce proof size and verification time in Zerocoin; see Section 3.9 for a comparison.

3.1.1 zk-SNARKs

A zk-SNARK is an efficient variant of a *zero-knowledge proof of knowledge* [63], which we first informally describe via an example. Suppose Alice wishes to prove to Bob the statement *"I (Alice) own 30 bitcoins"*. A simple method for Alice to do so is to point to 30 coins on the block chain and, for each of them, sign a message ("hello, world") using the secret key that controls that coin. Alas, this method *leaks knowledge* to Bob, by identifying which coins are Alice's. A zero-knowledge proof of knowledge allows Alice to achieve the same goal, while revealing *no information* to Bob (beyond the fact that she *knows* some secret keys that control 30 coins). Crucially, such proofs can be obtained for any statement that can be verified to be true by use of an efficient computation involving auxiliary inputs such as trapdoors and passwords (such statements are called "NP statements").

We now sketch in more technical terms the definition of a zk-SNARK; see Section 3.2 for more details. A zk-SNARK is a non-interactive zero-knowledge proof of knowledge that is *succinct*, i.e., for which proofs are very short and easy to verify. More precisely, let $\mathcal{L}$ be an NP language, and

let C be a nondeterministic decision circuit for $\mathcal{L}$ on a given instance size n. A zk-SNARK can be used to prove and verify membership in $\mathcal{L}$, for instances of size n, as follows. After taking C as input, a trusted party conducts a one-time setup phase that results in two public keys: a proving key pk and a verification key vk. The proving key pk enables any (untrusted) prover to produce a proof π attesting to the fact that $x \in \mathcal{L}$, for an instance x (of size n) of his choice. The non-interactive proof π is *zero knowledge* and a *proof of knowledge*. Anyone can use the verification key vk to verify the proof π; in particular zk-SNARK proofs are publicly verifiable: anyone can verify π, without ever having to interact with the prover who generated π. Succinctness requires that (for a given security level) π has *constant size* and can be verified in time that is linear in $|x|$ (rather than linear in $|C|$).

3.1.2 Centralized anonymous payment systems

Before describing our new decentralized payment system, we put it in context by recalling two pre-Bitcoin payment schemes, both of which relied on a *bank*, acting as a central trusted party.

Anonymous e-cash. Chaum [11] first obtained anonymous e-cash. In Chaum's scheme, the minting of a coin involves both a user, Alice, and the bank: to mint a coin of a given value v, Alice first selects a random secret serial number sn (unknown to the bank); then, the bank, after deducting v from Alice's balance, signs sn via a *blind signature*. Afterwards, if Alice wants to transfer her coin to Bob, she reveals sn to him and proves that sn was signed by the bank; during this transfer, Bob (or the bank) cannot deduce Alice's identity from the revealed information. Double-spending is prevented because the bank will not honor a coin with a previously-seen serial number.

Unforgeable e-cash. One problem with Chaum's scheme is that coins can be forged if the bank's secret key is compromised. Sander and Ta-Shma [44] addressed this, as follows. The bank maintains a public Merkle tree of "coin commitments", and users periodically retrieve its root rt; in particular, the bank maintains no secrets. When Alice requests a coin (of unit value), she picks a random serial number sn and auxiliary string r, and then sends $\mathsf{cm} := \mathsf{CRH}(\mathsf{sn}\|r)$ to the bank, where CRH is a collision-resistant hash; the bank deducts the appropriate amount from Alice's balance and then

records cm as a leaf in the Merkle tree. Afterwards, to pay Bob, Alice sends him sn along with a zero-knowledge proof of knowledge π of the following NP statement: *"there exists r such that $\mathsf{CRH}(\mathsf{sn}\|r)$ is a leaf in a Merkle tree with root rt"*. In other words, Alice can convince Bob that sn is the serial number contained in *some* coin commitment in the Merkle tree; but the zero-knowledge property prevents Bob from learning information about which coin commitment is Alice's, thereby protecting Alice's identity. Later, Bob can "cash out" Alice's coin by showing sn and π to the bank.[1]

Moving to a fungible anonymous decentralized system. In this paper, like [44], we hash a coin's serial number and use Merkle trees to compactly represent the set of minted coins. Unlike [44], we also ensure the privacy of a coin's value and support transactions that split and merge coins, thus achieving (and implementing) a new kind of fully-fungible and divisible payment scheme. As in Bitcoin (and in stark contrast to previous e-cash schemes), we do not rely on a trusted bank. Therefore, we require a new set of definitions and protocols, designed to protect Alice's anonymity while preventing her from falsely increasing her balance under the veil of her boosted privacy. An informal description of our payment scheme follows.

3.1.3 Decentralized anonymous payment schemes

We construct a *decentralized anonymous payment (DAP) scheme*, which is a decentralized e-cash scheme that allows direct anonymous payments of any amount. See Section 3.3 for a formal definition. Here, we outline our construction in six incremental steps; the construction details are in Section 3.4.

Our construction functions on top of any ledger-based base currency, such as Bitcoin. At any given time, a unique valid snapshot of the currency's *ledger* is available to all users. The ledger is a sequence of *transactions* and is append-only. Transactions include both the underlying currency's transactions, as well as new transactions introduced by our construction. For concreteness, we focus the discussion below on Bitcoin (though later definitions and constructions are stated abstractly).

[1] We omit details about how the bank can identify Alice in the event that she double spends her coin.

Step 1: user anonymity with fixed-value coins. We first describe a simplified construction, in which all coins have the same value of, e.g., 1 BTC. This construction, similar to the Zerocoin protocol, shows how to hide a payment's origin. In terms of tools, we make use of zk-SNARKs (recalled above) and a commitment scheme. Let COMM denote a statistically-hiding non-interactive commitment scheme (i.e., given randomness r and message m, the commitment is $c := \mathsf{COMM}_r(m)$; subsequently, c is opened by revealing r and m, and one can verify that $\mathsf{COMM}_r(m)$ equals c).

In the simplified construction, a new coin $\mathbf{c}$ is minted as follows: a user u samples a random *serial number* sn and a *trapdoor* r, computes a *coin commitment* $\mathsf{cm} := \mathsf{COMM}_r(\mathsf{sn})$, and sets $\mathbf{c} := (r, \mathsf{sn}, \mathsf{cm})$. A corresponding mint transaction $\mathsf{tx}_{\mathsf{Mint}}$, containing cm (but not sn or r), is sent to the ledger; $\mathsf{tx}_{\mathsf{Mint}}$ is appended to the ledger only if u has paid 1 BTC to a backing escrow pool (e.g., the 1 BTC may be paid via plaintext information encoded in $\mathsf{tx}_{\mathsf{Mint}}$). Mint transactions are thus certificates of deposit, deriving their value from the backing pool.

Subsequently, letting CMList denote the list of all coin commitments on the ledger, u may spend $\mathbf{c}$ by posting a spend transaction $\mathsf{tx}_{\mathsf{Spend}}$ that contains (i) the coin's serial number sn; and (ii) a zk-SNARK proof π of the NP statement *"I know r such that $\mathsf{COMM}_r(\mathsf{sn})$ appears in the list CMList of coin commitments"*. Assuming that sn does not already appear on the ledger (as part of a past spend transaction), u can redeem the deposited amount of 1 BTC, which u can either keep, transfer to someone else, or mint a new coin. (If sn does already appear on the ledger, this is considered double spending, and the transaction is discarded.)

User anonymity is achieved because the proof π is zero-knowledge: while sn is revealed, no information about r is, and finding which of the numerous commitments in CMList corresponds to a particular spend transaction $\mathsf{tx}_{\mathsf{Spend}}$ is equivalent to inverting $f(x) := \mathsf{COMM}_x(\mathsf{sn})$, which is assumed to be infeasible. Thus, the origin of the payment is anonymous.

Step 2: compressing the list of coin commitments. In the above NP statement, CMList is specified explicitly as a list of coin commitments. This naive representation severely limits scalability because the time and space complexity of most protocol algorithms (e.g., the proof verification

algorithm) grow linearly with CMList. Moreover, coin commitments corresponding to already-spent coins cannot be dropped from CMList to reduce costs, since they cannot be identified (due to the same zero-knowledge property that provides anonymity).

As in [44], we rely on a collision-resistant function CRH to avoid an explicit representation of CMList. We maintain an efficiently-updatable append-only CRH-based Merkle tree $\mathsf{Tree}(\mathsf{CMList})$ over the (growing) list CMList and let rt denote the root of $\mathsf{Tree}(\mathsf{CMList})$. It is well-known that rt can be updated to account for the insertion of new leaves with time and space proportional to just the tree depth. Hence, the time and space complexity is reduced from linear in the size of CMList to logarithmic. With this in mind, we modify the NP statement to the following one: *"I know r such that $\mathsf{COMM}_r(\mathsf{sn})$ appears as a leaf in a CRH-based Merkle tree whose root is rt"*. Compared with the naive data structure for CMList, this modification increases exponentially the size of CMList that a given zk-SNARK implementation can support. (Concretely: using Merkle trees of depth 64, Zerocash supports 2^{64} coins.)

Step 3: extending coins for direct anonymous payments. So far, the coin commitment cm of a coin $\mathbf{c}$ is a commitment to the coin's serial number sn. However, this creates a problem when transferring $\mathbf{c}$ to another user. Indeed, suppose that a user u_A created $\mathbf{c}$, and u_A sends $\mathbf{c}$ to another user u_B. First, since u_A knows sn, the spending of $\mathbf{c}$ by u_B is both non-anonymous (since u_A sees when $\mathbf{c}$ is spent, by recognizing sn) and risky (since u_A could still spend $\mathbf{c}$ first). Thus, u_B must immediately spend $\mathbf{c}$ and mint a new coin $\mathbf{c}'$ to protect himself. Second, if u_A in fact wants to transfer to u_B, e.g., 100 BTC, then doing so is both unwieldy (since it requires 100 transfers) and non-anonymous (since the amount of the transfer is leaked). And third, transfers in amounts that are not multiples of 1 BTC (the fixed value of a coin) are not supported. Thus, the simplified construction described is inadequate as a payment scheme.

We address this by modifying the derivation of a coin commitment, and using pseudorandom functions to target payments and to derive serial numbers, as follows. We use three pseudorandom functions (derived from a single one). For a seed x, these are denoted $\mathsf{PRF}^{\mathrm{addr}}_x(\cdot)$, $\mathsf{PRF}^{\mathrm{sn}}_x(\cdot)$, and

$\mathsf{PRF}^{\mathrm{pk}}_x(\cdot)$. We assume that $\mathsf{PRF}^{\mathrm{sn}}$ is moreover collision-resistant.

To provide targets for payments, we use *addresses*: each user u generates an address key pair $(a_{\mathsf{pk}}, a_{\mathsf{sk}})$, the *address public key* and *address private key* respectively. The coins of u contain the value a_{pk} and can be spent only with knowledge of a_{sk}. A key pair $(a_{\mathsf{pk}}, a_{\mathsf{sk}})$ is sampled by selecting a random seed a_{sk} and setting $a_{\mathsf{pk}} := \mathsf{PRF}^{\mathrm{addr}}_{a_{\mathsf{sk}}}(0)$. A user can generate and use any number of address key pairs.

Next, we redesign minting to allow for greater functionality. To mint a coin $\mathbf{c}$ of a desired value v, the user u first samples ρ, which is a secret value that determines the coin's serial number as $\mathsf{sn} := \mathsf{PRF}^{\mathrm{sn}}_{a_{\mathsf{sk}}}(\rho)$. Then, u commits to the tuple $(a_{\mathsf{pk}}, v, \rho)$ in two phases: (a) u computes $k := \mathsf{COMM}_r(a_{\mathsf{pk}} \| \rho)$ for a random r; and then (b) u computes $\mathsf{cm} := \mathsf{COMM}_s(v \| k)$ for a random s. The minting results in a coin $\mathbf{c} := (a_{\mathsf{pk}}, v, \rho, r, s, \mathsf{cm})$ and a mint transaction $\mathsf{tx}_{\mathsf{Mint}} := (v, k, s, \mathsf{cm})$. Crucially, due to the nested commitment, anyone can verify that cm in $\mathsf{tx}_{\mathsf{Mint}}$ is a coin commitment of a coin of value v (by checking that $\mathsf{COMM}_s(v\|k)$ equals cm) but cannot discern the owner (by learning the address key a_{pk}) or serial number (derived from ρ) because these are hidden in k. As before, $\mathsf{tx}_{\mathsf{Mint}}$ is accepted by the ledger only if u deposits the correct amount, in this case v BTC.

Coins are spent using the *pour* operation, which takes a set of input coins, to be consumed, and "pours" their value into a set of fresh output coins — such that the total value of output coins equals the total value of the input coins. Suppose that u, with address key pair $(a^{\mathsf{old}}_{\mathsf{pk}}, a^{\mathsf{old}}_{\mathsf{sk}})$, wishes to consume his coin $\mathbf{c}^{\mathsf{old}} = (a^{\mathsf{old}}_{\mathsf{pk}}, v^{\mathsf{old}}, \rho^{\mathsf{old}}, r^{\mathsf{old}}, s^{\mathsf{old}}, \mathsf{cm}^{\mathsf{old}})$ and produce two new coins $\mathbf{c}^{\mathsf{new}}_1$ and $\mathbf{c}^{\mathsf{new}}_2$, with total value $v^{\mathsf{new}}_1 + v^{\mathsf{new}}_2 = v^{\mathsf{old}}$, respectively targeted at address public keys $a^{\mathsf{new}}_{\mathsf{pk},1}$ and $a^{\mathsf{new}}_{\mathsf{pk},2}$. (The addresses $a^{\mathsf{new}}_{\mathsf{pk},1}$ and $a^{\mathsf{new}}_{\mathsf{pk},2}$ may belong to u or to some other user.) The user u, for each $i \in \{1, 2\}$, proceeds as follows: (i) u samples serial number randomness ρ^{new}_i; (ii) u computes $k^{\mathsf{new}}_i := \mathsf{COMM}_{r^{\mathsf{new}}_i}(a^{\mathsf{new}}_{\mathsf{pk},i} \| \rho^{\mathsf{new}}_i)$ for a random r^{new}_i; and (iii) u computes $\mathsf{cm}^{\mathsf{new}}_i := \mathsf{COMM}_{s^{\mathsf{new}}_i}(v^{\mathsf{new}}_i \| k^{\mathsf{new}}_i)$ for a random s^{new}_i.

This yields the coins $\mathbf{c}^{\mathsf{new}}_1 := (a^{\mathsf{new}}_{\mathsf{pk},1}, v^{\mathsf{new}}_1, \rho^{\mathsf{new}}_1, r^{\mathsf{new}}_1, s^{\mathsf{new}}_1, \mathsf{cm}^{\mathsf{new}}_1)$ and $\mathbf{c}^{\mathsf{new}}_2 := (a^{\mathsf{new}}_{\mathsf{pk},2}, v^{\mathsf{new}}_2, \rho^{\mathsf{new}}_2, r^{\mathsf{new}}_2, s^{\mathsf{new}}_2, \mathsf{cm}^{\mathsf{new}}_2)$. Next, u produces a zk-SNARK proof $\pi_{\mathtt{POUR}}$ for the following NP statement,

which we call POUR:

"Given the Merkle-tree root rt*, serial number* $\mathsf{sn}^{\mathsf{old}}$*, and coin commitments* $\mathsf{cm}_1^{\mathsf{new}}, \mathsf{cm}_2^{\mathsf{new}}$*, I know coins* $\mathbf{c}^{\mathsf{old}}, \mathbf{c}_1^{\mathsf{new}}, \mathbf{c}_2^{\mathsf{new}}$*, and address secret key* $a_{\mathsf{sk}}^{\mathsf{old}}$ *such that:*

- *The coins are well-formed: for* $\mathbf{c}^{\mathsf{old}}$ *it holds that* $k^{\mathsf{old}} = \mathsf{COMM}_{r^{\mathsf{old}}}(a_{\mathsf{pk}}^{\mathsf{old}} \| \rho^{\mathsf{old}})$ *and* $\mathsf{cm}^{\mathsf{old}} = \mathsf{COMM}_{s^{\mathsf{old}}}(v^{\mathsf{old}} \| k^{\mathsf{old}})$*; and similarly for* $\mathbf{c}_1^{\mathsf{new}}$ *and* $\mathbf{c}_2^{\mathsf{new}}$*.*
- *The address secret key matches the public key:* $a_{\mathsf{pk}}^{\mathsf{old}} = \mathsf{PRF}_{a_{\mathsf{sk}}^{\mathsf{old}}}^{\mathrm{addr}}(0)$*.*
- *The serial number is computed correctly:* $\mathsf{sn}^{\mathsf{old}} := \mathsf{PRF}_{a_{\mathsf{sk}}^{\mathsf{old}}}^{\mathrm{sn}}(\rho^{\mathsf{old}})$*.*
- *The coin commitment* $\mathsf{cm}^{\mathsf{old}}$ *appears as a leaf of a Merkle-tree with root* rt*.*
- *The values add up:* $v_1^{\mathsf{new}} + v_2^{\mathsf{new}} = v^{\mathsf{old}}$*."*

A resulting pour transaction $\mathsf{tx}_{\mathsf{Pour}} := (\mathsf{rt}, \mathsf{sn}^{\mathsf{old}}, \mathsf{cm}_1^{\mathsf{new}}, \mathsf{cm}_2^{\mathsf{new}}, \pi_{\mathtt{POUR}})$ is appended to the ledger. (As before, the transaction is rejected if the serial number sn appears in a previous transaction.)

Now suppose that u does not know, say, the address secret key $a_{\mathsf{sk},1}^{\mathsf{new}}$ that is associated with the public key $a_{\mathsf{pk},1}^{\mathsf{new}}$. Then, u cannot spend $\mathbf{c}_1^{\mathsf{new}}$ because he cannot provide $a_{\mathsf{sk},1}^{\mathsf{new}}$ as part of the witness of a subsequent pour operation. Furthermore, when a user who knows $a_{\mathsf{sk},1}^{\mathsf{new}}$ does spend $\mathbf{c}_1^{\mathsf{new}}$, the user u cannot track it, because he knows no information about its revealed serial number, which is $\mathsf{sn}_1^{\mathsf{new}} := \mathsf{PRF}_{a_{\mathsf{sk},1}^{\mathsf{new}}}^{\mathrm{sn}}(\rho_1^{\mathsf{new}})$.

Also observe that $\mathsf{tx}_{\mathsf{Pour}}$ reveals no information about how the value of the consumed coin was divided among the two new fresh coins, nor which coin commitment corresponds to the consumed coin, nor the address public keys to which the two new fresh coins are targeted. The payment was conducted in full anonymity.

More generally, a user may pour $N^{\mathsf{old}} \geq 0$ coins into $N^{\mathsf{new}} \geq 0$ coins. For simplicity we consider the case $N^{\mathsf{old}} = N^{\mathsf{new}} = 2$, without loss of generality. Indeed, for $N^{\mathsf{old}} < 2$, the user can mint a coin with value 0 and then provide it as a "null" input, and for $N^{\mathsf{new}} < 2$, the user can create (and discard) a new coin with value 0. For $N^{\mathsf{old}} > 2$ or $N^{\mathsf{new}} > 2$, the user can compose $\log N^{\mathsf{old}} + \log N^{\mathsf{new}}$ of the 2-input/2-output pours.

Step 4: sending coins. Suppose that $a_{\mathsf{pk},1}^{\mathsf{new}}$ is the address public key of u_1. In order to allow u_1

to actually spend the new coin $\mathbf{c}_1^{\mathsf{new}}$ produced above, u must somehow send the secret values in $\mathbf{c}_1^{\mathsf{new}}$ to u_1. One way is for u to send u_1 a private message, but the requisite private communication channel necessitates additional infrastructure or assumptions. We avoid this "out-of-band" channel and instead build this capability directly into our construction by leveraging the ledger as follows.

We modify the structure of an address key pair. Each user now has a key pair $(\mathsf{addr}_{\mathsf{pk}}, \mathsf{addr}_{\mathsf{sk}})$, where $\mathsf{addr}_{\mathsf{pk}} = (a_{\mathsf{pk}}, \mathsf{pk}_{\mathsf{enc}})$ and $\mathsf{addr}_{\mathsf{sk}} = (a_{\mathsf{sk}}, \mathsf{sk}_{\mathsf{enc}})$. The values $(a_{\mathsf{pk}}, a_{\mathsf{sk}})$ are generated as before. In addition, $(\mathsf{pk}_{\mathsf{enc}}, \mathsf{sk}_{\mathsf{enc}})$ is a key pair for a *key-private encryption scheme* [64].

Then, u computes the ciphertext $\mathbf{C}_1$ that is the encryption of the plaintext $(v_1^{\mathsf{new}}, \rho_1^{\mathsf{new}}, r_1^{\mathsf{new}}, s_1^{\mathsf{new}})$, under $\mathsf{pk}_{\mathsf{enc},1}^{\mathsf{new}}$ (which is part of u_1's address public key $\mathsf{addr}_{\mathsf{sk},1}^{\mathsf{new}}$), and includes $\mathbf{C}_1$ in the pour transaction $\mathsf{tx}_{\mathsf{Pour}}$. The user u_1 can then find and decrypt this message (using his $\mathsf{sk}_{\mathsf{enc},1}^{\mathsf{new}}$) by scanning the pour transactions on the public ledger. Again, note that adding $\mathbf{C}_1$ to $\mathsf{tx}_{\mathsf{Pour}}$ leaks neither paid amounts, nor target addresses due to the key-private property of the encryption scheme. (The user u does the same with $\mathbf{c}_2^{\mathsf{new}}$ and includes a corresponding ciphertext $\mathbf{C}_2$ in $\mathsf{tx}_{\mathsf{Pour}}$.)

Step 5: public outputs. The construction so far allows users to mint, merge, and split coins. But how can a user redeem one of his coins, i.e., convert it back to the base currency (Bitcoin)? For this, we modify the pour operation to include a *public output*. When spending a coin, the user u also specifies a nonnegative v_{pub} and a *transaction string* $\mathsf{info} \in \{0,1\}^*$. The balance equation in the NP statement $\mathtt{POUR}$ is changed accordingly: "$v_1^{\mathsf{new}} + v_2^{\mathsf{new}} + v_{\mathsf{pub}} = v^{\mathsf{old}}$". Thus, of the input value v^{old}, a part v_{pub} is publicly declared, and its target is specified, somehow, by the string info. The string info can be used to specify the destination of these redeemed funds (e.g., a Bitcoin wallet public key).[2] Both v_{pub} and info are now included in the resulting pour transaction $\mathsf{tx}_{\mathsf{Pour}}$. (The public output is optional, as the user u can set $v_{\mathsf{pub}} = 0$.)

Step 6: non-malleability. To prevent malleability attacks on a pour transaction $\mathsf{tx}_{\mathsf{Pour}}$ (e.g., embezzlement by re-targeting the public output of the pour by modifying info), we further modify the NP statement $\mathtt{POUR}$ and use digital signatures. Specifically, during the pour operation, the user u

[2] These public outputs can be considered as an "input" to a Bitcoin-style transaction, where the string info contains the Bitcoin output scripts. This mechanism also allows us to support Bitcoin's public transaction fees.

(i) samples a key pair $(\mathsf{pk}_{\mathsf{sig}}, \mathsf{sk}_{\mathsf{sig}})$ for a one-time signature scheme; (ii) computes $h_{\mathsf{Sig}} := \mathsf{CRH}(\mathsf{pk}_{\mathsf{sig}})$; (iii) computes the two values $h_1 := \mathsf{PRF}^{\mathrm{pk}}_{a^{\mathsf{old}}_{\mathsf{sk},1}}(h_{\mathsf{Sig}})$ and $h_2 := \mathsf{PRF}^{\mathrm{pk}}_{a^{\mathsf{old}}_{\mathsf{sk},2}}(h_{\mathsf{Sig}})$, which act as MACs to "tie" h_{Sig} to both address secret keys; (iv) modifies POUR to include the three values $h_{\mathsf{Sig}}, h_1, h_2$ and prove that the latter two are computed correctly; and (v) uses $\mathsf{sk}_{\mathsf{sig}}$ to sign every value associated with the pour operation, thus obtaining a signature σ, which is included, along with $\mathsf{pk}_{\mathsf{sig}}$, in $\mathsf{tx}_{\mathsf{Pour}}$. Since the $a^{\mathsf{old}}_{\mathsf{sk},i}$ are secret, and with high probability h_{Sig} changes for each pour transaction, the values h_1, h_2 are unpredictable. Moreover, the signature on the NP statement (and other values) binds all of these together, as argued in more detail in Appendix B.2 and Appendix B.3.

This ends the outline of the construction, which is summarized in part in Figure 3.1. We conclude by noting that, due to the zk-SNARK, our construction requires a one-time trusted setup of public parameters. The soundness of the proofs depends on this trust, though anonymity continues to hold even if the setup is corrupted by a malicious party.

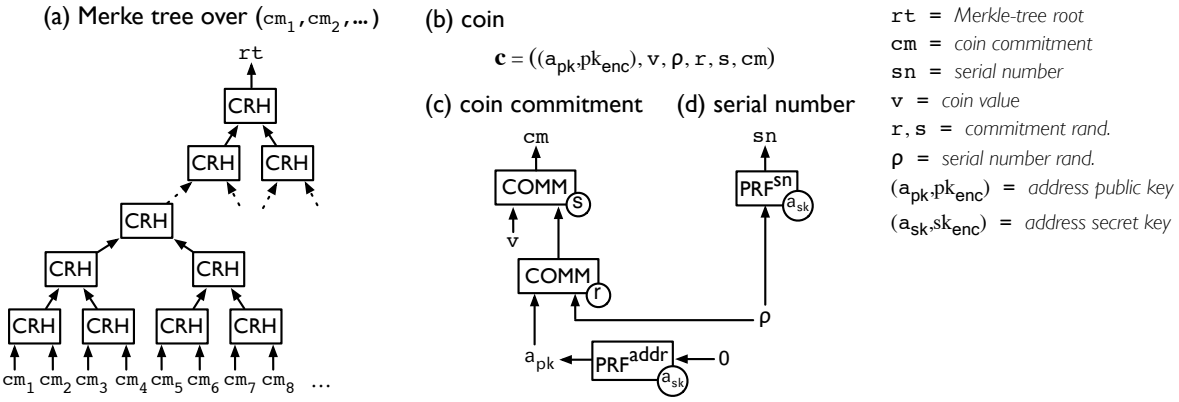


Figure 3.1: **(a)** Illustration of the CRH-based Merkle tree over the list CMList of coin commitments. **(b)** A coin **c**. **(c)** Illustration of the structure of a coin commitment cm. **(d)** Illustration of the structure of a coin serial number sn.

3.1.4 Zerocash

We outline Zerocash, a concrete implementation, at 128 bits of security, of our DAP scheme construction; see Section 3.5 for details. Zerocash entails carefully instantiating the cryptographic

ingredients of the construction to ensure that the zk-SNARK, the "heaviest" component, is efficient enough in practice. In the construction, the zk-SNARK is used to prove/verify a specific NP statement: POUR. While zk-SNARKs are asymptotically efficient, their concrete efficiency depends on the arithmetic circuit C that is used to decide the NP statement. Thus, we seek instantiations for which we can design a relatively small arithmetic circuit $C_{\mathtt{POUR}}$ for verifying the NP statement POUR.

Our approach is to instantiate all of the necessary cryptographic ingredients (commitment schemes, pseudorandom functions, and collision-resistant hashing) based on SHA256. We first design a hand-optimized circuit for verifying SHA256 computations (or, more precisely, its compression function, which suffices for our purposes).[3] Then, we use this circuit to construct $C_{\mathtt{POUR}}$, which verifies all the necessary checks for satisfying the NP statement $C_{\mathtt{POUR}}$.

This, along with judicious parameter choices, and a state-of-the-art implementation of a zk-SNARK for arithmetic circuits [59] (see Section 3.2.4), results in a zk-SNARK prover running time of a few minutes and zk-SNARK verifier running time of a few milliseconds. This allows the DAP scheme implementation to be practical for deployment, as our experiments show.

Zerocash can be integrated into Bitcoin or forks of it (commonly referred to as "altcoins"); we later describe how this is done.

3.1.5 Paper organization

The remainder of this paper is organized as follows. Section 3.2 provides background on zk-SNARKs. We define DAP schemes in Section 3.3, and our construction thereof in Section 3.4. Section 3.5 discusses the concrete instantiation in Zerocash. Section 3.6 describes the integration of Zerocash into existing ledger-based currencies. Section 3.7 provides microbenchmarks for our prototype implementation, as well as results based on full-network simulations. Section 3.8 describes optimizations. We discuss concurrent work in Section 3.9 and summarize our contributions and

[3] Alternatively, we could have opted to rely on the circuit generators [57, 18, 59], which support various classes of C programs, by writing C code expressing the POUR checks. However, as discussed later, these generic approaches are more expensive than our hand-optimized construction.

future directions in Section 3.10.

3.2 Background on zk-SNARKs

The main cryptographic primitive used in this paper is a special kind of *Succinct Non-interactive ARgument of Knowledge* (SNARK). Concretely, we use a *publicly-verifiable preprocessing zero-knowledge* SNARK, or zk-SNARK for short. In this section we provide basic background on zk-SNARKs, provide an informal definition, compare zk-SNARKs with the more familiar notion of NIZKs, and recall known constructions and implementations.

3.2.1 Informal definition

We informally define zk-SNARKs for arithmetic circuit satisfiability. We refer the reader to, e.g., [55] for a formal definition.

For a field $\mathbb{F}$, an $\mathbb{F}$-*arithmetic circuit* takes inputs that are elements in $\mathbb{F}$, and its gates output elements in $\mathbb{F}$. We naturally associate a circuit with the function it computes. To model nondeterminism we consider circuits that have an *input* $x \in \mathbb{F}^n$ and an auxiliary input $a \in \mathbb{F}^h$, called a *witness*. The circuits we consider only have *bilinear gates*.[4] Arithmetic circuit satisfiability is defined analogously to the boolean case, as follows.

Definition 3.2.1. *The* arithmetic circuit satisfiability problem *of an* $\mathbb{F}$*-arithmetic circuit* $C\colon \mathbb{F}^n \times \mathbb{F}^h \to \mathbb{F}^l$ *is captured by the relation* $\mathcal{R}_C = \{(x, a) \in \mathbb{F}^n \times \mathbb{F}^h \colon C(x, a) = 0^l\}$*; its language is* $\mathcal{L}_C = \{x \in \mathbb{F}^n : \exists\, a \in \mathbb{F}^h \text{ s.t. } C(x, a) = 0^l\}$.

Given a field $\mathbb{F}$, a (publicly-verifiable preprocessing) **zk-SNARK** for $\mathbb{F}$-arithmetic circuit satisfiability is a triple of polynomial-time algorithms $(\mathsf{KeyGen}, \mathsf{Prove}, \mathsf{Verify})$:

- $\mathsf{KeyGen}(1^\lambda, C) \to (\mathsf{pk}, \mathsf{vk})$. On input a security parameter λ (presented in unary) and an $\mathbb{F}$-arithmetic circuit C, the *key generator* KeyGen probabilistically samples a *proving key* pk and a

[4] A gate with inputs $y_1, \ldots, y_m \in \mathbb{F}$ is *bilinear* if the output is $\langle \vec{a}, (1, y_1, \ldots, y_m) \rangle \cdot \langle \vec{b}, (1, y_1, \ldots, y_m) \rangle$ for some $\vec{a}, \vec{b} \in \mathbb{F}^{m+1}$. These include addition, multiplication, negation, and constant gates.

verification key vk. Both keys are published as public parameters and can be used, any number of times, to prove/verify membership in $\mathcal{L}_C$.

- $\mathsf{Prove}(\mathsf{pk}, x, a) \to \pi$. On input a proving key pk and any $(x, a) \in \mathcal{R}_C$, the *prover* Prove outputs a non-interactive proof π for the statement $x \in \mathcal{L}_C$.

- $\mathsf{Verify}(\mathsf{vk}, x, \pi) \to b$. On input a verification key vk, an input x, and a proof π, the *verifier* Verify outputs $b = 1$ if he is convinced that $x \in \mathcal{L}_C$.

A zk-SNARK satisfies the following properties.

Completeness. For every security parameter λ, any $\mathbb{F}$-arithmetic circuit C, and any $(x, a) \in \mathcal{R}_C$, the honest prover can convince the verifier. Namely, $b = 1$ with probability $1 - \mathrm{negl}(\lambda)$ in the following experiment: $(\mathsf{pk}, \mathsf{vk}) \leftarrow \mathsf{KeyGen}(1^\lambda, C)$; $\pi \leftarrow \mathsf{Prove}(\mathsf{pk}, x, a)$; $b \leftarrow \mathsf{Verify}(\mathsf{vk}, x, \pi)$.

Succinctness. An honestly-generated proof π has $O_\lambda(1)$ bits and $\mathsf{Verify}(\mathsf{vk}, x, \pi)$ runs in time $O_\lambda(|x|)$. (Here, O_λ hides a fixed polynomial factor in λ.)

Proof of knowledge (and soundness). If the verifier accepts a proof output by a bounded prover, then the prover "knows" a witness for the given instance. (In particular, soundness holds against bounded provers.) Namely, for every $\mathrm{poly}(\lambda)$-size adversary $\mathcal{A}$, there is a $\mathrm{poly}(\lambda)$-size extractor $\mathcal{E}$ such that $\mathsf{Verify}(\mathsf{vk}, x, \pi) = 1$ and $(x, a) \notin \mathcal{R}_C$ with probability $\mathrm{negl}(\lambda)$ in the following experiment: $(\mathsf{pk}, \mathsf{vk}) \leftarrow \mathsf{KeyGen}(1^\lambda, C)$; $(x, \pi) \leftarrow \mathcal{A}(\mathsf{pk}, \mathsf{vk})$; $a \leftarrow \mathcal{E}(\mathsf{pk}, \mathsf{vk})$.

Perfect zero knowledge. An honestly-generated proof is perfect zero knowledge.[5] Namely, there is a polynomial-time simulator Sim such that for all stateful distinguishers $\mathcal{D}$ the following two probabilities are equal:

[5] While most zk-SNARK descriptions in the literature only mention statistical zero knowledge, all zk-SNARK constructions can be made perfect zero knowledge by allowing for a negligible error probability in completeness.

$$\Pr\left[\begin{array}{c|r} (x,a) \in \mathcal{R}_C & (\mathsf{pk},\mathsf{vk}) \leftarrow \mathsf{KeyGen}(1^\lambda, C) \\ \mathcal{D}(\pi) = 1 & (x,a) \leftarrow \mathcal{D}(\mathsf{pk},\mathsf{vk}) \\ & \pi \leftarrow \mathsf{Prove}(\mathsf{pk}, x, a) \end{array}\right] \text{ and } \Pr\left[\begin{array}{c|r} (x,a) \in \mathcal{R}_C & (\mathsf{pk},\mathsf{vk},\mathsf{trap}) \leftarrow \mathsf{Sim}(1^\lambda, C) \\ \mathcal{D}(\pi) = 1 & (x,a) \leftarrow \mathcal{D}(\mathsf{pk},\mathsf{vk}) \\ & \pi \leftarrow \mathsf{Sim}(\mathsf{trap}, x) \end{array}\right] .$$

(the probability that $\mathcal{D}(\pi) = 1$ on an honest proof) (the probability that $\mathcal{D}(\pi) = 1$ on a simulated proof)

Remark. Both proof of knowledge and zero knowledge are essential to the use of zk-SNARKs in this paper. Indeed, we consider circuits C that verify assertions about cryptographic primitives (such as using a knowledge of SHA256 pre-image as a binding commitment). Thus it *does not suffice* to merely know that, for a given input x, a witness for $x \in \mathcal{L}_C$ *exists*. Instead, proof of knowledge ensures that a witness can be efficiently found (by extracting it from the prover) whenever the verifier accepts a proof. As for zero knowledge, it ensures that a proof leaks no information about the witness, beyond the fact that $x \in \mathcal{L}_C$.

Remark. In the security proofs (see Appendix B.3), we deal with provers producing a vector of inputs $\vec{x}$ together with a vector of corresponding proofs $\vec{\pi}$. In such cases, it is convenient to use an extractor that can extract a vector of witnesses $\vec{a}$ containing a valid witness for each valid proof. This "multi-instance" extraction follows from the "single-instance" one described above [65, 66]. Namely, if $(\mathsf{KeyGen}, \mathsf{Prove}, \mathsf{Verify})$ is a zk-SNARK, then for any poly(λ)-size prover adversary $\mathcal{A}$ there exists a poly(λ)-size extractor $\mathcal{E}$ such that

$$\Pr\left[\begin{array}{c|r} \exists\, i \text{ s.t. } \begin{array}{r} \mathsf{Verify}(\mathsf{vk}, x_i, \pi_i) = 1 \\ (x_i, a_i) \notin \mathcal{R}_C \end{array} & \begin{array}{r} (\mathsf{pk},\mathsf{vk}) \leftarrow \mathsf{KeyGen}(1^\lambda, C) \\ (\vec{x},\vec{\pi}) \leftarrow \mathcal{A}(\mathsf{pk},\mathsf{vk}) \\ \vec{a} \leftarrow \mathcal{E}(\mathsf{pk},\mathsf{vk}) \end{array} \end{array}\right] \leq \mathrm{negl}(\lambda) .$$

3.2.2 Comparison with NIZK

zk-SNARKs are related to a familiar cryptographic primitive: *non-interactive zero-knowledge proofs of knowledge* (NIZKs). Both zk-SNARKs and NIZKs require a one-time trusted setup of

public parameters (proving and verification keys for zk-SNARKs, and a common reference string for NIZKs). Both provide the same guarantees of completeness, proof of knowledge, and zero knowledge. The difference lies in efficiency guarantees. In a NIZK, the proof length and verification time depend on the NP language being proved. For instance, for the language of circuit satisfiability, the proof length and verification time in [67, 68] are linear in the circuit size. Conversely, in a zk-SNARK, proof length depends only on the security parameter, and verification time depends only on the instance size (and security parameter) but not on the circuit or witness size.

Thus, zk-SNARKs can be thought of as "succinct NIZKs", having short proofs and fast verification times. Succinctness comes with a caveat: known zk-SNARK constructions rely on stronger assumptions than NIZKs do (see below).

3.2.3 Known constructions and security

There are many zk-SNARK constructions in the literature [53, 54, 55, 56, 57, 18, 58, 59]. We are interested in zk-SNARKs for arithmetic circuit satisfiability, and the most efficient ones for this language are based on *quadratic arithmetic programs* [56, 55, 57, 18, 59]; such constructions provide a linear-time KeyGen, quasilinear-time Prove, and linear-time Verify.

Security of zk-SNARKs is based on knowledge-of-exponent assumptions and variants of Diffie–Hellman assumptions in bilinear groups [53, 69, 70]. While knowledge-of-exponent assumptions are fairly strong, there is evidence that such assumptions may be inherent for constructing zk-SNARKs [71, 65].

Remark (fully-succinct zk-SNARKs)**.** The key generator KeyGen takes a circuit C as input. Thus, KeyGen's running time is at least linear in the size of the circuit C. One could require KeyGen to *not* have to take C as input, and have its output keys work for *all* (polynomial-size) circuits C. In such a case, KeyGen's running time would be independent of C. A zk-SNARK satisfying this stronger property is *fully succinct*. Theoretical constructions of fully-succinct zk-SNARKs are known, based on various cryptographic assumptions [72, 73, 66]. Despite achieving essentially-optimal

asymptotics [74, 75, 76, 77, 66] no implementations of them have been reported in the literature to date.

3.2.4 zk-SNARK implementations

There are three published implementations of zk-SNARKs: (i) Parno et al. [57] present an implementation of zk-SNARKs for programs having no data dependencies;[6] (ii) Ben-Sasson et al. [18] present an implementation of zk-SNARKs for arbitrary programs (with data dependencies); and (iii) Ben-Sasson et al. [59] present an implementation of zk-SNARKs that supports programs that modify their own code (e.g., for runtime code generation); their implementation also reduces costs for programs of larger size and allows for universal key pairs.

Each of the works above also achieves zk-SNARKs for arithmetic circuit satisfiability as a stepping stone towards their respective higher-level efforts. In this paper we are only interested in a zk-SNARK for arithmetic circuit satisfiability, and we rely on the implementation of [59] for such a zk-SNARK.[7] The implementation in [59] is itself based on the protocol of Parno et al. [57]. We thus refer the interested reader to [57] for details of the protocol, its intuition, and its proof of security; and to [59] for the implementation and its performance. In terms of concrete parameters, the implementation of [59] provides 128 bits of security, and the field $\mathbb{F}$ is of a 256-bit prime order p.

3.3 Definition of a decentralized anonymous payment scheme

We introduce the notion of a *decentralized anonymous payment scheme* (DAP scheme), extending the notion of *decentralized e-cash* [23]. Later, in Section 3.4, we provide a construction.

[6] They only support programs where array indices are restricted to be known compile-time constants; similarly, loop iteration counts (or at least upper bounds to these) must be known at compile time.

[7] In [59], one optimization to the verifier's runtime requires preprocessing the verification key vk; for simplicity, we do not use this optimization.

3.3.1 Data structures

We begin by describing, and giving intuition about, the data structures used by a DAP scheme. The algorithms that use and produce these data structures are introduced in Section 3.3.2.

Basecoin ledger. Our protocol is applied on top of a ledger-based base currency such as Bitcoin; for generality we refer to this base currency as *Basecoin*. At any given time T, all users have access to L_T, the *ledger* at time T, which is a sequence of *transactions*. The ledger is append-only (i.e., $T < T'$ implies that L_T is a prefix of $L_{T'}$).[8] The transactions in the ledger include both Basecoin transactions as well as two new transaction types described below.

Public parameters. A list of *public parameters* pp is available to all users in the system. These are generated by a trusted party at the "start of time" and are used by the system's algorithms.

Addresses. Each user generates at least one *address key pair* $(\mathsf{addr}_{\mathsf{pk}}, \mathsf{addr}_{\mathsf{sk}})$. The public key $\mathsf{addr}_{\mathsf{pk}}$ is published and enables others to direct payments to the user. The secret key $\mathsf{addr}_{\mathsf{sk}}$ is used to receive payments sent to $\mathsf{addr}_{\mathsf{pk}}$. A user may generate any number of address key pairs.

Coins. A *coin* is a data object $\mathbf{c}$, to which we associate the following.

- A *coin commitment*, denoted $\mathsf{cm}(\mathbf{c})$: a string that appears on the ledger once $\mathbf{c}$ is *minted*.
- A *coin value*, denoted $v(\mathbf{c})$: the denomination of $\mathbf{c}$, as measured in basecoins, as an integer between 0 and a maximum value v_{max} (which is a system parameter).
- A *coin serial number*, denoted $\mathsf{sn}(\mathbf{c})$: a unique string associated with $\mathbf{c}$, used to prevent double spending.
- A *coin address*, denoted $\mathsf{addr}_{\mathsf{pk}}(\mathbf{c})$: an address public key, representing who owns $\mathbf{c}$.

Any other quantities associated with a coin $\mathbf{c}$ (e.g., various trapdoors) are implementation details.

New transactions. Besides Basecoin transactions, there are two new types of transactions.

- *Mint transactions.* A mint transaction $\mathsf{tx}_{\mathsf{Mint}}$ is a tuple $(\mathsf{cm}, v, *)$, where cm is a coin commitment, v is a coin value, and $*$ denotes other (implementation-dependent) information. The transaction

[8] In reality, the Basecoin ledger (such as the one of Bitcoin) is not perfect and may incur temporary inconsistencies. In this respect our construction is as good as the underlying ledger. We discuss the effects of this on anonymity and mitigations in Section 3.6.5.

$\mathsf{tx}_{\mathsf{Mint}}$ records that a coin $\mathbf{c}$ with coin commitment cm and value v has been minted.

- *Pour transactions.* A pour transaction $\mathsf{tx}_{\mathsf{Pour}}$ is a tuple $(\mathsf{rt}, \mathsf{sn}_1^{\mathsf{old}}, \mathsf{sn}_2^{\mathsf{old}}, \mathsf{cm}_1^{\mathsf{new}}, \mathsf{cm}_2^{\mathsf{new}}, v_{\mathsf{pub}}, \mathsf{info}, *)$, where rt is a root of a Merkle tree, $\mathsf{sn}_1^{\mathsf{old}}, \mathsf{sn}_2^{\mathsf{old}}$ are two coin serial numbers, $\mathsf{cm}_1^{\mathsf{new}}, \mathsf{cm}_2^{\mathsf{new}}$ are two coin commitments, v_{pub} is a coin value, info is an arbitrary string, and $*$ denotes other (implementation-dependent) information. The transaction $\mathsf{tx}_{\mathsf{Pour}}$ records the pouring of two input (and now consumed) coins $\mathbf{c}_1^{\mathsf{old}}, \mathbf{c}_2^{\mathsf{old}}$, with respective serial numbers $\mathsf{sn}_1^{\mathsf{old}}, \mathsf{sn}_2^{\mathsf{old}}$, into two new output coins $\mathbf{c}_1^{\mathsf{new}}, \mathbf{c}_2^{\mathsf{new}}$, with respective coin commitments $\mathsf{cm}_1^{\mathsf{new}}, \mathsf{cm}_2^{\mathsf{new}}$, as well as a public output v_{pub} (which may be zero). Furthermore, $\mathsf{tx}_{\mathsf{Pour}}$ also records an information string info (perhaps containing information on who is the recipient of v_{pub} basecoins) and that, when this transaction was made, the root of the Merkle tree over coin commitments was rt (see below).

Commitments of minted coins and serial numbers of spent coins. For any given time T,

- CMList_T denotes the list of all coin commitments appearing in mint and pour transactions in L_T;
- SNList_T denotes the list of all serial numbers appearing in pour transactions in L_T.

While both of these lists can be deduced from L_T, it will be convenient to think about them as separate (as, in practice, these may be separately maintained for efficiency reasons; cf. Section 3.8.3).

Merkle tree over commitments. For any given time T, Tree_T denotes a Merkle tree over CMList_T and rt_T its root. Moreover, the function $\mathsf{Path}_T(\mathsf{cm})$ gives the authentication path from a coin commitment cm appearing in CMList_T to the root of Tree_T.[9] For convenience, we assume that L_T also stores $\mathsf{rt}_{T'}$ for all $T' \leq T$ (i.e., it stores all past Merkle tree roots).

3.3.2 Algorithms

A DAP scheme Π is a tuple of polynomial-time algorithms

$$(\mathsf{Setup}, \mathsf{CreateAddress}, \mathsf{Mint}, \mathsf{Pour}, \mathsf{VerifyTransaction}, \mathsf{Receive})$$

with the following syntax and semantics.

[9] While we refer to Mekle trees for simplicity, it is straightforward to extend the definition to allow other data structures representing sets with fast insertion and efficient proofs of membership.

System setup. The algorithm Setup generates a list of public parameters:

Setup

- INPUTS: security parameter λ
- OUTPUTS: public parameters pp

The algorithm Setup is executed by a trusted party. The resulting public parameters pp are published and made available to all parties (e.g., by embedding them into the protocol's implementation). The setup is done *only once*; afterwards, no trusted party is needed, and no global secrets or trapdoors are kept.

Creating payment addresses. The algorithm $\mathsf{CreateAddress}$ generates a new address key pair:

$\mathsf{CreateAddress}$

- INPUTS: public parameters pp
- OUTPUTS: address key pair $(\mathsf{addr}_{\mathsf{pk}}, \mathsf{addr}_{\mathsf{sk}})$

Each user generates at least one address key pair $(\mathsf{addr}_{\mathsf{pk}}, \mathsf{addr}_{\mathsf{sk}})$ in order to receive coins. The public key $\mathsf{addr}_{\mathsf{pk}}$ is published, while the secret key $\mathsf{addr}_{\mathsf{sk}}$ is used to redeem coins sent to $\mathsf{addr}_{\mathsf{pk}}$. A user may generate any number of address key pairs; doing so does not require any interaction.

Minting coins. The algorithm Mint generates a coin (of a given value) and a mint transaction:

Mint

- INPUTS:
  - public parameters pp
  - coin value $v \in \{0, 1, \ldots, v_{\mathsf{max}}\}$
  - destination address public key $\mathsf{addr}_{\mathsf{pk}}$
- OUTPUTS: coin $\mathbf{c}$ and mint transaction $\mathsf{tx}_{\mathsf{Mint}}$

A system parameter, v_{max}, caps the value of any single coin. The output coin $\mathbf{c}$ has value v and coin address $\mathsf{addr}_{\mathsf{pk}}$; the output mint transaction $\mathsf{tx}_{\mathsf{Mint}}$ equals $(\mathsf{cm}, v, *)$, where cm is the coin commitment

of $\mathbf{c}$.

Pouring coins. The Pour algorithm transfers value from input coins into new output coins, marking the input coins as consumed. Moreover, a fraction of the input value may be publicly revealed. Pouring allows users to subdivide coins into smaller denominations, merge coins, and transfer ownership of anonymous coins, or make public payments.[10]

Pour

- INPUTS:
  - public parameters pp
  - the Merkle root rt
  - old coins $\mathbf{c}_1^{\mathsf{old}}, \mathbf{c}_2^{\mathsf{old}}$
  - old addresses secret keys $\mathsf{addr}_{\mathsf{sk},1}^{\mathsf{old}}, \mathsf{addr}_{\mathsf{sk},2}^{\mathsf{old}}$
  - authentication path path_1 from commitment $\mathsf{cm}(\mathbf{c}_1^{\mathsf{old}})$ to root rt, authentication path path_2 from commitment $\mathsf{cm}(\mathbf{c}_2^{\mathsf{old}})$ to root rt
  - new values $v_1^{\mathsf{new}}, v_2^{\mathsf{new}}$
  - new addresses public keys $\mathsf{addr}_{\mathsf{pk},1}^{\mathsf{new}}, \mathsf{addr}_{\mathsf{pk},2}^{\mathsf{new}}$
  - public value v_{pub}
  - transaction string info
- OUTPUTS: new coins $\mathbf{c}_1^{\mathsf{new}}, \mathbf{c}_2^{\mathsf{new}}$ and pour transaction $\mathsf{tx}_{\mathsf{Pour}}$

Thus, the Pour algorithm takes as input two distinct input coins $\mathbf{c}_1^{\mathsf{old}}, \mathbf{c}_2^{\mathsf{old}}$, along with corresponding address secret keys $\mathsf{addr}_{\mathsf{sk},1}^{\mathsf{old}}, \mathsf{addr}_{\mathsf{sk},2}^{\mathsf{old}}$ (required to redeem the two input coins). To ensure that $\mathbf{c}_1^{\mathsf{old}}, \mathbf{c}_2^{\mathsf{old}}$ have been previously minted, the Pour algorithm also takes as input the Merkle root rt (allegedly, equal to the root of Merkle tree over all coin commitments so far), along with two authentication paths $\mathsf{path}_1, \mathsf{path}_2$ for the two coin commitments $\mathsf{cm}(\mathbf{c}_1^{\mathsf{old}}), \mathsf{cm}(\mathbf{c}_2^{\mathsf{old}})$. Two input values $v_1^{\mathsf{new}}, v_2^{\mathsf{new}}$ specify the values of two new anonymous coins $\mathbf{c}_1^{\mathsf{new}}, \mathbf{c}_2^{\mathsf{new}}$ to be generated, and two input address

[10] We consider pours with 2 inputs and 2 outputs, for simplicity and (as discussed in Section 3.1.3) without loss of generality.

public keys $\mathsf{addr}^{\mathsf{new}}_{\mathsf{pk},1}, \mathsf{addr}^{\mathsf{new}}_{\mathsf{pk},2}$ specify the recipients of $\mathbf{c}^{\mathsf{new}}_1, \mathbf{c}^{\mathsf{new}}_2$. A third value, v_{pub}, specifies the amount to be publicly spent (e.g., to redeem coins or pay transaction fees). The sum of output values $v^{\mathsf{new}}_1 + v^{\mathsf{new}}_2 + v_{\mathsf{pub}}$ must be equal to the sum of the values of the input coins (and cannot exceed v_{max}). Finally, the Pour algorithm also receives an arbitrary string info, which is bound into the output pour transaction $\mathsf{tx}_{\mathsf{Pour}}$.

The Pour algorithm outputs two new coins $\mathbf{c}^{\mathsf{new}}_1, \mathbf{c}^{\mathsf{new}}_2$ and a pour transaction $\mathsf{tx}_{\mathsf{Pour}}$. The transaction $\mathsf{tx}_{\mathsf{Pour}}$ equals $(\mathsf{rt}, \mathsf{sn}^{\mathsf{old}}_1, \mathsf{sn}^{\mathsf{old}}_2, \mathsf{cm}^{\mathsf{new}}_1, \mathsf{cm}^{\mathsf{new}}_2, v_{\mathsf{pub}}, \mathsf{info}, *)$, where $\mathsf{cm}^{\mathsf{new}}_1, \mathsf{cm}^{\mathsf{new}}_2$ are the two coin commitments of the two output coins, and $*$ denotes other (implementation-dependent) information. Crucially, $\mathsf{tx}_{\mathsf{Pour}}$ reveals only one value, the public value v_{pub} (which may be zero); it does not reveal the payment addresses or values of the old or new coins.

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

VerifyTransaction

- INPUTS:
  - public parameters pp
  - a (mint or pour) transaction tx
  - the current ledger L
- OUTPUTS: bit b, equals 1 iff the transaction is valid

Both mint and pour transactions must be verified before being considered well-formed. In practice, transactions can be verified by the nodes in the distributed system maintaining the ledger, as well as by users who rely on these transactions.

Receiving coins. The algorithm Receive scans the ledger and retrieves unspent coins paid to a particular user address:

Receive

- INPUTS:
  - recipient address key pair $(\mathsf{addr}_{\mathsf{pk}}, \mathsf{addr}_{\mathsf{sk}})$

  - the current ledger L
- OUTPUTS: set of (unspent) received coins

When a user with address key pair $(\mathsf{addr}_{\mathsf{pk}}, \mathsf{addr}_{\mathsf{sk}})$ wishes to receive payments sent to $\mathsf{addr}_{\mathsf{pk}}$, he uses the Receive algorithm to scan the ledger. For each payment to $\mathsf{addr}_{\mathsf{pk}}$ appearing in the ledger, Receive outputs the corresponding coins whose serial numbers do not appear on the ledger L. Coins received in this way may be spent, just like minted coins, using the Pour algorithm. (We only require Receive to detect coins paid to $\mathsf{addr}_{\mathsf{pk}}$ via the Pour algorithm and not also detect coins minted by the user himself.)

Next, we describe completeness (Section 3.3.3) and security (Section 3.3.4).

3.3.3 Completeness

Completeness of a DAP scheme requires that unspent coins can be spent. More precisely, consider a *ledger sampler* $\mathcal{S}$ outputting a ledger L. If $\mathbf{c}_1$ and $\mathbf{c}_2$ are two coins whose coin commitments appear in (valid) transactions on L, but their serial numbers do not appear in L, then $\mathbf{c}_1$ and $\mathbf{c}_2$ can be spent using Pour. Namely, running Pour results in a pour transaction $\mathsf{tx}_{\mathsf{Pour}}$ that VerifyTransaction accepts, and the new coins can be received by the intended recipients (by using Receive); moreover, $\mathsf{tx}_{\mathsf{Pour}}$ correctly records the intended v_{pub} and transaction string info. This property is formalized via an *incompleteness experiment* INCOMP.

Definition 3.3.1. *A DAP scheme* $\Pi = (\mathsf{Setup}, \mathsf{CreateAddress}, \mathsf{Mint}, \mathsf{Pour}, \mathsf{VerifyTransaction}, \mathsf{Receive})$ *is* ***complete*** *if no polynomial-size ledger sampler* $\mathcal{S}$ *wins* INCOMP *with more than negligible probability. (See Appendix B.1 for details.)*

3.3.4 Security

Security of a DAP scheme is characterized by three properties, which we call *ledger indistinguishability*, *transaction non-malleability*, and *balance*.

Definition 3.3.2. *A DAP scheme* $\Pi = (\mathsf{Setup}, \mathsf{CreateAddress}, \mathsf{Mint}, \mathsf{Pour}, \mathsf{VerifyTransaction}, \mathsf{Receive})$ *is* ***secure*** *if it satisfies ledger indistinguishability, transaction non-malleability, and balance.*

Below, we provide an informal overview of each property, and defer formal definitions to Appendix B.2.

Each property is formalized as a game between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$. In each game, the behavior of honest parties is realized via a DAP scheme oracle $\mathcal{O}^{\mathsf{DAP}}$, which maintains a ledger L and provides an interface for executing CreateAddress, Mint, Pour and Receive algorithms for honest parties. To elicit behavior from honest parties, $\mathcal{A}$ passes a query to $\mathcal{C}$, which (after sanity checks) proxies the query to $\mathcal{O}^{\mathsf{DAP}}$. For each query that requests an honest party to perform an action, $\mathcal{A}$ specifies identities of previous transactions and the input values, and learns the resulting transaction, but not any of the secrets or trapdoors involved in producing that transaction. The oracle $\mathcal{O}^{\mathsf{DAP}}$ also provides an **Insert** query that allows $\mathcal{A}$ to directly add aribtrary transactions to the ledger L.

Ledger indistinguishability. This property captures the requirement that the ledger reveals no new information to the adversary beyond the publicly-revealed information (values of minted coins, public values, information strings, total number of transactions, etc.), even when the adversary can adaptively induce honest parties to perform DAP operations of his choice. That is, no bounded adversary $\mathcal{A}$ can distinguish between two ledgers L_0 and L_1, constructed by $\mathcal{A}$ using queries to two DAP scheme oracles, when the queries to the two oracles are *publicly consistent*: they have matching type and are identical in terms of publicly-revealed information and the information related to addresses controlled by $\mathcal{A}$.

Ledger indistinguishability is formalized by an experiment L-IND that proceeds as follows. First, a challenger samples a random bit b and initializes two DAP scheme oracles $\mathcal{O}_0^{\mathsf{DAP}}$ and $\mathcal{O}_1^{\mathsf{DAP}}$, maintaining ledgers L_0 and L_1. Throughout, the challenger allows $\mathcal{A}$ to issue queries to $\mathcal{O}_0^{\mathsf{DAP}}$ and $\mathcal{O}_1^{\mathsf{DAP}}$, thus controlling the behavior of honest parties on L_0 and L_1. The challenger provides the adversary with the view of both ledgers, but in randomized order: $L_{\mathsf{Left}} := L_b$ and $L_{\mathsf{Right}} := L_{1-b}$. The adversary's goal is to distinguish whether the view he sees corresponds to $(L_{\mathsf{Left}}, L_{\mathsf{Right}}) = (L_0, L_1)$,

i.e. $b = 0$, or to $(L_{\mathsf{Left}}, L_{\mathsf{Right}}) = (L_1, L_0)$, i.e. $b = 1$.

At each round of the experiment, the adversary issues queries in pairs Q, Q' of matching query type. If the query type is **CreateAddress**, then the same address is generated at both oracles. If it is to **Mint**, **Pour** or **Receive**, then Q is forwarded to L_0 and Q' to L_1; for **Insert** queries, query Q is forwarded to L_{Left} and Q' is forwarded to L_{Right}. The adversary's queries are restricted in the sense that they must maintain the *public consistency* of the two ledgers. For example, the public values for **Pour** queries must be the same, as well as minted amounts for **Mint** queries.

At the conclusion of the experiment, $\mathcal{A}$ outputs a guess b', and wins when $b = b'$. Ledger indistinguishability requires that $\mathcal{A}$ wins $\mathsf{L\text{-}IND}$ with probability at most negligibly greater than $1/2$.

Transaction non-malleability. This property requires that no bounded adversary $\mathcal{A}$ can alter any of the data stored within a (valid) pour transaction $\mathsf{tx}_{\mathsf{Pour}}$. This *transaction non-malleability* prevents malicious attackers from modifying others' transactions before they are added to the ledger (e.g., by re-targeting the Basecoin public output of a pour transaction).

Transaction non-malleability is formalized by an experiment $\mathsf{TR\text{-}NM}$, in which $\mathcal{A}$ adaptively interacts with a DAP scheme oracle $\mathcal{O}^{\mathsf{DAP}}$ and then outputs a pour transaction tx^*. Letting $\mathcal{T}$ denote the set of pour transactions returned by $\mathcal{O}^{\mathsf{DAP}}$, and L denote the final ledger, $\mathcal{A}$ wins the game if there exists $\mathsf{tx} \in \mathcal{T}$, such that (i) $\mathsf{tx}^* \neq \mathsf{tx}$; (ii) tx^* reveals a serial number contained in tx; and (iii) both tx and tx^* are valid with respect to the ledger L' containing all transactions preceding tx on L. In other words, $\mathcal{A}$ wins the game if tx^* manages to modify some previous pour transaction to spend the same coin in a different way.

Transaction non-malleability requires that $\mathcal{A}$ wins $\mathsf{TR\text{-}NM}$ with only negligible probability. (Note that $\mathcal{A}$ can of course produce valid pour transactions that are unrelated to those in $\mathcal{T}$; the condition that tx^* reveals a serial number of a previously-spent coin captures non-malleability.)

BAL. This property requires that no bounded adversary $\mathcal{A}$ can own more money than what he minted or received via payments from others.

BAL is formalized by an experiment BAL, in which $\mathcal{A}$ adaptively interacts with a DAP

scheme oracle $\mathcal{O}^{\mathsf{DAP}}$ and then outputs a set of coins S_{coin}. Letting ADDR be set of addresses returned by **CreateAddress** queries (i.e., addresses of "honest" users), $\mathcal{A}$ wins the game if the total value he can spend or has spent (either as coins or Basecoin public outputs) is greater than the value he has minted or received. That is, $\mathcal{A}$ wins if $v_{\mathsf{Unspent}} + v_{\mathsf{Basecoin}} + v_{\mathcal{A}\to\mathsf{ADDR}} > v_{\mathsf{Mint}} + v_{\mathsf{ADDR}\to\mathcal{A}}$ where: (i) v_{Unspent} is the total value of unspent coins in S_{coin}; (ii) v_{Basecoin} is the total value of public outputs placed by $\mathcal{A}$ on the ledger; (iii) v_{Mint} is the total value of $\mathcal{A}$'s mint transactions; (iv) $v_{\mathsf{ADDR}\to\mathcal{A}}$ is the total value of payments received by $\mathcal{A}$ from addresses in ADDR; (v) $v_{\mathcal{A}\to\mathsf{ADDR}}$ is the total value of payments sent by $\mathcal{A}$ to addresses in ADDR.

BAL requires that $\mathcal{A}$ wins BAL with only negligible probability.

3.4 Construction of a decentralized anonymous payment scheme

We show how to construct a DAP scheme (introduced in Section 3.3) using zk-SNARKs and other building blocks. Later, in Section 3.5, we give a concrete instantiation of this construction.

3.4.1 Cryptographic building blocks

We first introduce notation for the standard cryptographic building blocks that we use. We assume familiarity with the definitions of these building blocks; for more details, see, e.g., [78]. Throughout, λ denotes the security parameter.

Collision-resistant hashing. We use a collision-resistant hash function $\mathsf{CRH}\colon \{0,1\}^* \to \{0,1\}^{O(\lambda)}$.

Pseudorandom functions. We use a pseudorandom function family $\mathsf{PRF} = \{\mathsf{PRF}_x\colon \{0,1\}^* \to \{0,1\}^{O(\lambda)}\}_x$ where x denotes the seed. From PRF_x, we derive three "non-overlapping" pseudorandom functions, chosen arbitrarily as $\mathsf{PRF}^{\mathrm{addr}}_x(z) := \mathsf{PRF}_x(00\|z)$, $\mathsf{PRF}^{\mathrm{sn}}_x(z) := \mathsf{PRF}_x(01\|z)$, $\mathsf{PRF}^{\mathrm{pk}}_x(z) := \mathsf{PRF}_x(10\|z)$. Furthermore, we assume that $\mathsf{PRF}^{\mathrm{sn}}$ is also collision resistant, in the sense that it is infeasible to find $(x,z) \neq (x',z')$ such that $\mathsf{PRF}^{\mathrm{sn}}_x(z) = \mathsf{PRF}^{\mathrm{sn}}_{x'}(z')$.

Statistically-hiding commitments. We use a commitment scheme COMM where the bind-

ing property holds computationally, while the hiding property holds statistically. It is denoted $\{\mathsf{COMM}_x \colon \{0,1\}^* \to \{0,1\}^{O(\lambda)}\}_x$ where x denotes the commitment trapdoor. Namely, to reveal a commitment cm to a value z, it suffices to provide z and the trapdoor x; then one can check that $\mathsf{cm} = \mathsf{COMM}_x(z)$.

One-time strongly-unforgeable digital signatures. We use a digital signature scheme $\mathsf{Sig} = (\mathcal{G}_{\mathsf{sig}}, \mathcal{K}_{\mathsf{sig}}, \mathcal{S}_{\mathsf{sig}}, \mathcal{V}_{\mathsf{sig}})$ that works as follows.

- $\mathcal{G}_{\mathsf{sig}}(1^\lambda) \to \mathsf{pp}_{\mathsf{sig}}$. Given a security parameter λ (presented in unary), $\mathcal{G}_{\mathsf{sig}}$ samples public parameters $\mathsf{pp}_{\mathsf{sig}}$ for the encryption scheme.
- $\mathcal{K}_{\mathsf{sig}}(\mathsf{pp}_{\mathsf{sig}}) \to (\mathsf{pk}_{\mathsf{sig}}, \mathsf{sk}_{\mathsf{sig}})$. Given public parameters $\mathsf{pp}_{\mathsf{sig}}$, $\mathcal{K}_{\mathsf{sig}}$ samples a public key and a secret key for a single user.
- $\mathcal{S}_{\mathsf{sig}}(\mathsf{sk}_{\mathsf{sig}}, m) \to \sigma$. Given a secret key $\mathsf{sk}_{\mathsf{sig}}$ and a message m, $\mathcal{S}_{\mathsf{sig}}$ signs m to obtain a signature σ.
- $\mathcal{V}_{\mathsf{sig}}(\mathsf{pk}_{\mathsf{sig}}, m, \sigma) \to b$. Given a public key $\mathsf{pk}_{\mathsf{sig}}$, message m, and signature σ, $\mathcal{V}_{\mathsf{sig}}$ outputs $b = 1$ if the signature σ is valid for message m; else it outputs $b = 0$.

The signature scheme Sig satisfies the security property of *one-time strong unforgeability against chosen-message attacks* ($\mathsf{SUF\text{-}1CMA}$ security).

Key-private public-key encryption. We use a public-key encryption scheme $\mathsf{Enc} = (\mathcal{G}_{\mathsf{enc}}, \mathcal{K}_{\mathsf{enc}}, \mathcal{E}_{\mathsf{enc}}, \mathcal{D}_{\mathsf{enc}})$ that works as follows.

- $\mathcal{G}_{\mathsf{enc}}(1^\lambda) \to \mathsf{pp}_{\mathsf{enc}}$. Given a security parameter λ (presented in unary), $\mathcal{G}_{\mathsf{enc}}$ samples public parameters $\mathsf{pp}_{\mathsf{enc}}$ for the encryption scheme.
- $\mathcal{K}_{\mathsf{enc}}(\mathsf{pp}_{\mathsf{enc}}) \to (\mathsf{pk}_{\mathsf{enc}}, \mathsf{sk}_{\mathsf{enc}})$. Given public parameters $\mathsf{pp}_{\mathsf{enc}}$, $\mathcal{K}_{\mathsf{enc}}$ samples a public key and a secret key for a single user.
- $\mathcal{E}_{\mathsf{enc}}(\mathsf{pk}_{\mathsf{enc}}, m) \to c$. Given a public key $\mathsf{pk}_{\mathsf{enc}}$ and a message m, $\mathcal{E}_{\mathsf{enc}}$ encrypts m to obtain a ciphertext c.
- $\mathcal{D}_{\mathsf{enc}}(\mathsf{sk}_{\mathsf{enc}}, c) \to m$. Given a secret key $\mathsf{sk}_{\mathsf{enc}}$ and a ciphertext c, $\mathcal{D}_{\mathsf{enc}}$ decrypts c to produce a message m (or $\perp$ if decryption fails).

The encryption scheme Enc satisfies two security properties: (i) *ciphertext indistinguishability under*

chosen-ciphertext attack (IND-CCA security); and (ii) *key indistinguishability under chosen-ciphertext attack* (IK-CCA security). While the first property is standard, the second is less known; informally, IK-CCA requires that ciphertexts cannot be linked to the public key used to encrypt them, or to other ciphertexts encrypted with the same public key. For definitions, we refer the reader to [64].

3.4.2 zk-SNARKs for pouring coins

As outlined in Section 3.1.3, our construction invokes a zk-SNARK for a specific NP statement, POUR, which we now define. We first recall the context motivating POUR. When a user u *pours* "old" coins $\mathbf{c}_1^{\mathsf{old}}, \mathbf{c}_2^{\mathsf{old}}$ into new coins $\mathbf{c}_1^{\mathsf{new}}, \mathbf{c}_2^{\mathsf{new}}$, a corresponding pour transaction

$$\mathsf{tx}_{\mathsf{Pour}} = (\mathsf{rt}, \mathsf{sn}_1^{\mathsf{old}}, \mathsf{sn}_2^{\mathsf{old}}, \mathsf{cm}_1^{\mathsf{new}}, \mathsf{cm}_2^{\mathsf{new}}, v_{\mathsf{pub}}, \mathsf{info}, *)$$

is generated. In our construction, we need to provide evidence in "$*$" that various conditions were respected by the pour operation. Concretely, $\mathsf{tx}_{\mathsf{Pour}}$ should demonstrate that (i) u owns $\mathbf{c}_1^{\mathsf{old}}, \mathbf{c}_2^{\mathsf{old}}$; (ii) coin commitments for $\mathbf{c}_1^{\mathsf{old}}, \mathbf{c}_2^{\mathsf{old}}$ appear somewhere on the ledger; (iii) the revealed serial numbers $\mathsf{sn}_1^{\mathsf{old}}, \mathsf{sn}_2^{\mathsf{old}}$ are of $\mathbf{c}_1^{\mathsf{old}}, \mathbf{c}_2^{\mathsf{old}}$; (iv) the revealed coin commitments $\mathsf{cm}_1^{\mathsf{new}}, \mathsf{cm}_2^{\mathsf{new}}$ are of $\mathbf{c}_1^{\mathsf{new}}, \mathbf{c}_2^{\mathsf{new}}$; (v) balance is preserved. Our construction achieves this by including a zk-SNARK proof $\pi_{\mathtt{POUR}}$ for the statement POUR which checks the above invariants (as well as others needed for non-malleability).

The statement POUR. Concretely, the NP statement POUR is defined as follows.

- Instances are of the form $x = (\mathsf{rt}, \mathsf{sn}_1^{\mathsf{old}}, \mathsf{sn}_2^{\mathsf{old}}, \mathsf{cm}_1^{\mathsf{new}}, \mathsf{cm}_2^{\mathsf{new}}, v_{\mathsf{pub}}, h_{\mathsf{Sig}}, h_1, h_2)$. Thus, an instance x specifies a root rt for a CRH-based Merkle tree (over the list of commitments so far), the two serial numbers of the consumed coins, two coin commitments for the two new coins, a public value, and fields $h_{\mathsf{Sig}}, h_1, h_2$ used for non-malleability.

- Witnesses are of the form $a = (\mathsf{path}_1, \mathsf{path}_2, \mathbf{c}_1^{\mathsf{old}}, \mathbf{c}_2^{\mathsf{old}}, \mathsf{addr}_{\mathsf{sk},1}^{\mathsf{old}}, \mathsf{addr}_{\mathsf{sk},2}^{\mathsf{old}}, \mathbf{c}_1^{\mathsf{new}}, \mathbf{c}_2^{\mathsf{new}})$ where, for each

$i \in \{1,2\}$:

$$\begin{aligned}
\mathbf{c}_i^{\mathsf{old}} &= (\mathsf{addr}_{\mathsf{pk},i}^{\mathsf{old}}, v_i^{\mathsf{old}}, \rho_i^{\mathsf{old}}, r_i^{\mathsf{old}}, s_i^{\mathsf{old}}, \mathsf{cm}_i^{\mathsf{old}}) \ , \\
\mathbf{c}_i^{\mathsf{new}} &= (\mathsf{addr}_{\mathsf{pk},i}^{\mathsf{new}}, v_i^{\mathsf{new}}, \rho_i^{\mathsf{new}}, r_i^{\mathsf{new}}, s_i^{\mathsf{new}}, \mathsf{cm}_i^{\mathsf{new}}) \text{ for the same } \mathsf{cm}_i^{\mathsf{new}} \text{ as in } x, \\
\mathsf{addr}_{\mathsf{pk},i}^{\mathsf{old}} &= (a_{\mathsf{pk},i}^{\mathsf{old}}, \mathsf{pk}_{\mathsf{enc},i}^{\mathsf{old}}) \ , \\
\mathsf{addr}_{\mathsf{pk},i}^{\mathsf{new}} &= (a_{\mathsf{pk},i}^{\mathsf{new}}, \mathsf{pk}_{\mathsf{enc},i}^{\mathsf{new}}) \ , \\
\mathsf{addr}_{\mathsf{sk},i}^{\mathsf{old}} &= (a_{\mathsf{sk},i}^{\mathsf{old}}, \mathsf{sk}_{\mathsf{enc},i}^{\mathsf{old}}) \ .
\end{aligned}$$

Thus, a witness a specifies authentication paths for the two new coin commitments, the entirety of coin information about both the old and new coins, and address secret keys for the old coins.

Given a POUR instance x, a witness a is valid for x if the following holds:

1. For each $i \in \{1,2\}$:
   (a) The coin commitment $\mathsf{cm}_i^{\mathsf{old}}$ of $\mathbf{c}_i^{\mathsf{old}}$ appears on the ledger, i.e., path_i is a valid authentication path for leaf $\mathsf{cm}_i^{\mathsf{old}}$ with respect to root rt, in a CRH-based Merkle tree.
   (b) The address secret key $a_{\mathsf{sk},i}^{\mathsf{old}}$ matches the address public key of $\mathbf{c}_i^{\mathsf{old}}$, i.e., $a_{\mathsf{pk},i}^{\mathsf{old}} = \mathsf{PRF}_{a_{\mathsf{sk},i}^{\mathsf{old}}}^{\mathrm{addr}}(0)$.
   (c) The serial number $\mathsf{sn}_i^{\mathsf{old}}$ of $\mathbf{c}_i^{\mathsf{old}}$ is computed correctly, i.e., $\mathsf{sn}_i^{\mathsf{old}} = \mathsf{PRF}_{a_{\mathsf{sk},i}^{\mathsf{old}}}^{\mathrm{sn}}(\rho_i^{\mathsf{old}})$.
   (d) The coin $\mathbf{c}_i^{\mathsf{old}}$ is well-formed, i.e., $\mathsf{cm}_i^{\mathsf{old}} = \mathsf{COMM}_{s_i^{\mathsf{old}}}(\mathsf{COMM}_{r_i^{\mathsf{old}}}(a_{\mathsf{pk},i}^{\mathsf{old}} \| \rho_i^{\mathsf{old}}) \| v_i^{\mathsf{old}})$.
   (e) The coin $\mathbf{c}_i^{\mathsf{new}}$ is well-formed, i.e., $\mathsf{cm}_i^{\mathsf{new}} = \mathsf{COMM}_{s_i^{\mathsf{new}}}(\mathsf{COMM}_{r_i^{\mathsf{new}}}(a_{\mathsf{pk},i}^{\mathsf{new}} \| \rho_i^{\mathsf{new}}) \| v_i^{\mathsf{new}})$.
   (f) The address secret key $a_{\mathsf{sk},i}^{\mathsf{old}}$ ties h_{Sig} to h_i, i.e., $h_i = \mathsf{PRF}_{a_{\mathsf{sk},i}^{\mathsf{old}}}^{\mathrm{pk}}(i \| h_{\mathsf{Sig}})$.
2. Balance is preserved: $v_1^{\mathsf{new}} + v_2^{\mathsf{new}} + v_{\mathsf{pub}} = v_1^{\mathsf{old}} + v_2^{\mathsf{old}}$ (with $v_1^{\mathsf{old}}, v_2^{\mathsf{old}} \geq 0$ and $v_1^{\mathsf{old}} + v_2^{\mathsf{old}} \leq v_{\mathsf{max}}$).

Recall that in this paper zk-SNARKs are relative to the language of arithmetic circuit satisfiability (see Section 3.2); thus, we express the checks in POUR via an arithmetic circuit, denoted $C_{\texttt{POUR}}$. In particular, the depth d_{tree} of the Merkle tree needs to be hardcoded in $C_{\texttt{POUR}}$, and we thus make it a parameter of our construction (see below); the maximum number of supported coins is then $2^{d_{\mathsf{tree}}}$.

3.4.3 Algorithm constructions

We proceed to describe the construction of the DAP scheme $\Pi = (\mathsf{Setup}, \mathsf{CreateAddress}, \mathsf{Mint}, \mathsf{Pour}, \mathsf{VerifyTransaction}, \mathsf{Receive})$ whose intuition was given in Section 3.1.3. Figure 3.2 gives the pseudocode for each one of the six algorithms in Π, in terms of the building blocks introduced in Section 3.4.1 and Section 3.4.2. In the construction, we hardcode two quantities: the maximum value of a coin, v_{max}, and the depth of the Merkle tree, d_{tree}.

3.4.4 Completeness and security

Our main theorem states that the above construction is indeed a DAP scheme.

Theorem 3.4.1. *The tuple* $\Pi = (\mathsf{Setup}, \mathsf{CreateAddress}, \mathsf{Mint}, \mathsf{Pour}, \mathsf{VerifyTransaction}, \mathsf{Receive})$*, as defined in Section 3.4.3, is a complete (cf. Definition 3.3.1) and secure (cf. Definition 3.3.2) DAP scheme.*

We provide a proof of Theorem 3.4.1 in Appendix B.3. We note that our construction can be modified to yield statistical (i.e., everlasting) anonymity; see the discussion in Section 3.8.1.

Remark (trusted setup)**.** Security of Π relies on a trusted party running Setup to generate the public parameters (once and for all). This trust is needed for the transaction non-malleability and balance properties but not for ledger indistinguishability. Thus, even if a powerful espionage agency were to corrupt the setup, anonymity will *still* be maintained. Moreover, if one wishes to mitigate the trust requirements of this step, one can conduct the computation of Setup using secure multiparty computation techniques; we leave this to future work.

Remark (use of pp)**.** According to the definition of a DAP scheme (see Section 3.3), the public parameters pp are given as input to each one of the six algorithms; this is also how we presented our construction in Figure 3.2. However, in our construction, the public parameters pp equal a tuple $(\mathsf{pk}_{\mathtt{POUR}}, \mathsf{vk}_{\mathtt{POUR}}, \mathsf{pp}_{\mathsf{enc}}, \mathsf{pp}_{\mathsf{sig}})$, and not every algorithm needs every component of pp. Concretely, $\mathsf{CreateAddress}$ only needs $\mathsf{pp}_{\mathsf{enc}}$; Mint only the security parameter λ; Pour only $\mathsf{pk}_{\mathtt{POUR}}$ and $\mathsf{pp}_{\mathsf{sig}}$;

Setup

- INPUTS: security parameter λ
- OUTPUTS: public parameters pp

1. Construct $C_{\mathtt{POUR}}$ for $\mathtt{POUR}$ at security λ.
2. Compute $(\mathsf{pk}_{\mathtt{POUR}}, \mathsf{vk}_{\mathtt{POUR}}) := \mathsf{KeyGen}(1^\lambda, C_{\mathtt{POUR}})$.
3. Compute $\mathsf{pp}_{\mathsf{enc}} := \mathcal{G}_{\mathsf{enc}}(1^\lambda)$.
4. Compute $\mathsf{pp}_{\mathsf{sig}} := \mathcal{G}_{\mathsf{sig}}(1^\lambda)$.
5. Set $\mathsf{pp} := (\mathsf{pk}_{\mathtt{POUR}}, \mathsf{vk}_{\mathtt{POUR}}, \mathsf{pp}_{\mathsf{enc}}, \mathsf{pp}_{\mathsf{sig}})$.
6. Output pp.

CreateAddress

- INPUTS: public parameters pp
- OUTPUTS: address key pair $(\mathsf{addr}_{\mathsf{pk}}, \mathsf{addr}_{\mathsf{sk}})$

1. Compute $(\mathsf{pk}_{\mathsf{enc}}, \mathsf{sk}_{\mathsf{enc}}) := \mathcal{K}_{\mathsf{enc}}(\mathsf{pp}_{\mathsf{enc}})$.
2. Randomly sample a $\mathsf{PRF}^{\mathrm{addr}}$ seed a_{sk}.
3. Compute $a_{\mathsf{pk}} = \mathsf{PRF}^{\mathrm{addr}}_{a_{\mathsf{sk}}}(0)$.
4. Set $\mathsf{addr}_{\mathsf{pk}} := (a_{\mathsf{pk}}, \mathsf{pk}_{\mathsf{enc}})$.
5. Set $\mathsf{addr}_{\mathsf{sk}} := (a_{\mathsf{sk}}, \mathsf{sk}_{\mathsf{enc}})$.
6. Output $(\mathsf{addr}_{\mathsf{pk}}, \mathsf{addr}_{\mathsf{sk}})$.

Mint

- INPUTS:
  - public parameters pp
  - coin value $v \in \{0, 1, \ldots, v_{\mathsf{max}}\}$
  - destination address public key $\mathsf{addr}_{\mathsf{pk}}$
- OUTPUTS: coin $\mathbf{c}$ and mint transaction $\mathsf{tx}_{\mathsf{Mint}}$

1. Parse $\mathsf{addr}_{\mathsf{pk}}$ as $(a_{\mathsf{pk}}, \mathsf{pk}_{\mathsf{enc}})$.
2. Randomly sample a $\mathsf{PRF}^{\mathrm{sn}}$ seed ρ.
3. Randomly sample two COMM trapdoors r, s.
4. Compute $k := \mathsf{COMM}_r(a_{\mathsf{pk}} \| \rho)$.
5. Compute $\mathsf{cm} := \mathsf{COMM}_s(v \| k)$.
6. Set $\mathbf{c} := (\mathsf{addr}_{\mathsf{pk}}, v, \rho, r, s, \mathsf{cm})$.
7. Set $\mathsf{tx}_{\mathsf{Mint}} := (\mathsf{cm}, v, *)$, where $* := (k, s)$.
8. Output $\mathbf{c}$ and $\mathsf{tx}_{\mathsf{Mint}}$.

VerifyTransaction

- INPUTS:
  - public parameters pp
  - a (mint or pour) transaction tx
  - the current ledger L
- OUTPUTS: bit b, equals 1 iff the transaction is valid

1. If given a mint transaction $\mathsf{tx} = \mathsf{tx}_{\mathsf{Mint}}$:
   (a) Parse $\mathsf{tx}_{\mathsf{Mint}}$ as $(\mathsf{cm}, v, *)$, and $*$ as (k, s).
   (b) Set $\mathsf{cm}' := \mathsf{COMM}_s(v \| k)$.
   (c) Output $b := 1$ if $\mathsf{cm} = \mathsf{cm}'$, else output $b := 0$.
2. If given a pour transaction $\mathsf{tx} = \mathsf{tx}_{\mathsf{Pour}}$:
   (a) Parse $\mathsf{tx}_{\mathsf{Pour}}$ as $(\mathsf{rt}, \mathsf{sn}_1^{\mathsf{old}}, \mathsf{sn}_2^{\mathsf{old}}, \mathsf{cm}_1^{\mathsf{new}}, \mathsf{cm}_2^{\mathsf{new}}, v_{\mathsf{pub}}, \mathsf{info}, *)$, and $*$ as $(\mathsf{pk}_{\mathsf{sig}}, h_1, h_2, \pi_{\mathtt{POUR}}, \mathbf{C}_1, \mathbf{C}_2, \sigma)$.
   (b) If $\mathsf{sn}_1^{\mathsf{old}}$ or $\mathsf{sn}_2^{\mathsf{old}}$ appears on L (or $\mathsf{sn}_1^{\mathsf{old}} = \mathsf{sn}_2^{\mathsf{old}}$), output $b := 0$.
   (c) If the Merkle root rt is not on L, output $b := 0$.
   (d) Compute $h_{\mathsf{Sig}} := \mathsf{CRH}(\mathsf{pk}_{\mathsf{sig}})$.
   (e) Set $x := (\mathsf{rt}, \mathsf{sn}_1^{\mathsf{old}}, \mathsf{sn}_2^{\mathsf{old}}, \mathsf{cm}_1^{\mathsf{new}}, \mathsf{cm}_2^{\mathsf{new}}, v_{\mathsf{pub}}, h_{\mathsf{Sig}}, h_1, h_2)$.
   (f) Set $m := (x, \pi_{\mathtt{POUR}}, \mathsf{info}, \mathbf{C}_1, \mathbf{C}_2)$.
   (g) Compute $b := \mathcal{V}_{\mathsf{sig}}(\mathsf{pk}_{\mathsf{sig}}, m, \sigma)$.
   (h) Compute $b' := \mathsf{Verify}(\mathsf{vk}_{\mathtt{POUR}}, x, \pi_{\mathtt{POUR}})$, and output $b \wedge b'$.

Pour

- INPUTS:
  - public parameters pp
  - the Merkle root rt
  - old coins $\mathbf{c}_1^{\mathsf{old}}, \mathbf{c}_2^{\mathsf{old}}$
  - old addresses secret keys $\mathsf{addr}_{\mathsf{sk},1}^{\mathsf{old}}, \mathsf{addr}_{\mathsf{sk},2}^{\mathsf{old}}$
  - path path_1 from commitment $\mathsf{cm}(\mathbf{c}_1^{\mathsf{old}})$ to root rt, path path_2 from commitment $\mathsf{cm}(\mathbf{c}_2^{\mathsf{old}})$ to root rt
  - new values $v_1^{\mathsf{new}}, v_2^{\mathsf{new}}$
  - new addresses public keys $\mathsf{addr}_{\mathsf{pk},1}^{\mathsf{new}}, \mathsf{addr}_{\mathsf{pk},2}^{\mathsf{new}}$
  - public value v_{pub}
  - transaction string info
- OUTPUTS: new coins $\mathbf{c}_1^{\mathsf{new}}, \mathbf{c}_2^{\mathsf{new}}$ and pour transaction $\mathsf{tx}_{\mathsf{Pour}}$

1. For each $i \in \{1, 2\}$:
   (a) Parse $\mathbf{c}_i^{\mathsf{old}}$ as $(\mathsf{addr}_{\mathsf{pk},i}^{\mathsf{old}}, v_i^{\mathsf{old}}, \rho_i^{\mathsf{old}}, r_i^{\mathsf{old}}, s_i^{\mathsf{old}}, \mathsf{cm}_i^{\mathsf{old}})$.
   (b) Parse $\mathsf{addr}_{\mathsf{sk},i}^{\mathsf{old}}$ as $(a_{\mathsf{sk},i}^{\mathsf{old}}, \mathsf{sk}_{\mathsf{enc},i}^{\mathsf{old}})$.
   (c) Compute $\mathsf{sn}_i^{\mathsf{old}} := \mathsf{PRF}^{\mathrm{sn}}_{a_{\mathsf{sk},i}^{\mathsf{old}}}(\rho_i^{\mathsf{old}})$.
   (d) Parse $\mathsf{addr}_{\mathsf{pk},i}^{\mathsf{new}}$ as $(a_{\mathsf{pk},i}^{\mathsf{new}}, \mathsf{pk}_{\mathsf{enc},i}^{\mathsf{new}})$.
   (e) Randomly sample a $\mathsf{PRF}^{\mathrm{sn}}$ seed ρ_i^{new}.
   (f) Randomly sample two COMM trapdoors $r_i^{\mathsf{new}}, s_i^{\mathsf{new}}$.
   (g) Compute $k_i^{\mathsf{new}} := \mathsf{COMM}_{r_i^{\mathsf{new}}}(a_{\mathsf{pk},i}^{\mathsf{new}} \| \rho_i^{\mathsf{new}})$.
   (h) Compute $\mathsf{cm}_i^{\mathsf{new}} := \mathsf{COMM}_{s_i^{\mathsf{new}}}(v_i^{\mathsf{new}} \| k_i^{\mathsf{new}})$.
   (i) Set $\mathbf{c}_i^{\mathsf{new}} := (\mathsf{addr}_{\mathsf{pk},i}^{\mathsf{new}}, v_i^{\mathsf{new}}, \rho_i^{\mathsf{new}}, r_i^{\mathsf{new}}, s_i^{\mathsf{new}}, \mathsf{cm}_i^{\mathsf{new}})$.
   (j) Set $\mathbf{C}_i := \mathcal{E}_{\mathsf{enc}}(\mathsf{pk}_{\mathsf{enc},i}^{\mathsf{new}}, (v_i^{\mathsf{new}}, \rho_i^{\mathsf{new}}, r_i^{\mathsf{new}}, s_i^{\mathsf{new}}))$.
2. Generate $(\mathsf{pk}_{\mathsf{sig}}, \mathsf{sk}_{\mathsf{sig}}) := \mathcal{K}_{\mathsf{sig}}(\mathsf{pp}_{\mathsf{sig}})$.
3. Compute $h_{\mathsf{Sig}} := \mathsf{CRH}(\mathsf{pk}_{\mathsf{sig}})$.
4. Compute $h_1 := \mathsf{PRF}^{\mathrm{pk}}_{a_{\mathsf{sk},1}^{\mathsf{old}}}(1 \| h_{\mathsf{Sig}})$ and $h_2 := \mathsf{PRF}^{\mathrm{pk}}_{a_{\mathsf{sk},2}^{\mathsf{old}}}(2 \| h_{\mathsf{Sig}})$.
5. Set $x := (\mathsf{rt}, \mathsf{sn}_1^{\mathsf{old}}, \mathsf{sn}_2^{\mathsf{old}}, \mathsf{cm}_1^{\mathsf{new}}, \mathsf{cm}_2^{\mathsf{new}}, v_{\mathsf{pub}}, h_{\mathsf{Sig}}, h_1, h_2)$.
6. Set $a := (\mathsf{path}_1, \mathsf{path}_2, \mathbf{c}_1^{\mathsf{old}}, \mathbf{c}_2^{\mathsf{old}}, \mathsf{addr}_{\mathsf{sk},1}^{\mathsf{old}}, \mathsf{addr}_{\mathsf{sk},2}^{\mathsf{old}}, \mathbf{c}_1^{\mathsf{new}}, \mathbf{c}_2^{\mathsf{new}})$.
7. Compute $\pi_{\mathtt{POUR}} := \mathsf{Prove}(\mathsf{pk}_{\mathtt{POUR}}, x, a)$.
8. Set $m := (x, \pi_{\mathtt{POUR}}, \mathsf{info}, \mathbf{C}_1, \mathbf{C}_2)$.
9. Compute $\sigma := \mathcal{S}_{\mathsf{sig}}(\mathsf{sk}_{\mathsf{sig}}, m)$.
10. Set $\mathsf{tx}_{\mathsf{Pour}} := (\mathsf{rt}, \mathsf{sn}_1^{\mathsf{old}}, \mathsf{sn}_2^{\mathsf{old}}, \mathsf{cm}_1^{\mathsf{new}}, \mathsf{cm}_2^{\mathsf{new}}, v_{\mathsf{pub}}, \mathsf{info}, *)$ where $* := (\mathsf{pk}_{\mathsf{sig}}, h_1, h_2, \pi_{\mathtt{POUR}}, \mathbf{C}_1, \mathbf{C}_2, \sigma)$.
11. Output $\mathbf{c}_1^{\mathsf{new}}, \mathbf{c}_2^{\mathsf{new}}$ and $\mathsf{tx}_{\mathsf{Pour}}$.

Receive

- INPUTS:
  - public parameters pp
  - recipient address key pair $(\mathsf{addr}_{\mathsf{pk}}, \mathsf{addr}_{\mathsf{sk}})$
  - the current ledger L
- OUTPUTS: set of received coins

1. Parse $\mathsf{addr}_{\mathsf{pk}}$ as $(a_{\mathsf{pk}}, \mathsf{pk}_{\mathsf{enc}})$.
2. Parse $\mathsf{addr}_{\mathsf{sk}}$ as $(a_{\mathsf{sk}}, \mathsf{sk}_{\mathsf{enc}})$.
3. For each Pour transaction $\mathsf{tx}_{\mathsf{Pour}}$ on the ledger:
   (a) Parse $\mathsf{tx}_{\mathsf{Pour}}$ as $(\mathsf{rt}, \mathsf{sn}_1^{\mathsf{old}}, \mathsf{sn}_2^{\mathsf{old}}, \mathsf{cm}_1^{\mathsf{new}}, \mathsf{cm}_2^{\mathsf{new}}, v_{\mathsf{pub}}, \mathsf{info}, *)$, and $*$ as $(\mathsf{pk}_{\mathsf{sig}}, h_1, h_2, \pi_{\mathtt{POUR}}, \mathbf{C}_1, \mathbf{C}_2, \sigma)$.
   (b) For each $i \in \{1, 2\}$:
      i. Compute $(v_i, \rho_i, r_i, s_i) := \mathcal{D}_{\mathsf{enc}}(\mathsf{sk}_{\mathsf{enc}}, \mathbf{C}_i)$.
      ii. If $\mathcal{D}_{\mathsf{enc}}$'s output is not $\perp$, verify that:
         - $\mathsf{cm}_i^{\mathsf{new}}$ equals $\mathsf{COMM}_{s_i}(v_i \| \mathsf{COMM}_{r_i}(a_{\mathsf{pk}} \| \rho_i))$;
         - $\mathsf{sn}_i := \mathsf{PRF}^{\mathrm{sn}}_{a_{\mathsf{sk}}}(\rho_i)$ is not not on L.
      iii. If both checks succeed, output $\mathbf{c}_i := (\mathsf{addr}_{\mathsf{pk}}, v_i, \rho_i, r_i, s_i, \mathsf{cm}_i^{\mathsf{new}})$.

Figure 3.2: Construction of a DAP scheme using zk-SNARKs and other ingredients.

VerifyTransaction only $\mathsf{vk}_{\mathtt{POUR}}$; and Receive only λ. In particular, since we rely on zk-SNARKs to prove/verify POUR, $\mathsf{pk}_{\mathtt{POUR}}$ is of constant, but large, size, and is only required by Pour. All other components of pp are of small constant size.

Remark (checking received coins in ledger)**.** The algorithm Receive tests whether the serial number of a received coin already appears on the ledger, in order not to output coins that the user has already received and spent by himself. Other users are, in any case, unable to spend coins addressed to this user.

3.5 Zerocash

We describe a concrete instantiation of a DAP scheme; this instantiation forms the basis of Zerocash. Later, in Section 3.6, we discuss how Zerocash can be integrated with existing ledger-based currencies.

3.5.1 Instantiation of building blocks

We instantiate the DAP scheme construction from Section 3.4 (see Figure 3.2), aiming at a level of security of 128 bits. Doing so requires concrete choices, described next.

$\mathsf{CRH}, \mathsf{PRF}, \mathsf{COMM}$ from SHA256. Let $\mathcal{H}$ be the SHA256 compression function, which maps a 512-bit input to a 256-bit output. We mostly rely on $\mathcal{H}$, rather than the "full" hash, since this suffices for our fixed-size single-block inputs, and it simplifies the construction of $C_{\mathtt{POUR}}$ (see Section 3.5.2). We instantiate $\mathsf{CRH}, \mathsf{PRF}, \mathsf{COMM}$ via $\mathcal{H}$ (under suitable assumptions on $\mathcal{H}$).

First, we instantiate the collision-resistant function CRH as $\mathcal{H}(z)$ for $z \in \{0,1\}^{512}$; this function compresses "two-to-one", so it can be used to construct binary Merkle trees.[11]

Next, we instantiate the pseudorandom function $\mathsf{PRF}_x(z)$ as $\mathcal{H}(x\|z)$, with $x \in \{0,1\}^{256}$ as

[11] A single exception: we still compute h_{Sig} according to the full hash SHA256, rather than its compression function, because there is no need for this computation to be verified by $C_{\mathtt{POUR}}$.

the seed, and $z \in \{0,1\}^{256}$ as the input.[12] Thus, the derived functions are:

$$\mathsf{PRF}^{\mathrm{addr}}_x(z) := \mathcal{H}(x\|00\|z)\,, \quad \mathsf{PRF}^{\mathrm{sn}}_x(z) := \mathcal{H}(x\|01\|z)\,, \quad \mathsf{PRF}^{\mathrm{pk}}_x(z) := \mathcal{H}(x\|10\|z)\ ,$$

with $x \in \{0,1\}^{256}$ and $z \in \{0,1\}^{254}$.

As for the commitment scheme COMM, we only use it in the following pattern:

$$k := \mathsf{COMM}_r(a_{\mathsf{pk}}\|\rho)\ ,$$

$$\mathsf{cm} := \mathsf{COMM}_s(v\|k)\ .$$

Due to our instantiation of PRF, a_{pk} is 256 bits. So we can set ρ also to 256 bits and r to 256+128 = 384 bits; then we can compute

$$k := \mathsf{COMM}_r(a_{\mathsf{pk}}\|\rho) \quad \text{as} \quad \mathcal{H}(r\|[\mathcal{H}(a_{\mathsf{pk}}\|\rho)]_{128})\ .$$

Above, $[\cdot]_{128}$ denotes that we are truncating the 256-bit string to 128 bits (say, by dropping least-significant bits, as in our implementation). Heuristically, for any string $z \in \{0,1\}^{128}$, the distribution induced by $\mathcal{H}(r\|z)$ is 2^{-128}-close to uniform, and this forms the basis of the statistically-hiding property. For computing cm, we set coin values to be 64-bit integers (so that, in particular, $v_{\mathsf{max}} = 2^{64} - 1$ in our implementation), and then compute

$$\mathsf{cm} := \mathsf{COMM}_s(v\|k) \quad \text{as} \quad \mathcal{H}(k\|0^{192}\|v)\ .$$

Noticeably, above we are *ignoring* the commitment randomness s. The reason is that we already know that k, being the output of a statistically-hiding commitment, can serve as randomness for the next commitment scheme.

[12] This assumption is reminiscent of previous works analyzing the security of hash-based constructions (e.g., [79]). However in this work we assume that a portion of the compression function is the *seed* for the pseudorandom function, rather than using the chaining variable as in [79].

Instantiating the NP statement POUR. The above choices imply a concrete instantiation of the NP statement POUR (see Section 3.4.2). Specifically, in our implementation, POUR checks that the following holds, for each $i \in \{1, 2\}$:

- path_i is an authentication path for leaf $\mathsf{cm}_i^{\mathsf{old}}$ with respect to root rt, in a CRH-based Merkle tree;
- $a_{\mathsf{pk},i}^{\mathsf{old}} = \mathcal{H}(a_{\mathsf{sk},i}^{\mathsf{old}} \| 0^{256})$;
- $\mathsf{sn}_i^{\mathsf{old}} = \mathcal{H}(a_{\mathsf{sk},i}^{\mathsf{old}} \| 01 \| [\rho_i^{\mathsf{old}}]_{254})$;
- $\mathsf{cm}_i^{\mathsf{old}} = \mathcal{H}(\mathcal{H}(r_i^{\mathsf{old}} \| [\mathcal{H}(a_{\mathsf{pk},i}^{\mathsf{old}} \| \rho_i^{\mathsf{old}})]_{128}) \| 0^{192} \| v_i^{\mathsf{old}})$;
- $\mathsf{cm}_i^{\mathsf{new}} = \mathcal{H}(\mathcal{H}(r_i^{\mathsf{new}} \| [\mathcal{H}(a_{\mathsf{pk},i}^{\mathsf{new}} \| \rho_i^{\mathsf{new}})]_{128}) \| 0^{192} \| v_i^{\mathsf{new}})$; and
- $h_i = \mathcal{H}(a_{\mathsf{sk},i}^{\mathsf{old}} \| 10 \| b_i \| [h_{\mathsf{Sig}}]_{253})$ where $b_1 := 0$ and $b_2 := 1$.

Moreover, POUR checks that $v_1^{\mathsf{new}} + v_2^{\mathsf{new}} + v_{\mathsf{pub}} = v_1^{\mathsf{old}} + v_2^{\mathsf{old}}$, with $v_1^{\mathsf{old}}, v_2^{\mathsf{old}} \geq 0$ and $v_1^{\mathsf{old}} + v_2^{\mathsf{old}} < 2^{64}$.

Finally, as mentioned, in order for $C_{\texttt{POUR}}$ to be well-defined, we need to fix a Merkle-tree depth d_{tree}. In our implementation, we fix $d_{\mathsf{tree}} = 64$, and thus support up to 2^{64} coins.

Instantiating Sig. For the signature scheme Sig, we use ECDSA to retain consistency and compatibility with the existing `bitcoind` source code. However, standard ECDSA is malleable: both (r, s) and $(r, -s)$ verify as valid signatures. We use a non-malleable variant, where s is restricted to the "lower half" of field elements. While we are not aware of a formal SUF-1CMA proof for this variant, its use is consistent with proposals to resolve Bitcoin transaction malleability [80].[13]

Instantiating Enc. For the encryption scheme Enc, we use the key-private Elliptic-Curve Integrated Encryption Scheme (ECIES) [81]; it is one of the few standardized key-private encryption schemes with available implementations.

3.5.2 Arithmetic circuit for pouring coins

Our DAP scheme construction from Section 3.4 (see Figure 3.2) also requires zk-SNARKs relative to the NP statement POUR. These are obtained by invoking a zk-SNARK for arithmetic circuit satisfiability (see Section 3.2.4) on an arithmetic circuit $C_{\texttt{POUR}}$, which verifies the NP statement

[13] In practice, one might replace this ECDSA variant with an EC-Schnorr signature satisfying SUF-1CMA security with proper encoding of EC group elements; the performance would be similar.

`POUR`. In our instantiation, we rely on the implementation of [59] for the basic zk-SNARK (see Section 3.2.4), and apply it to the circuit $C_{\texttt{POUR}}$ whose construction is described next.

An arithmetic circuit for verifying SHA256's compression function

The vast majority of the "verification work" in `POUR` is verifying computations of $\mathcal{H}$, the compression function of SHA256 (see Section 3.5.1). Thus, we begin by discussing our construction of an arithmetic circuit $C_{\mathcal{H}}$ for verifying SHA256 computations. Later, in Section 3.5.2, we discuss the construction of $C_{\texttt{POUR}}$, given the circuit $C_{\mathcal{H}}$.

We wish to construct an arithmetic circuit $C_{\mathcal{H}}$ such that, for every 256-bit digest h and 512-bit input z, $(h, z) \in \mathcal{R}_{C_{\mathcal{H}}}$ if and only if $h = \mathcal{H}(z)$. Naturally, our goal is to minimize the size of $C_{\mathcal{H}}$. Our high-level strategy is to construct $C_{\mathcal{H}}$, piece by piece, by closely following the SHA256 official specification [82]. For each subcomputation of SHA256, we use nondeterminism and field operations to verify the subcomputation using as few gates as possible.

Overview of SHA256's compression function. The primitive unit in SHA256 is a 32-bit *word*. All subcomputations are simple word operations: three bitwise operations (`and`, `or`, `xor`), shift-right, rotate-right, and addition modulo 2^{32}. The compression function internally has a *state* of 8 words, initialized to a fixed value, and then transformed in 64 successive rounds by following the 64-word *message schedule* (deduced from the input z). The 256-bit output is the concatenation of the 8 words of the final state.

Representing a state. We find that, for each word operation (except for addition modulo 2^{32}), it is more efficient to verify the operation when its inputs are represented as separate wires, each carrying a bit. Thus, $C_{\mathcal{H}}$ maintains the 8-word state as 256 individual wires, and the 64-word message schedule as $64 \cdot 32$ wires.

Addition modulo 32. To verify addition modulo 2^{32} we use techniques employed in previous work [57, 18, 59]. Given two words A and B, we compute $\alpha := \sum_{i=0}^{31} 2^i (A_i + B_i)$. Because $\mathbb{F}$ has characteristic larger than 2^{33}, there is no wrap around; thus, field addition coincides with integer

addition. We then make a non-deterministic guess for the 33 bits α_i of α (including carry), and enforce consistency by requiring that $\alpha = \sum_{i=0}^{32} 2^i \alpha_i$. To ensure that each $\alpha_i \in \{0, 1\}$, we use a 33-gate subcircuit computing $\alpha_i(\alpha_i - 1)$, all of which must be 0 for the subcircuit to be satisfiable. Overall, verifying addition modulo 2^{32} only requires 34 gates. This approach extends in a straightforward way to summation of more than two terms.

Verifying the SHA256 message schedule. The first 16 words W_i of the message schedule are the 16 words of the 512-bit input z. The remaining 48 words are computed as $W_t := \sigma_1(W_{t-2}) + W_{t-7} + \sigma_0(W_{t-15}) + W_{t-16}$, where $\sigma_0(W) := \texttt{rotr}_7(W) \oplus \texttt{rotr}_{18}(W) \oplus \texttt{shr}_3(W)$ and σ_1 has the same structure but different rotation and shift constants.

The rotation and shift amounts are constants, so rotates and shifts can be achieved by suitable wiring to previously computed bits (or the constant 0 for high-order bits in `shr`). Thus, since the XOR of 3 bits can be computed using 2 gates, both σ_0 and σ_1 can be computed in 64 gates. We then compute (or more precisely, guess and verify) the addition modulo 2^{32} of the four terms.

Verifying the SHA256 round function. The round function modifies the 8-word state by changing two of its words and then permuting the 8-word result.

Each of the two modified words is a sum modulo 2^{32} of (i) round-specific constant words K_t; (ii) message schedule words W_t; and (iii) words obtained by applying simple functions to state words. Two of those functions are bitwise *majority* ($\mathsf{Maj}(A, B, C)_i = 0$ if $A_i + B_i + C_i \leq 1$ else 1) and bitwise *choice* ($\mathsf{Ch}(A, B, C)_i = B_i$ if $A_i = 1$, else C_i). We verify correct computation of Maj using 2 gates per output bit, and Ch with 1.

Then, instead of copying 6 unchanged state words to obtain the permuted result, we make the permutation implicit in the circuit's wiring, by using output wires of previous sub-computations (sometimes reaching 4 round functions back) as input wires to the current sub-computation.

Performance. Overall, we obtain an arithmetic circuit $C_{\mathcal{H}}$ for verifying SHA256's compression function with less than 30 000 arithmetic gates. See Figure 3.3 for a breakdown of gate counts.

Comparison with generic approaches. We constructed the circuit $C_{\mathcal{H}}$ from scratch. We could

Gate count for $C_{\mathcal{H}}$	
Message schedule	8032
All rounds	19 584
1 round (of 64)	306
Finalize	288
Total	27 904

Figure 3.3: Size of circuit $C_{\mathcal{H}}$ for SHA256's compression function.

have instead opted for more generic approaches: implement SHA256's compression function in a higher-level language, and use a circuit generator to obtain a corresponding circuit. However, generic approaches are significantly more expensive for our application, as we now explain.

Starting from the SHA256 implementation in PolarSSL (a popular cryptographic library) [83], it is fairly straightforward to write a C program for computing $\mathcal{H}$. We wrote such a program, and gave it as input to the circuit generator of [57]. The output circuit had 58160 gates, more than twice larger than our hand-optimized circuit.

Alternatively, we also compiled the same C program to TinyRAM, which is the architecture supported in [18]; we obtained a 5371-instruction assembly code that takes 5704 cycles to execute on TinyRAM. We could then invoke the circuit generator in [18] when given this TinyRAM program and time bound. However, each TinyRAM cycle costs ≈ 1000 gates, so the resulting circuit would have at least $5.7 \cdot 10^6$ gates, i.e., over 190 times larger than our circuit. A similar computation holds for the circuit generator in [59], which supports an even more flexible architecture.

Thus, overall, we are indeed much better off constructing $C_{\mathcal{H}}$ from scratch. Of course, this is not surprising, because a SHA256 computation is almost a "circuit computation": it does not make use of complex program flow, accesses to memory, and so on. Thus, relying on machinery developed to support much richer classes of programs does not pay off.

Arithmetic circuit for POUR

The NP statement POUR requires verifying membership in a Merkle tree based on $\mathcal{H}$, a few additional invocations of $\mathcal{H}$, and integer addition and comparison. We construct the circuit $C_{\texttt{POUR}}$ for POUR by combining various subcircuits verifying each of these. There remains to to discuss the

subcircuits for verifying membership in a Merkle tree (using the aforementioned subcircuit $C_{\mathcal{H}}$ for verifying invocations of $\mathcal{H}$), and integer addition and comparison.

Merkle tree membership. We need to construct an arithmetic circuit that, given a root rt, authentication path path, and coin commitment cm, is satisfied if and only if path is a valid authentication path for the leaf cm with respect to the root rt. The authentication path path includes, for each layer i, an auxiliary hash value h_i and a bit r_i specifying whether h_i was the left ($r_i = 0$) or the right ($r_i = 1$) child of the parent node. We then check membership in the Merkle tree by verifying invocations of $\mathcal{H}$, bottom-up. Namely, for $d = 64$, we set $k_{d-1} = \mathsf{cm}$; then, for each $i = d-1, \ldots, 1$, we set $B_i = h_i \| k_i$ if $r_i = 0$ else $k_i \| h_i$, and compute $k_{i-1} = \mathcal{H}(B_i)$. Finally we check that the root k_0 matches the given root rt.

Integer addition. We need to construct an arithmetic circuit that, given 64-bit integers A, B, C (presented as binary strings), is satisfied if and only if $C = A + B$ *over the integers*. Again relying on the fact that $\mathbb{F}$'s characteristic is sufficiently large, we do so by checking that $\sum_{i=0}^{63} 2^i c_i = \sum_{i=0}^{63} 2^i (b_i + a_i)$ over $\mathbb{F}$; this is enough, because there is no wrap around.

Integer comparison. We need to construct an arithmetic circuit that, given two 64-bit integers A, B (represented in binary), is satisfied if and only if $A + B$ fits in 64 bits (i.e. $A + B < 2^{64}$). We do so by checking that $\sum_{i=0}^{63} 2^i (b_i + a_i) = \sum_{i=0}^{63} c_i$ for *some* $c_i \in \{0, 1\}$. Indeed, if $A + B < 2^{64}$ then it suffices to take c_i as the binary representation of $A + B$. However, if $A + B \geq 2^{64}$ then no choice of c_i can satisfy the constraint as $\sum_{i=0}^{63} c_i \leq 2^{64} - 1$. Overall, this requires 65 gates (1 gate for the equality check, and 64 gates for ensuring that $c_0, \ldots, c_{63}$ are boolean).

Overall circuit sizes. See Figure 3.4 for the size of $C_{\mathtt{POUR}}$. More than 99% of the gates are devoted to verifying invocations of $\mathcal{H}$.

Gate count for $C_{\mathtt{POUR}}$	
Ensure $\mathsf{cm}_1^{\mathsf{old}}$ is in Merkle tree	1 802 304
(1 layer out of 64)	(28 161)
Ensure $\mathsf{cm}_2^{\mathsf{old}}$ is in Merkle tree	1 802 304
(1 layer out of 64)	(28 161)
Check computation of $\mathsf{sn}_1^{\mathsf{old}}, \mathsf{sn}_2^{\mathsf{old}}$	$2 \times 27\,904$
Check computation of $a_{\mathsf{pk},1}^{\mathsf{old}}, a_{\mathsf{pk},2}^{\mathsf{old}}$	$2 \times 27\,904$
Check computation of $\mathsf{cm}_1^{\mathsf{old}}, \mathsf{cm}_2^{\mathsf{old}}, \mathsf{cm}_1^{\mathsf{new}}, \mathsf{cm}_2^{\mathsf{new}}$	$4 \times 83\,712$
Check computation of h_1, h_2	$2 \times 27\,904$
Ensure that $v_1^{\mathsf{new}} + v_2^{\mathsf{new}} + v_{\mathsf{pub}} = v_1^{\mathsf{old}} + v_2^{\mathsf{old}}$	1
Ensure that $v_1^{\mathsf{old}} + v_2^{\mathsf{old}} < 2^{64}$	65
Miscellaneous	2384
Total	4 109 330

Figure 3.4: Size of the circuit $C_{\mathtt{POUR}}$, which verifies the statement `POUR`.

3.6 Integration with existing ledger-based currencies

Zerocash can be deployed atop any ledger (even one maintained by a central bank). Here, we briefly detail integration with the Bitcoin protocol. Unless explicitly stated otherwise, in the following section when referring to *Bitcoin*, and its unit of account *bitcoin* (plural bitcoins), we mean the underlying protocol and software, not the currency system. (The discussion holds, with little or no modification, for many forks of Bitcoin, also known as "altcoins", such as Litecoin.)

By introducing new transaction types and payment semantics, Zerocash breaks compatibility with the Bitcoin network. While Zerocash could be integrated into Bitcoin (the actual currency and its supporting software) via a "flag day" where a super-majority of Bitcoin miners simultaneously adopt the new software, we neither expect nor advise such integration in the near future and suggest using Zerocash in a separate altcoin.

Integrating Zerocash into Bitcoin consists of adding a new transaction type, Zerocash transactions, and modifying the protocol and software to invoke Zerocash's DAP interface to create and verify these transactions. There are at least two possible approaches to this integration. The first approach replaces all bitcoins with zerocoins, making all transactions anonymous at the cost of losing any additional Bitcoin functionality provided by, e.g., the Bitcoin scripting language (see Section 3.6.2). The second approach maintains this functionality, adding a *parallel* Zerocash currency,

zerocoin, which can be converted to and from bitcoin at a one-to-one rate (see Section 3.6.3). Options for protocol-level modifications for the later approach are discussed in Section 3.6.4; the former can be readily inferred. In Section 3.6.5 we discuss anonymizing the network layer of Bitcoin and anonymity safeguards.

Semantics of Bitcoin]

3.6.1 Semantics of Bitcoin

In Bitcoin, each transaction specifies a list of *inputs* (pointers to previous transactions) and *outputs* (each containing an amount and destination public key). Bitcoin protocol semantics require that:

1. the bitcoins provided as input to a transaction must sum to the value of the bitcoins output, plus a transaction fee[14];
2. the public key specified in the claimed output must match the signature claiming it (to prevent theft); and
3. outputs may only be used once as an input (to prevent double spending).

These properties are enforced by explicit checks that reveal both the amounts and identities involved in a transfer. These checks are done when before individual miners create a block (which consists of many transactions and the hash of the previous block) via a proof of work, and again when the rest of the network validates the block before appending it to the block chain — the distributed ledger of Bitcoin. Blocks containing transactions that do not pass these checks are rejected.

3.6.2 Integration by replacing the base currency

One approach is to alter the underlying system so that all monetary transactions are done using Zerocash, i.e., by invoking the DAP interface and writing/reading the associated transactions

[14]The sole exception is the *coinbase transaction* of a block, which is used to create new bitcoins as a reward for solving the cryptographic puzzle associated with the block. It takes no input coins.

in the distributed ledger.

As seen in Section 3.3, this suffices to offer the core functionality of payments, minting, merging, splitting, etc., while assuring users that all transactions using this currency are anonymous. However, this has several drawbacks: (1) All pour transactions incur the cost of generating a zk-SNARK proof. (2) If Bitcoin supports additional features, such as a scripting language for specifying conditions for claiming bitcoins (as in Bitcoin), then these features are lost.[15] (3) Bitcoin allows the flexibility of spending unconfirmed transactions; instead, with a Zerocash-only Bitcoin, this flexibility is lost: transactions must be confirmed before they can be spent. (And this imposes a minimal delay between receiving funds and spending them.)

3.6.3 Integration by hybrid currency

A different approach is to extend Bitcoin with a parallel, anonymized currency of "zerocoins", existing alongside bitcoins, using the same ledger, and with the ability to convert freely between the two. The behavior and functionality of regular bitcoins is unaltered; in particular, they may support functionality such as scripting.

In this approach, the Bitcoin ledger consists of Bitcoin-style transactions, containing inputs and outputs [9]. Each input is either a pointer to an output of a previous transaction (as in plain Bitcoin), or a Zerocash pour transaction (which contributes its *public* value, v_{pub}, of bitcoins to this transaction). Outputs are either an amount and destination public address/script (as in plain Bitcoin), or a Zerocash mint transaction (which consumes the input bitcoins to produce zerocoins). The usual invariant over bitcoins is maintained and checked in plain view: the sum of bitcoin inputs (including pours' v_{pub}) must be at least the sum of bitcoin outputs (including mints' v), and any difference is offered as a transaction fee. However, the accounting for zerocoins consumed and produced is done separately and implicitly by the DAP scheme.

The life cycle of a zerocoin is as follows.

[15] However, in principle `POUR` could be extended to include a scripting language interpreter.

Creating new zerocoins. A mint transaction consumes v worth of bitcoins as inputs, and outputs coin commitment worth v zerocoins. The v bitcoins are effectively destroyed, in exchange for the newly-minted zerocoins.

Spending zerocoins. Zerocoins can then be transferred, split, and merged into other zerocoins arbitrarily, via pour transactions which, instead of explicit inputs, include zero-knowledge proofs that such inputs exist. Pour transactions may optionally reveal a non-zero public output v_{pub}. This is either left unclaimed as a transaction fee,[16] placed into a standard Bitcoin transaction output (e.g., one paying to a public key) or consumed by a mint transaction. Thus, v_{pub} bitcoins are created ex nihilo (similarly to how coinbase transactions produce bitcoin outputs as mining reward), in exchange for destroying that amount of zerocoins. The Bitcoin outputs must be included in the transaction string info, which is included as part of a pour transaction; transaction non-malleability ensures that all this information is bound together.

Spending multiple zerocoins. To allow for pours to span more than two input and output coins, $\mathsf{tx}_{\mathsf{Pour}}$ structures may be chained together within one transaction by marking some output coin commitments as intermediates and having subsequent pours in the same transaction constructed relative to an ephemeral Merkle tree consisting of only the intermediates commitments. For example, a transaction might accept four input coins, with the first two Pour operations combining two of the inputs to produce an intermediate commitment each and a final Pour combining the two intermediate commitments into a final output new coin. Since the intermediate results are consumed instantly within the transaction, they need not be recorded in the global Merkle tree or have their serial numbers marked as spent.

Transaction fees. Collecting transaction fees is done as usual, via a coinbase transaction added to each block, which pays as mining reward the difference between the total inputs (bitcoin and pours' v_{pub}) and total outputs (bitcoin and mints' v) in this block. Payment is either in bitcoins or in newly-minted zerocoins (via a Mint).

[16] Since transaction fees may potentially be claimed by any node in the network, they represent the sole zerocoin output that cannot be hidden from public view even in a Zerocash-only system.

Validation and block generation. All transactions are verified via VerifyTransaction when they are received by a node. Any plain Bitcoin inputs and outputs are processed as usual, and any Zerocash inputs and outputs are checked using VerifyTransaction with the entire Bitcoin transaction fed in as info for authentication. Once these transactions are assembled into a candidate block, each transaction needs to be verified again to ensure its serial number has not become spent or its Merkle root invalid. If these checks pass, the set of new coin commitments and spent serial numbers output by the included transactions are added to the global sets, and the new Merkle root and a digest of the serial number list is stored in the new block.[17] Embedding this data simplifies statekeeping and allows nodes to readily verify they have the correct coin list and serial number list. Upon receiving a candidate block, nodes validate the block is formed correctly with respect to the above procedure.

Receiving payments. In order to receive payments to an address, users may scan the block chain by running the Receive on every pour transaction. Alternatively they may receive coin information via some out-of-band mechanism (e.g., via encrypted email). The former process is nearly identical to the one proposed for "stealth addresses" for Bitcoin. In the worst case, scanning the block chain requires a trial decryption of every ciphertext $\mathbf{C}$. We expect many scenarios to provide explicit notification, e.g., in interactive purchases where a communication channel already exists from the payer to the payee. (Implementations may opt to drop the receive mechanism entirely, and require out-of-band notification, in order to avoid storing the ciphertexts in the block chain.)

3.6.4 Extending the Bitcoin protocol to support the combined semantics

While the section above describes the life-cycle of a zerocoin and semantics of the system, there remains the question of how transactions acquire the above necessary semantics. Two implementation approaches are possible, with different engineering tradeoffs.

The first approach is to extend the protocol and its implementation with hard-coded validation of Zerocash transactions, reading them from new, designated fields in transactions and

[17] This can be stored in the coinbase transaction, as certain other data currently is, or in a new field in the block header.

running VerifyTransaction. In this case the zk-SNARK itself effectively replaces the scripting language for Zerocash transactions.

The second approach is to extend Bitcoin's scripting language by adding an opcode that invokes VerifyTransaction, with the requisite arguments embeded alongside the opcode script. Such transactions must be exempt from the requirement they reference an input (as they are Zerocash transactions are self-contained), and, like coinbase transactions, be able to create bitcoins ex nihilo (to account for v_{pub}). Moreover, while VerifyTransaction is run at the standard point in the Bitcoin transaction processing flow for evaluating scripts, the coin commitments and spent serial numbers are not actually added to CMList (resp., SNList) until their containing block is accepted (i.e., merely verifying a transaction does not have side effects).

3.6.5 Additional anonymity considerations

Zerocash only anonymizes the transaction ledger. Network traffic used to announce transactions, retrieve blocks, and contact merchants still leaks identifying information (e.g., IP addresses). Thus users need some anonymity network to safely use Zerocash. The most obvious way to do this is via Tor [84]. Given that Zerocash transactions are not low latency themselves, Mixnets (e.g., Mixminion [85]) are also a viable way to add anonymity (and one that, unlike Tor, is not as vulnerable to traffic analysis). Using mixnets that provide email-like functionality has the added benefit of providing an out-of-band notification mechanism that can replace Receive.

Additionally, although in theory all users have a single view of the block chain, a powerful attacker could potentially fabricate an additional block *solely* for a targeted user. Spending any coins with respect to the updated Merkle tree in this "poison-pill" block will uniquely identify the targeted user. To mitigate such attacks, users should check with trusted peers their view of the block chain and, for sensitive transactions, only spend coins relative to blocks further back in the ledger (since creating the illusion for multiple blocks is far harder).

3.7 Experiments

To measure the performance of Zerocash, we ran several experiments. First, we benchmarked the performance of the zk-SNARK for the NP statement POUR (Section 3.7.1) and of the six DAP scheme algorithms (Section 3.7.2). Second, we studied the impact of a higher block verification time via a simulation of a Bitcoin network (Section 3.7.3).

3.7.1 Performance of zk-SNARKs for pouring coins

Our zk-SNARK for the NP statement POUR is obtained by constructing an arithmetic circuit $C_{\mathtt{POUR}}$ for verifying POUR, and then invoking the generic implementation of zk-SNARK for arithmetic circuit satisfiability of [59] (see Section 3.2.4). The arithmetic circuit $C_{\mathtt{POUR}}$ is built from scratch and hand-optimized to exploit nondeterministic verification and the large field characteristic (see Section 3.5.2) .

Figure 3.5 reports performance characteristics of the resulting zk-SNARK for POUR. This includes three settings: single-thread performance on a laptop machine; and single-thread and multi-thread performance on a desktop machine. (The time measurements are the average of 10 runs, with standard deviation under 2.5%.) For instance, with single-thread code on the laptop machine, we obtain that:

- Key generation takes 7 min 48 s, and results in a proving key $\mathsf{pk}_{\mathtt{POUR}}$ of 896 MiB and a verification key $\mathsf{vk}_{\mathtt{POUR}}$ of 749 B. This is performed only once, as part of the Setup algorithm.
- Producing a proof $\pi_{\mathtt{POUR}}$ requires about 3 minutes; proofs have a constant size of 288 B. Proof generation is a subroutine of the Pour algorithm, and the resulting proof is included in the corresponding pour transaction.
- A proof $\pi_{\mathtt{POUR}}$ can be verified in only 8.5 ms. Proof verification is a subroutine of the VerifyTransaction algorithm, when it is given as input a pour transaction to verify.

		Intel Core i7-2620M @ 2.70GHz 12GB of RAM	Intel Core i7-4770 @ 3.40GHz 16GB of RAM	
		1 thread	1 thread	4 threads
KeyGen	Time	7 min 48 s	5 min 11 s	1 min 47 s
	Proving key	896 MiB		
	Verification key	749 B		
Prove	Time	2 min 55 s	1 min 59 s	46 s
	Proof	288 B		
Verify	Time	8.5 ms	5.4 ms	

Figure 3.5: Performance of our zk-SNARK for the NP statement POUR. ($N = 10$, $\sigma \leq 2.5\%$)

3.7.2 Performance of Zerocash algorithms

In Figure 3.6 we report performance characteristics for each of the six DAP scheme algorithms in our implementation (single-thread on our desktop machine). For VerifyTransaction, we separately report the cost of verifying mint and pour transactions and, in the latter case, we exclude the cost of scanning L (e.g., to check if a serial number is duplicate);[18] for the case of Receive, we report the cost to process a given pour transaction in L.

We obtain that:

- Setup takes about 5 minutes to run; its running time is dominated by the running time of KeyGen on $C_{\texttt{POUR}}$. (Either way, Setup is run only once.) The size of the resulting public parameters pp is dominated by the size of $\mathsf{pk}_{\texttt{POUR}}$.
- CreateAddress takes 326.0 ms to run. The size of the resulting address key pair is just a few hundred bytes.
- Mint takes 23 µs to run. It results in a coin of size 463 B and mint transaction of size 72 B.
- Pour takes about 2 minutes to run. Besides Setup, it is the only "expensive" algorithm to run; as expected, its running time is dominated by the running time of Prove. For a transaction string info, it results in (two new coins and) a pour transaction of size 996 B + |info|.
- VerifyTransaction takes 8.3 µs to verify a mint transaction and 5.7 ms to verify a pour transaction;

[18] Naturally, if SNList has 2^{64} serial numbers (the maximum possible in our implementation), then scanning is very expensive! However, we do not expect that a system like Zerocash will grow to 2^{64} transactions. Still, such a system may grow to the point that scanning SNList is too expensive. We detail possible mitigations to this in Section 3.8.3.

the latter's time is dominated by that of Verify, which checks the zk-SNARK proof $\pi_{\texttt{POUR}}$.

- Receive takes 1.6 ms per pour transaction.

Note that the above numbers do not include the costs of maintaining the Merkle tree because doing so is not the responsibility of the DAP scheme algorithms. Nevertheless, these additional costs are not large: (i) each update of the root of the CRH-based Merkle tree only requires d_{tree} invocations of CRH, and (ii) an authentication path consists of only d_{tree} digests of CRH. In our implementation, where $\mathsf{CRH} = \mathcal{H}$ (the SHA256 compression function) and $d_{\mathsf{tree}} = 64$, each update requires 64 invocations of $\mathcal{H}$ and an authentication path requires $64 \cdot 32\,\mathrm{B} = 2\,\mathrm{KiB}$ of storage.

Remark. If one does not want to rely on the ledger to communicate coins, via the ciphertexts $\mathbf{C}_1, \mathbf{C}_2$, and instead rely instead on some out-of-band mechanism (e.g., encrypted email), then the Receive algorithm is not needed, and moreover, many of the aforementioned sizes decrease because some pieces of data are not needed anymore; we denoted these pieces of data with "$\star$" in Figure 3.6. (E.g., the size of an address key pair is reduced to only 64 B, and the size of a coin to only 120 B.)

3.7.3 Large-scale network simulation

Because Bitcoin mining typically takes place on dedicated GPUs or ASICs, the CPU resources to execute the DAP scheme algorithms are often of minimal consequence to network performance. There is one potential exception to this rule: the VerifyTransaction algorithm must be run by all of the network nodes in the course of routine transaction validation. The time it takes to perform this verification may have significant impact on network performance.

In the Zerocash implementation (as in Bitcoin), every Zerocash transaction is verified at each hop as it is forwarded though the network and, potentially, again when blocks containing the transaction are verified. Verifying a block consists of checking the proof of work and validating the contained transactions. Thus Zerocash transactions may take longer to spread though the network and blocks containing Zerocash transactions may take longer to verify. While we are concerned with

		Intel Core i7-4770 @ 3.40GHz 16GB of RAM 1 thread
Setup	Time	5 min 17 s
	Size of pp	896 MiB
	size of $\mathsf{pk}_{\mathtt{POUR}}$	896 MiB
	size of $\mathsf{vk}_{\mathtt{POUR}}$	749 B
	⋆ size of $\mathsf{pp}_{\mathsf{enc}}$	0 B
	size of $\mathsf{pp}_{\mathsf{sig}}$	0 B
CreateAddress	Time	326.0 ms
	Size of $\mathsf{addr}_{\mathsf{pk}}$	343 B
	size of a_{pk}	32 B
	⋆ size of $\mathsf{pk}_{\mathsf{enc}}$	311 B
	Size of $\mathsf{addr}_{\mathsf{sk}}$	319 B
	size of a_{sk}	32 B
	⋆ size of $\mathsf{sk}_{\mathsf{enc}}$	287 B
Mint	Time	23 µs
	Size of coin $\mathbf{c}$	463 B
	size of $\mathsf{addr}_{\mathsf{pk}}$	343 B
	size of v	8 B
	size of ρ	32 B
	size of r	48 B
	size of s	0 B
	size of cm	32 B
	Size of $\mathsf{tx}_{\mathsf{Mint}}$	72 B
	size of cm	32 B
	size of v	8 B
	size of k	32 B
	size of s	0 B
Pour	Time	2 min 2.01 s
	Size of $\mathsf{tx}_{\mathsf{Pour}}$	996 B + $\lvert\mathsf{info}\rvert$
	size of rt	32 B
	size of $\mathsf{sn}_1^{\mathsf{old}}, \mathsf{sn}_2^{\mathsf{old}}$	2 × 32 B
	size of $\mathsf{cm}_1^{\mathsf{new}}, \mathsf{cm}_2^{\mathsf{new}}$	2 × 32 B
	size of v_{pub}	8 B
	size of info	$\lvert\mathsf{info}\rvert$
	size of $\mathsf{pk}_{\mathsf{sig}}$	66 B
	size of h_1, h_2	2 × 32 B
	size of $\pi_{\mathtt{POUR}}$	288 B
	⋆ size of $\mathbf{C}_1, \mathbf{C}_2$	2 × 173 B
	size of σ	64 B
VerifyTransaction	Time for mint tx	8.3 µs
	Time for pour tx (excludes L scan)	5.7 ms
Receive	Time (per pour tx)	1.6 ms

Figure 3.6: Performance of Zerocash algorithms. Above, we report the sizes of $\mathsf{pp}_{\mathsf{enc}}$ and $\mathsf{pp}_{\mathsf{sig}}$ as 0 B, because these parameters are "hardcoded" in the libraries we rely on for Enc and Sig. ($N = 10$ with $\sigma \leq 2.5\%$ for all except that, due to variability at short timescales, $\sigma(\mathsf{Mint}) \leq 3.3\,\mu s$ and $\sigma(\mathsf{VerifyTransaction}) \leq 1.9\,\mu s$)

the first issue, the potential impact of the second issue is cause for greater concern. This is because Zerocash transactions cannot be spent until they make it onto the ledger.

Because blocks are also verified at each hop before they are forwarded through the network, delays in block verification slow down the propagation of new blocks through the network. This causes nodes to waste CPU-cycles mining on out-of-date blocks, reducing the computational power

of the network and making it easier to mount a "51% attack" (dishonest majority of miners) on the distributed ledger.

It is a priori unclear whether this potential issue is a real concern. Bitcoin caches transaction verifications, so a transaction that was already verified when it propagated through the network need not be verified again when it is seen in a block. The unknown is what percentage of transactions in a block are actually in any given node's cache. We thus conduct a simulation of the Bitcoin network to investigate both the time it takes Zerocash transactions to make it onto the ledger and establish the effects of Zerocash transactions on block verification and propagation. We find that Zerocash transactions can be spent reasonably quickly and that the effects of increased block validation time are minimal.

Simulation design. Because Zerocash requires breaking changes to the Bitcoin protocol, we cannot test our protocol in the live Bitcoin network or even in the dedicated testnet. We must run our own private testnet. For efficiency and cost reasons, we would like to run as many Bitcoin nodes as possible on the least amount of hardware. This raises two issues. First, reducing the proof of work to practical levels while still preserving a realistic rate of new blocks is difficult (especially on virtualized hardware with variable performance). Second, the overhead of zk-SNARK verification prevents us from running many Bitcoin nodes on one virtualized server.

The frequency of new blocks can be modeled as a Poisson process with a mean of Λ_{block} seconds.[19] To generate blocks stochastically, we modify `bitcoind` to fix its block difficulty at a trivial level [20] and run a Poisson process, on the simulation control server, which trivially mines a block on a randomly selected node. This preserves the distribution of blocks, without the computational overhead of a real proof of work. Another Poisson process triggering mechanism, with a different mean Λ_{tx}, introduces new transactions at random network nodes.

[19] Since computational power is added to the Bitcoin network faster than the 2-week difficulty adjustment period, the frequency of block generation is actually skewed. As our experiments run for at most an hour, we ignore this.

[20] These code modifications have been rendered moot by the subsequent inclusion of a "regtest" mode in Bitcoin 0.9 that allows for precisely this type of behavior and block generation on command. At the time of our experiments, this feature was not available in a stable release. Future work should use this feature.

To differentiate which transactions represent normal Bitcoin expenditures versus which contain Zerocash pour transactions, simulated Zerocash transactions pay a unique amount of bitcoins (we set this value arbitrarily at 7 BTC). If a transaction's output matches this preset value, and it is not in verification cache, then our modified Bitcoin client inserts a 10 ms delay simulating the runtime of VerifyTransaction.[21] Otherwise transactions are processed as specified by the Bitcoin protocol. We vary the amount of simulated Zerocash traffic by varying the number of transactions with this particular output amount. This minimizes code changes and estimates only the generic impact of verification delays and not of any specific implementation choice.

Methodology. Recent research [60] suggests that the Bitcoin network contains 16,000 distinct nodes though most are likely no longer participating: approximately 3,500 are reachable at any given time. Each node has an average of 32 open connections to randomly selected peers. As of November 2013, the peak observed transaction rate for Bitcoin is slightly under one transaction per second [86].

In our simulation, we use a 1000-node network in which each node has an average of 32 peers, transactions are generated with a mean of $\Lambda_{\mathsf{tx}} = 1\,\mathrm{s}$, a duration of 1 hour, and a variable percentage ϵ of Zerocash traffic. To allow for faster experiments, instead of generating a block every 10 minutes as in Bitcoin, we create blocks at an average of every $\Lambda_{\mathsf{block}} = 150\,\mathrm{s}$ (as in Litecoin, a popular altcoin).

We run our simulation for different traffic mixes, where ϵ indicates the percentage of Zerocash transactions and $\epsilon \in \{0\%, 25\%, 50\%, 75\%, 100\%\}$. Each simulation is run on 200 Amazon EC2 general-purpose `m1.medium` instances, in one region on a `10.10./16` private network. On each instance, we deploy 5 instances of `bitcoind`.[22]

Results. Transactions are triggered by a blocking function call on the simulation control node that must connect to a random node and wait for it to complete sending a transaction. Because the Poisson process modeling transactions generates delays between such calls and not between the exact

[21] We used a generous delay of 10 ms (higher than the time reported in Figure 3.6) to leave room for machines slower than our desktop machine.

[22] Higher densities of nodes per VM resulted in issues initializing all of the `bitcoind` instances on boot.

points when the node actuals sends the transactions, the actual transaction rate is skewed. In our experiments the real transaction rate shifts away from our target of one per second to an average of one every 1.4 seconds.

In Figure 3.7 we plot three metrics for $\epsilon \in \{0\%, 25\%, 50\%, 75\%, 100\%\}$. Each is the average defined over the data from the entire run of the simulation for a given ϵ (i.e., they include multiple transactions and blocks).[23] *Transaction latency* is the interval between a transaction's creation and its inclusion in a block. *Block propagation time* comes in two flavors: (1) the average time for a new block to reach a node computed over the times for all nodes, and (2) the same average computed over only the last node to see the block.

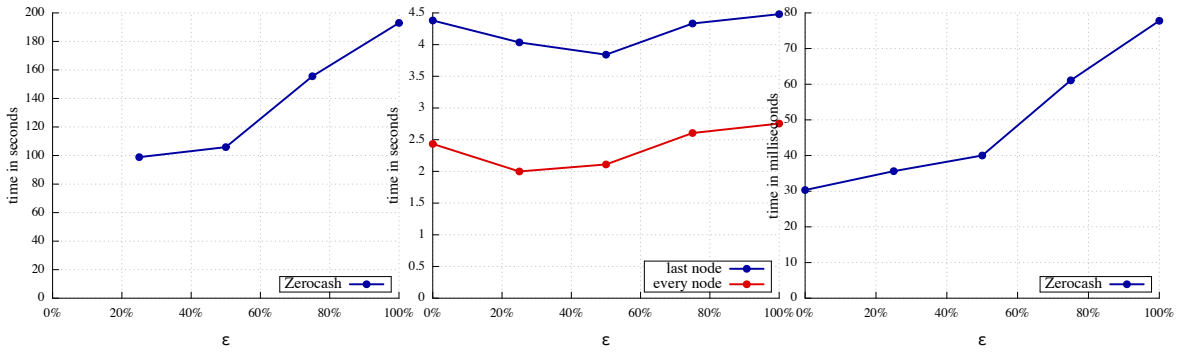
Block verification time is the average time, over all nodes, required to verify a block. If verification caching was not effective, we would expect to see a marked increase in both block verification time and propagation time. Since blocks occur on average every 150 s, and we expect approximately one transaction each second, we should see $150 \times 10\,\text{ms} = 1500\,\text{ms}$ of delay if all transactions were non-cached Zerocash transactions. Instead, we see worst case 80 ms and conclude caching is effective. This results in a negligible effect on block propagation (likely because network operations dominate).

The time needed for a transaction to be confirmed, and hence spendable, is roughly 190 s. For slower block generation rates (e.g., Bitcoin's block every 10 minutes) this should mean users must wait only one block before spending received transactions.

3.8 Optimizations and extensions

We outline several optimizations and extensions to Zerocash: everlasting anonymity (Section 3.8.1), faster block propagation (Section 3.8.2), and improved storage requirements (Section 3.8.3).

[23] Because our simulated Bitcoin nodes ran on shared EC2 instances, they were subject to variable external load, limiting the benchmark precision. Still, it clearly demonstrates that the mild additional delay does not cause catastrophic network effects.



(a) Transaction latency **(b)** Block propagation time **(c)** Block verification time

Figure 3.7: The average values of the three metrics we study, as a function of ϵ, the percentage of transactions that are Zerocash transactions. Note that, in (a), latency is undefined when $\epsilon = 0$ and hence omitted.

3.8.1 Everlasting anonymity

Since transactions may persist virtually forever on the ledger, users may wish to ensure the anonymity of their transactions also lasts forever, even if particular primitives are eventually broken (by cryptanalytic breakthrough, engineering progress, or quantum computers). As we now explain, the DAP scheme construction described in Section 3.4 is only computationally private, but can be modified to achieve *everlasting anonymity*.

Recall that every Pour operation publishes a pour transaction $\mathsf{tx}_{\mathsf{Pour}} = (\mathsf{rt}, \mathsf{sn}_1^{\mathsf{old}}, \mathsf{sn}_2^{\mathsf{old}}, \mathsf{cm}_1^{\mathsf{new}}, \mathsf{cm}_2^{\mathsf{new}}, v_{\mathsf{pub}}, \mathsf{info}, *)$, where $* = (\mathsf{pk}_{\mathsf{sig}}, h_1, h_2, \pi_{\mathtt{POUR}}, \mathbf{C}_1, \mathbf{C}_2, \sigma)$ and $\mathbf{C}_i = \mathcal{E}_{\mathsf{enc}}(\mathsf{pk}_{\mathsf{enc},i}^{\mathsf{new}}, (v_i^{\mathsf{new}}, \rho_i^{\mathsf{new}}, r_i^{\mathsf{new}}, s_i^{\mathsf{new}}))$. Observe that:

- Since $h_{\mathsf{Sig}} = \mathsf{CRH}(\mathsf{pk}_{\mathsf{sig}})$ and $h_i = \mathsf{PRF}^{\mathrm{pk}}_{a_{\mathsf{sk},i}^{\mathsf{old}}}(h_{\mathsf{Sig}})$, an unbounded adversary $\mathcal{A}$ can iterate over all x until $\mathsf{PRF}^{\mathrm{pk}}_x(h_{\mathsf{Sig}})$ equals h_i; with overwhelming probability, there is only one such x, in which case it equals $a_{\mathsf{sk},i}^{\mathsf{old}}$. Thus, $\mathcal{A}$ learns $a_{\mathsf{sk},i}^{\mathsf{old}}$, and hence $a_{\mathsf{pk},i}^{\mathsf{old}} := \mathsf{PRF}^{\mathrm{addr}}_{a_{\mathsf{sk},i}^{\mathsf{old}}}(0)$. This identifies the sender.

- An unbounded $\mathcal{A}$ can also decrypt $\mathbf{C}_i$, so to learn $(v_i^{\mathsf{new}}, \rho_i^{\mathsf{new}}, r_i^{\mathsf{new}}, s_i^{\mathsf{new}})$; then, $\mathcal{A}$ can try all possible x until $\mathsf{COMM}_{s_i^{\mathsf{new}}}(v_i^{\mathsf{new}} \| \mathsf{COMM}_{r_i^{\mathsf{new}}}(\mathsf{PRF}^{\mathrm{addr}}_x(0) \| \rho_i^{\mathsf{new}}))$ equals $\mathsf{cm}_i^{\mathsf{new}}$; with overwhelming probability, there is only one such x, in which case it equals $a_{\mathsf{sk},i}^{\mathsf{new}}$. This identifies the recipient.

The above attacks can be prevented as follows. First, every sender must use any given address

only once (for receiving or sending coins): after receiving a coin $\mathbf{c}$, a user u should immediately generate a new address and pour $\mathbf{c}$ into a fresh one $\mathbf{c}'$ relative to the new address; only afterwards can u spend the coin. Second, a user should not put any data in a ciphertext $\mathbf{C}_i$ to communicate a coin's information, but must instead use some (informationally-secure) out-of-band channel to do so. With these modifications (and recalling that COMM is statistically hiding and $\pi_{\mathtt{POUR}}$ is a perfect-zero-knowledge proof), one can verify that the pour transaction $\mathsf{tx}_{\mathsf{Pour}}$ is statistically hiding, i.e., leaks no information even to unbounded adversaries.[24]

3.8.2 Fast block propagation

As mentioned in Section 3.7.3, the higher block-verification time of Zerocash compared to, e.g., Bitcoin does not greatly effect block propagation. Even so, we note a simple modification that further mitigates concerns. Upon receiving a block, a node validates the proof of work and (optionally) transactions other than mint and pour, and then forward the block right away. Only afterwards, the node executes $\mathsf{VerifyTransaction}$ on any mint/pour transactions, before accepting it for use in transacting. Thus, blocks are still validated by every node (so the security properties are unhampered), and propagation delays in the broadcast of blocks are reduced.

In principle, this opens the possibility of a denial-of-service attack, in which the network is spammed with invalid blocks which pass the proof-of-work check but contain invalid mint or pour transactions. However, this attack appears unrealistic given the enormous (by design) cost of creating blocks passing the proof-of-work check.

3.8.3 Improved storage requirements

Beyond the ledger L, users need to maintain two lists: CMList, the list of all coin commitments, and SNList, the list of all serial numbers of spent coins (see Section 3.3.1). In our construction, CMList is required to deduce authentication paths to create new pour transactions (via Pour), while

[24] As for mint transactions, one can verify that they are already statistically hiding, without any modifications.

SNList is used to verify pour transactions (via $\mathsf{VerifyTransaction}$). As the ledger grows, both CMList and SNList grow in size, and can eventually impose substantial storage requirements (though both are derived from, and smaller than, the block chain per se). We now explain how these storage requirements can be mitigated, by relying on smaller representations of CMList and SNList that suffice within our construction.

Supporting many coin commitments

To execute the Pour algorithm to spend a coin $\mathbf{c}$, a user u needs to provide an authentication path from $\mathbf{c}$'s coin commitment to rt, the Merkle-tree root over CMList. If we make the following protocol modifications, u does not need all of CMList to compute this authentication path.

In each block B of transactions, we store the Merkle-tree path path_B from the first coin commitment in B to the root rt_B of the Merkle tree over CMList when the last block in the ledger is B. (In Zerocash, the additional per-block storage cost to store this information is only 2 KiB.)

Note that, given a block B and its successor block B', the corresponding authentication paths path_B and $\mathsf{path}_{B'}$ can be easily checked for consistency as follows. Let CMList_B and $\mathsf{CMList}_{B'}$ be the two lists of coin commitments corresponding to the two ledgers ending in block B and B' respectively; since CMList_B (i.e., coin commitments to "to the left" of path_B) is a prefix of $\mathsf{CMList}_{B'}$, $\mathsf{path}_{B'}$ can be computed from path_B and B in time $O(|B|d_{\mathsf{tree}})$, where d_{tree} is the tree depth.

When the user u first receives (or mints) the coin $\mathbf{c}$, and its coin commitment is included in a block B, u immediately computes path_B, by using the predecessor block and its authentication path. Afterwards, each time a new block is added to the ledger, u obtains a new path for $\mathbf{c}$ by using the new block and the old path for $\mathbf{c}$. Thus, u only needs to act each time a new block is added, and each such update costs $O(d_{\mathsf{tree}})$ per transaction in the block.

Overall, u incurs a storage requirement of only $O(d_{\mathsf{tree}})$ for each coin he owns, and does not need to store CMList anymore.

Supporting many spent serial numbers

To execute the VerifyTransaction algorithm on a pour transaction $\mathsf{tx}_{\mathsf{Pour}}$, a user u needs access to SNList (in order to check for duplicate serial numbers). Note, in Bitcoin, nodes need to maintain only the list of unspent transaction outputs, which is pruned as outputs are spent. In a DAP scheme, in contrast, nodes have to maintain SNList, which is a list that *always grows*. We now explain how to mitigate this storage requirement, in three incremental steps.

Step 1. The first step is to build a Merkle tree over SNList so to allow easy-to-verify non-membership proofs for SNList; this can be done by letting the leaves of the Merkle tree be the intervals of unspent serial numbers. Then, given the root rt of such tree, a serial number sn claimed to be unspent, and an authentication path path for an interval I, the user can check that path is valid for rt and that sn lies in I; the root rt and path path would be part of the pour transaction $\mathsf{tx}_{\mathsf{Pour}}$ to be verified. The problem with this approach, however, is that generating path (and also updating rt) requires knowledge of all of SNList.

Step 2. Next, instead of maintaining SNList in a single Merkle tree, we divide SNList, maintaining its chronological order, into sublists of serial numbers $\mathsf{SNList}_0, \mathsf{SNList}_1, \ldots$ and build a Merkle tree over the intervals induced by each sublist (i.e., apply Step 1 to each sublist). This modification implies a corresponding modification for the auxiliary information stored in a pour transaction that allows VerifyTransaction to check it. Now, however, producing such auxiliary information is less expensive. Indeed, a user with a coin $\mathbf{c}$ should maintain a list of authentication paths $\mathsf{path}_{\mathbf{c},0}, \mathsf{path}_{\mathbf{c},1}, \ldots$ (one for each sublist). Only the last path, corresponding to the active sublist, needs to be updated when a serial number is added; the other sublists and authentication paths remain unchanged (and these old sublists can in fact be discarded). When the user spends the coin, he can simply include these paths in the pour transaction. While updating these paths is an efficient operation, computing the initial paths for $\mathbf{c}$ is not, as it still requires the full set of sublists.

Step 3. To enable users to avoid the initial cost of computing paths for a new coin, we proceed as

follows. First, a coin $\mathbf{c}$ is extended to contain a time stamp $T_{\mathbf{c}}$ corresponding to when $\mathbf{c}$ is created (minted or poured into); the coin's commitment is modified to depend on the timestamp, and the timestamp is included in the clear within the transaction that creates the coin. Then, a user, upon spending $\mathbf{c}$, produces a zk-SNARK for the following NP statement: "for each Merkle-tree root created (or updated) after $T_{\mathbf{c}}$ there is an interval and an authentication path for that interval such that the serial number of $\mathbf{c}$ is in that interval". Depending on the number of Merkle trees in such an NP statement, such proofs may already be more efficient to produce, compared to the naive (Step 1) solution, using existing zk-SNARK implementations.

3.9 Related work

Danezis et al. [62] suggest using zk-SNARKs to reduce proof size and verification time in Zerocoin. Our work differs from [62] in both supported functionality and scalability.

First, [62]'s protocol, like Zerocoin, only supports fixed-value coins, and is best viewed as a decentralized mix. Instead, we define, construct, and implement a full-fledged decentralized electronic currency, which provides anonymous payments of any amount.

Second, in [62], the complexity of the zk-SNARK generator, prover, and verifier all scale superlinearly in the number of coins, because their arithmetic circuit computes, *explicitly*, a product over all coins. In particular, the number of coins "mixed together" for anonymity cannot be large. Instead, in our construction, the respective complexities are polylogarithmic, polylogarithmic, and constant in the number of coins; our approach supports a practically-unbounded number of coins.

While we do not rely on Pedersen commitments, our approach also yields statistical (i.e., everlasting) anonymity; see the discussion in Section 3.8.1.

3.10 Conclusion

Decentralized currencies should ensure a user's privacy from his peers when conducting legitimate financial transactions. Zerocash provides such privacy protection, by hiding user identities, transaction amounts, and account balances from public view. This, however, may be criticized for hampering accountability, regulation, and oversight. Yet Zerocash need not be limited to enforcing the basic monetary invariants of a currency system. The underlying zk-SNARK cryptographic proof machinery is flexible enough to support a wide range of policies. It can, for example, let a user prove that he paid his due taxes on all transactions *without* revealing those transactions, their amounts, or even the amount of taxes paid. As long as the policy can be specified by efficient nondeterministic computation using NP statements, it can (in principle) be enforced using zk-SNARKs, and added to Zerocash. This can enable automated, privacy-preserving verification and enforcement of a wide range of compliance and regulatory policies that would otherwise be invasive to check directly or might be bypassed by corrupt authorities. This raises research, policy, and engineering questions regarding which such policies are desirable and practically realizable.

Another research question is what new functionality can be realized by augmenting the capabilities already present in Bitcoin's scripting language with zk-SNARKs that allow fast verification of expressive statements.

Acknowledgments

We thank Amazon for their assistance and kind donation of EC2 resources, and Gregory Maxwell for his advice regarding the Bitcoin codebase. We thank Iddo Ben-Tov and the SCIPR Lab members — Daniel Genkin, Lior Greenblatt, Shaul Kfir, Gil Timnat, and Michael Riabzev — for inspiring discussions. We thank Sharon Kessler for editorial advice.

This work was supported by: Amazon.com through an AWS in Education research grant; the Broadcom Foundation and Tel Aviv University Authentication Initiative; the Center for Science of Information (CSoI), an NSF Science and Technology Center, under grant agreement CCF-0939370; the Check Point Institute for Information Security; the U.S. Defense Advanced Research Projects Agency (DARPA) and the Air Force Research Laboratory (AFRL) under contract FA8750-11-2-0211; the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement number 240258; the Israeli Centers of Research Excellence I-CORE program (center 4/11); the Israeli Ministry of Science and Technology; the Office of Naval Research under contract N00014-11-1-0470; the Simons Foundation, with a Simons Award for Graduate Students in Theoretical Computer Science; and the Skolkovo Foundation with agreement dated 10/26/2011.

The views expressed are those of the authors and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

Chapter 4

Bolt

This chapter is based on joint work with Matthew Green [87] which will appear in the 2017 ACM Conference on Computer and Communications Security.

4.1 Introduction

4.1.1 Background on Payment Channels

A payment channel is a relationship established between two participants in a privacy-preserving decentralized ledger-based currency network. While payments may flow in either direction on an established channel, the parties themselves are not symmetric: for a payment channel to work, at least one party must initiate the connection. For simplicity of exposition, we will refer to the initiating party as a *customer*, and the responding party as a *merchant*. We assume that the payment network includes a means to validate published transactions and to resolve disputes according to public rules. In principle these requirements can be satisfied by the scripting systems of consensus networks such as Monero or ZCash, using only minimal script extensions (which we discuss in §4.5.) We stress that our proposals in this work focus on the privacy of payment channels, and thus we assume the privacy of the underlying funding network.

When two parties wish to open a channel, the parties first agree on the respective balance shares of the channel, which we represent by non-negative integers B_0^{merch} and B_0^{cust}. The parties establish the channel by posting a payment to the network. Provided that these transactions are correctly structured, the network places the submitted funds in "escrow" until a subsequent closure transaction is received. The customer now conducts payments by interacting off-chain with the merchant. For some positive or negative integer payment amount ϵ_i, the i^{th} payment can be viewed as a request to update $B_i^{\mathsf{cust}} := B_{i-1}^{\mathsf{cust}} - \epsilon_i$ and $B_i^{\mathsf{merch}} := B_{i-1}^{\mathsf{merch}} + \epsilon_i$, with the sole restriction that $B_i^{\mathsf{merch}} \geq 0$ and $B_i^{\mathsf{cust}} \geq 0$. At any point, one or both parties may request to close the channel by posting a channel closure message to the ledger. If the closure messages indicate that the parties disagree about the current state of the channel, the ledger executes a dispute resolution algorithm to determine the final channel balances. After a delay sufficient to ensure each party has had an opportunity to contribute its closure message, the parties may recover their final shares of the channel balance using an on-chain payment transaction.

Any payment channel must meet two specific requirements, which we refer to as *universal arbitration* and *succinctness*:

1. **Universal arbitration.** In the event that two parties disagree about the state of a shared channel, the payment network can reliably arbitrate the dispute without requiring any private information.

2. **Succinctness.** To make payments scalable, all information posted to the ledger must be compact — *i.e.,* the size of this data should not grow linearly with the balance of the channel, the number of transactions or the amounts exchanged.

The latter property is an essential requirement for the setting of payment channels, since it rules out degenerate solutions that result in a posted transaction for every offline payment, or that post the full off-chain payment interaction to the ledger.

4.1.2 Customers, Merchants, and the Limits of Anonymity for Payment Channels

Informally our constructions for payment channels provide the following privacy guarantee:

> *Upon receiving a payment from some customer, the merchant learns no information beyond the fact that a valid payment (of some known positive or negative value) has occurred on a channel that is open with them. The network learns only that a channel of some balance has been opened or closed.*

Note, however, that the privacy protections against a channel participant are slightly weaker than those against third parties. This is an inherent limitation of the payment channel setting. Moreover, these limitations change depending on if a payment is made over a single direct channel or an indirect channel consisting of a series channels between the customer, one or more intermediaries, and a merchant. We explain these limitations further here.

Direct channels. The direct channel setting has three limitations. First, the privacy provided for direct channels is asymmetric: only the party initiating the payment is anonymous and unlinkable between payments, while the target of the payment is pseudonymous. This holds because at least one party must know which payment channel is being used.

Second, when receiving a payment on a channel, the recipient knows the payment came from someone with whom they have an open channel. This is also fundamental to the nature of channels, since they must be established with a counter-party before being used.

This final requirement suggests that recipients should use well-known channel parameters to group all channels, and thus maximize the anonymity set of its customers. If a recipient provides unique channel parameters to each potential payer (*i.e.,* behaves as though it was a different party to each payer), than the payer receives no privacy — as the set of channels open under that set of parameters has an anonymity set of a single person.

This setting maps well to a situations where the payment target is known, *e.g.,* where a single merchant or website accepts payments from many anonymous customers. Thus for the remainder of the paper we term this well-known target party the *merchant*, and refer to the paying

party as the *customer*. We keep this terminology even when in settings where payment amounts are negative (resulting in a reverse payment), since one party must still be well known and this terminology maps to the case where, e.g., a merchant is refunding a customer for a previous purchase. We stress that to anyone not a party to the payment channel, privacy is absolute.

Indirect channels For indirect channels which involve one or more intermediate channels, the privacy guarantees may, surprisingly, be stronger than the direct case. First, this configuration facilitates a larger anonymity set, since it encompasses any party who has a channel open with the entry intermediary (for the initiator) as well as anyone who has a channel open with the exit intermediary (for the target). Additionally, when channels contain a single intermediary, they can be configured such that the merchant remains anonymous to the customer. Specifically, although the underlying pair-wise channels still offer asymmetric privacy (*i.e.,* one party is well-known), we can arrange the indirect channels so that the customer and merchant are both holding the private end of their channel and instead the intermediary is the only well-known party. We discuss this arrangement in §4.4.3.

The advantages of intermediaries do not fully generalize to chains containing more than a single intermediary. Specifically, we show that channels with a single intermediary can be configured to hide the payment amount from the intermediary. However, channels which involve more than one intermediary cannot hide the value of a payment from all intermediaries. Regardless of cryptographic underpinnings, at least one endpoint[1] of each channel must know the channel balance or else the channel cannot be closed. As a result, in any chain of channels with multiple intermediaries, at least one channel will have an intermediary party on both endpoints, and one of these parties will inevitably learn the value of the payments. This is not a limitation of our techniques but simply a consequence of the nature of payment channels.

[1]It is possible that neither endpoint knows the balance in full and instead must cooperate to learn it. This does not alter the problem.

4.1.3 Overview of our constructions

In this work we investigate two separate paradigms for constructing anonymous payment channels. Our first construction builds on the electronic cash, or e-cash, paradigm first introduced by Chaum [11] and extended in many subsequent works, *e.g.,* [88, 89, 90]. This unidirectional construction allows for succinct payments of fixed-value tokens from a customer to a merchant, while preserving the anonymity and functionality of a traditional payment channel. Our second construction extends these ideas to allow for variable-valued payments that traverse the channel in either direction (*i.e.,* each payment may have positive or negative value), at the cost of a more complex abort condition. Finally, we show how to extend our second construction to support path payments where users pay anonymously via a single untrusted intermediate party or a chain of intermediaries.

We now present the intuition behind our constructions.

Unidirectional payment channels from e-cash. An e-cash scheme is a specialized protocol in which a trusted party known as a *bank* issues one-time tokens (called *coins*) that customers can redeem exactly one time. "Offline" e-cash protocols seem like a natural candidate for implementing a one-way payment channel. For purposes of exposition, let us first consider a "strawman" proposal based on some ideal offline e-cash scheme that allows for the detection of doubly-spent coins. In this proposal, the merchant plays the role of the bank. After confirming that the customer has funded a channel, it issues a "wallet" of anonymous coins to the customer, who then spends them back to the merchant. To close the channel, the customer spends the remaining coins to herself and posts the evidence to the payment network. The merchant can dispute the customer's statement by providing evidence of a doubly-spent coin.

This strawman protocol suffers from several weaknesses. Most obviously, it is not *succinct*, since closure requires the customer to post all of her unspent coins. Secondly, there is an issue of timing: the merchant cannot issue a wallet to the customer until the customer's funds have been

escrowed by the network, a process that can take from minutes to hours. At the same time, the customer must be assured that she can recover her funds in the event that the merchant fails to issue her a wallet, or aborts during wallet activation. Finally, to avoid customer "framing" attacks (in which a merchant issues coins to itself and then accuses the customer of double-spending) we require an e-cash scheme with a specific property called *exculpability*: namely, it is possible for any third party (in our case the payment network) to distinguish "true" double spends — made by a cheating customer — from false double-spends created by the merchant.

Intuition behind our unidirectional construction. To address the first concern, we begin with a *compact* e-cash scheme [90]. Introduced by Camenisch *et al*, this is a form of e-cash in which B separate coins can be generated from a constant-sized wallet stored at the customer (here B is polynomial in the wallet size). While compact e-cash reduces the wallet storage cost, it does not immediately give rise to a succinct closure mechanism for our channels. The key innovation in our construction is a new mechanism that reduces channel closure to a single fixed-size message — at the cost of some increased (off-chain) interaction between the merchant and customer.

To create a payment channel in our construction, the customer first commits to a set of secrets used to formulate the wallet. These are embedded within a succinct *wallet commitment* that the customer transmits to the payment network along with the customer's escrow funds (and an ephemeral public signature verification key pk_c). The customer and merchant now engage in an interactive channel establishment protocol that operates as follows. The customer first generates B coin spend transactions, and attaches to each a non-interactive zero knowledge proof that each coin is tied to the wallet commitment. She then individually encrypts each of the resulting transactions using a symmetric encryption scheme such that each ciphertext C_i embeds a single spend transaction, along with the decryption key for ciphertext C_{i+1}. After individually signing each of the resulting ciphertexts using her secret key, the customer transmits the signed results to the merchant for safekeeping. A critical aspect of this scheme is that from the merchant's perspective these ciphertexts are opaque: the customer does not need to prove to the merchant that any ciphertext is well-formed.

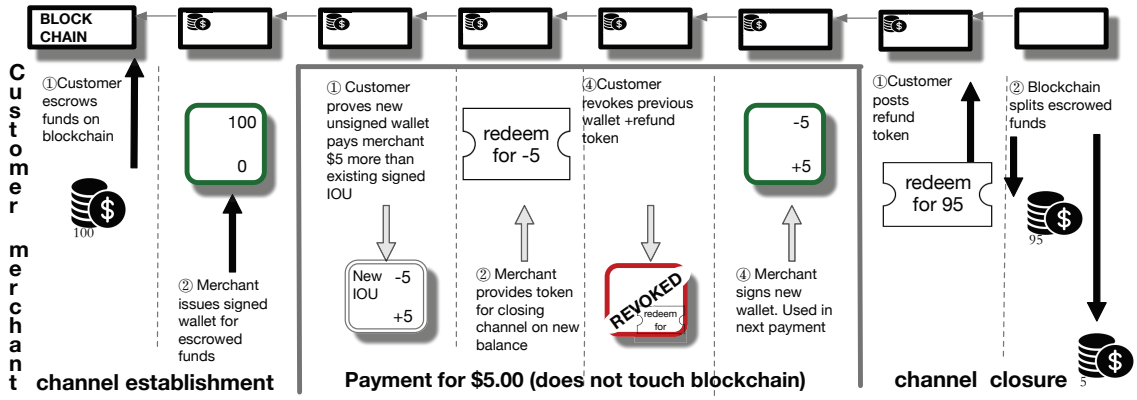


Figure 4.1: High level description of bidirectional channel protocol. The customer is the anonymous party. The merchant is a known identity. Only channel establishment and closure touch the blockchain.

When the customer wishes to close an active channel with remaining balance N (for $0 < N \leq B$), she computes $j = (B - N) + 1$ and posts a signed message $(\mathsf{channel\ ID}, j, k_j)$ to the network, with k_j being the decryption key for the j^{th} ciphertext. The merchant can use this tuple to decrypt each of the ciphertexts $C_j, \ldots, C_N$ and thus detect further spending on the channel. If the customer cheats by revealing an invalid decryption key, or if any ciphertext decrypts to an invalid coin, or if the resulting transactions indicate that she has double-spent any coin, the merchant can post indisputable evidence of this cheating to the network — which, to punish the customer, will grant the full channel balance to the merchant.

Bidirectional payment channels. A restriction on the previous construction is that it is *unidirectional*: all payments must flow from the customer to the merchant. While this is sufficient for many useful applications — such as micropayments for web browsing — some applications of payment channels require payments to flow from the merchant to the customer. As we further discuss below, a notable example of such an application is *third party payments*, where two parties send funds via an intermediary, who must increase the value of one channel while decreasing the other.

For these applications, we propose a second construction that combines techniques from existing (non-anonymous) payment channels with blind signatures and efficient zero-knowledge proofs.

As in the existing payment channel systems [91, 8], the customer and merchant first on agree on an initial channel state, with the customer holding B_0^{cust} escrowed funds, and the merchant provides a signature on this balance. When the customer wishes to pay the merchant an arbitrary positive or negative amount ϵ, she conducts an interactive protocol to (1) prove knowledge of the previous signature on the current balance B_{i-1}^{cust}, and (2) demonstrate that she possesses sufficient balance to complete the payment. She then (3) blindly extracts a new signed *refund token* from the merchant containing the updated balance $B_i^{\mathsf{cust}} = B_{i-1}^{\mathsf{cust}} - \epsilon$. At any point, the customer may post her most recent refund token to the blockchain to redeem her available funds. See figure 4.1.

The main challenge in this approach is to prevent a dishonest customer from retaining and using earlier versions of her refund token on channel closure. To prevent this, during each payment, the customer interacts with the merchant to present a *revocation token* for the previous state. As long as the customer behaves honestly, this revocation token can never be linked to the channel or to any previous transactions. However, if the customer misbehaves by posting an obsolete refund token, the merchant can instantly detect this condition and present the revocation token to the network as proof of the customer's malfeasance – in which case, the network awards the balance of the channel to the merchant. Unlike the e-cash approach, this proposal suffers from the possibility that one of the parties will *abort* the protocol early; we address this by using the network to enforce fairness.

From direct to third-party payments. As the concluding element of our work, we show how a bidirectional payment channel can be used to construct *third-party* payments, in which a first party **A** pays a second party **B** via a common, untrusted intermediary **I** to which both parties have previously established a channel. In practice, this capability eliminates the need for parties to maintain channels with all of their peers. The key advantage of our proposal is that the intermediary **I** cannot link transactions to individual users, nor — surprisingly — can they learn the amount being paid in a given transaction. Similarly, even if **I** is compromised, it cannot claim any transactions passing through it. This technique makes anonymous payment channels usable in practice, provided there exists a highly-available (untrusted) intermediary to route the connections. We provide the full

details of our construction and how to extend it to support multiple intermediaries in §4.4.3.

Aborts. Our unidirectional protocol provides privacy guarantees that are similar to the underlying e-cash protocol, with the obvious (and necessary) limitation that final channel balances are revealed on closure. Payments between a customer and merchant are non-interactive and completely anonymous. The bidirectional payment construction, on the other hand, provides a slightly weaker guarantee: by aborting during protocol execution, the merchant can place the customer in a state where she is unable to conduct future transactions. This does not prevent the customer from resorting to the network to close the channel, but it does raise concerns for anonymity in two ways:

1. The merchant can arbitrarily reduce the anonymity set by (even temporarily) evicting other users through induced aborts.

2. The merchant may link a user to a repeating sequence of transactions by aborting the user in the middle of the sequence.

For many traditional commerce settings, the consequences of such aborts may be minimal: no matter the payment mechanism, the merchant can fail to deliver the promised goods and the customer will almost certainly abort. For other settings, such as micropayments, these possibilities should be considered. In such settings customers should scan the network for premature closures and abort the channel if the number of open channels with a merchant falls below their minimal anonymity set.

4.1.4 Comparison to related work

In concurrent work, Heilman *et al.* proposed an elegant mixing system called Tumblebit [92]. Tumblebit is compatible with classical Bitcoin and operates in two modes. The first allows users to anonymize (aka mix or "launder") their own coins. The second mode allows for payment channels between distinct users. Overcoming the limited choice of cryptographic primitives to get Bitcoin compatibility is a serious achievement, but for Tumblebit it comes at the cost of far more limited features, performance, and privacy in comparison to Bolt's payment channels.

Most significantly, in a payment from Alice to Bob, "Bob and the Tumbler can collude to learn the true identity of Alice" [92] since Alice identifies herself to the Tumbler when making a payment because she must pay the Tumbler with traceable Bitcoins.[2] In contrast both our schemes provide provable privacy for Alice even in the face of corrupted and colluding parties. Second, Tumblebit does not hide payment values.

On the functionality side: Tumblebit payments are of a single fixed value and payment channels are unidirectional. In contrast we provide for bidirectional payment channels with variable valued payments. Tumblebit payments are also not succinct: a channel allowing n payment needs either $O(n)$ state on the blockchain or $O(n^2)$ invocations of their protocol.

On the performance side: at $387ms$ per channel payment, Tumblebit is 5 times slower than our prototype implementation of Bolt's bidirectional channels. We stress that this is not due to a design flaw in Tumblebit: working within the confines of Bitcoin compatibility is extremely challenging and comes at a high cost.

Finally, like Tumblebit, our unidirectional protocol provides full protections from aborts. Our bidirectional protocol does not and requires an underlying anonymous currency for safety (see §4.1.3). Variable payments seem to require multiple rounds of interaction, thus risking aborts terminating in invalid intermediate states.

4.1.5 Outline of this paper

The remainder of this paper proceeds as follows. In §2.2 we present definitions for anonymous payment channels. In §4.3 we present the building blocks of our scheme. In §4.4 we describe the protocols for our payment channel constructions, and in §4.5 we present concrete instantiations of these protocols. Finally, in §4.6 we discuss the related work.

[2]This is likely fundamental. See Section 7.c of [92] for further discussion. Although Heilman et al. provide some mitigations for these attacks, as they acknowledge the Tumbler and Bob can still correlate Alice's interactions. Thus they cannot offer Alice provable privacy from Bob.

4.2 Definitions

Notation: Let λ be a security parameter. We write $P(\mathcal{A}(a), \mathcal{B}(b)) \to (c, d)$ to indicate a protocol P run between parties $\mathcal{A}$ and $\mathcal{B}$, where a is $\mathcal{A}$'s input, c is $\mathcal{A}$'s output, b is $\mathcal{B}$'s input and d is $\mathcal{B}$'s output. We will define $\mathrm{negl}(\cdot)$ as a negligible function. We will use $\mathsf{val}_{\mathsf{max}}$ to denote the maximum balance of a payment channel, and denote by the set of integers $\{\epsilon_{\mathsf{min}}, \ldots, \epsilon_{\mathsf{max}}\}$ the range of valid payment amounts.

4.2.1 Anonymous Payment Channels

An Anonymous Payment Channel (APC) is a construct established between two parties that interact via a payment network. In this section we first describe the properties of an *anonymous payment channel scheme*, which is a collection of algorithms and protocols used to establish these channels. We then explain how these schemes can be used to construct channels in a payment network. We now provide a formal definition of an APC scheme.

Definition 4.2.1 (APC scheme)**.** An anonymous payment channel scheme consists of a tuple of possibly probabilistic algorithms $(\mathsf{KeyGen}, \mathsf{Init}_{\mathcal{C}}, \mathsf{Init}_{\mathcal{M}}, \mathsf{Refund}, \mathsf{Refute}, \mathsf{Resolve})$ and two interactive protocols $(\mathsf{Establish}, \mathsf{Pay})$. These are defined in Figure 4.2. For completeness we also define an optional function $\mathsf{Setup}(1^\lambda)$ to be run by a trusted party for generating the parameters pp, *e.g.,* a Common Reference String. In some instantiations the CRS is not required. In this case, we set $\mathsf{pp} := 1^\lambda$.[3]

Using Anonymous Payment Channels. An anonymous payment channel scheme must be used in combination with a payment network capable of conditionally escrowing funds and binding these escrow transactions funds to some data. Such payment networks can be constructed using blockchain-based systems, although they can be built from other technology as well. In this work we assume only the existence of a payment network with these capabilities, and leave the details of the payment

[3]Looking forward to our recommended instantiations in §4.5, we propose to use a CRS based on public randomness.

Key generation and channel initialization algorithms:

$\mathsf{KeyGen}(\mathsf{pp})$. This algorithm generates a keypair (pk, sk) for use by each customer or merchant.

$\mathsf{Init}_{\mathsf{P}}(\mathsf{pp}, B_0^{\mathsf{cust}}, B_0^{\mathsf{merch}}, pk, sk)$. For $\mathcal{P} \in \{\mathcal{C}, \mathcal{M}\}$ this algorithm is run by each party prior to opening a channel. On input the initial channel balances, public parameters and the party's keypair, the $\mathsf{Init}_{\mathcal{C}}$ algorithm outputs the party's channel token $\mathsf{T}_{\mathcal{P}}$ and a corresponding secret $csk_{\mathcal{P}}$.

Two-party protocols run between a customer $\mathcal{C}$ and a merchant $\mathcal{M}$:

$\mathsf{Establish}(\{\mathcal{C}(\mathsf{pp}, \mathsf{T}_{\mathcal{M}}, csk_{\mathcal{C}})\}, \{\mathcal{M}((\mathsf{pp}, \mathsf{T}_{\mathcal{C}}, csk_{\mathcal{M}})\}$. On input public parameters and each of the initial channel tokens, the $\mathsf{Establish}$ protocol activates a channel between two parties who have previously escrowed funds. If successful, the merchant receives $\mathsf{established}$ and the customer receives a wallet w. Either party may receive the distinguished failure symbol $\perp$.

$\mathsf{Pay}(\{\mathcal{C}(\mathsf{pp}, \epsilon, w_{\mathsf{old}})\}, \{\mathcal{M}(\mathsf{pp}, \epsilon, \mathbf{S}_{\mathsf{old}})\})$. On input parameters, a payment amount ϵ, and a wallet w_{old} from a customer, and the merchant's current state $\mathbf{S}_{\mathsf{old}}$ (initially $\emptyset$) from the merchant: the customer receives a payment success bit $R_{\mathcal{C}}$ and new wallet w_{new} if the interaction succeeded. The merchant receives a payment success bit $R_{\mathcal{M}}$ and an updated state $\mathbf{S}_{\mathsf{new}}$ if the interaction succeeded.

Channel closure and dispute algorithms, run by the customer and merchant respectively:

$\mathsf{Refund}(\mathsf{pp}, \mathsf{T}_{\mathcal{M}}, csk_{\mathcal{C}}, w)$. On input a wallet w, outputs a customer channel closure message $\mathsf{rc}_{\mathcal{C}}$.

$\mathsf{Refute}(\mathsf{pp}, \mathsf{T}_{\mathcal{C}}, \mathbf{S}, \mathsf{rc}_{\mathcal{C}})$. On input the merchant's current state $\mathbf{S}_{\mathsf{old}}$ and a customer channel closure message, outputs a merchant channel closure message $\mathsf{rc}_{\mathcal{M}}$ and an updated merchant state $\mathbf{S}_{\mathsf{new}}$.

Dispute resolution algorithm, run by the network:

$\mathsf{Resolve}(\mathsf{pp}, \mathsf{T}_{\mathcal{C}}, \mathsf{T}_{\mathcal{M}}, \mathsf{rc}_{\mathcal{C}}, \mathsf{rc}_{\mathcal{M}})$. On input the customer and merchant's channel tokens $\mathsf{T}_{\mathcal{C}}, \mathsf{T}_{\mathcal{M}}$, along with closure messages $\mathsf{rc}_{\mathcal{C}}$, $\mathsf{rc}_{\mathcal{M}}$ (where either message may be null), this algorithm outputs the final channel balance $B_{\mathsf{final}}^{\mathsf{merch}}, B_{\mathsf{final}}^{\mathsf{cust}}$.

Figure 4.2: Definition of an Anonymous Payment Channel scheme.

network's implementation (*e.g.,* modeling a blockchain) as a separate problem. We now describe how these algorithms and protocols are used to establish a channel on a payment network.

To instantiate an anonymous payment channel, the merchant $\mathcal{M}$ first generates a long-lived keypair $(pk_{\mathcal{M}}, sk_{\mathcal{M}}) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$ that will identify it to all customers. The merchant initializes its state $\mathbf{S} \leftarrow \emptyset$. A customer $\mathcal{C}$ generates an ephemeral keypair $(pk_{\mathcal{C}}, sk_{\mathcal{C}})$ for use on a single channel. The customer and merchant agree on their respective initial channel balances $B_0^{\mathsf{cust}}, B_0^{\mathsf{merch}}$. They now perform the following steps:

1. Each party executes the $\mathsf{Init}_{\mathcal{C}}$ algorithm on the agreed initial channel balances, in order to derive the channel tokens $\mathsf{T}_{\mathcal{C}}, \mathsf{T}_{\mathcal{M}}$.

2. The two parties transmit these tokens to the payment network along with a transaction to escrow the appropriate funds.

3. Once the funds have been verifiably escrowed, the two parties run the $\mathsf{Establish}$ protocol to

activate the payment channel. If the parties disagree about the initial channel balances, this protocol returns $\perp$ and the parties may close the channel.

4. If channel establishment succeeds, the customer initiates the Pay protocol as many times as desired, until one or both parties close the channel.

5. If the customer wishes to close the channel, she runs Refund and transmits $\mathsf{rc}_{\mathcal{C}}$ along with the channel identifier to the payment network.[4]

6. The merchant runs Refute on the customer's closure token to obtain the merchant closure token $\mathsf{rc}_{\mathcal{M}}$.

At the conclusion of this process, the network runs the $\mathsf{Resolve}$ algorithm to determine the final channel balance and allows each party to collect the determined share of the escrowed funds.

4.2.2 Correctness and Security

We now described the correctness and security of an anonymous payment channel scheme. Here we provide intuition, and present formal definitions in Appendix C.2.

Correctness. Informally, an APC scheme is correct if for all correctly-generated parameters pp and opening balances $B_0^{\mathsf{cust}}, B_0^{\mathsf{merch}} \in \{0, \ldots, \mathsf{val}_{\mathsf{max}}\}$, every correct (and honest) interaction following the paradigm described above always produces a correct outcome. Namely, each valid execution of the Pay protocol produces success, and the final outcome of Refute correctly reflects the final channel balance.

Security. The security of an Anonymous Payment Channel scheme is defined in terms of two games, which we refer to as *payment anonymity* and *balance*. We now provide an informal description of each property, and refer the reader to Appendix C.2 for the formal definitions.

[4]Here we assume that channel closure is initiated by the customer. In cases where the merchant wishes to initiate channel closure, it may transmit a special message to the network requesting that the customer close the channel.

Payment anonymity. Intuitively, we require that the merchant, even in collaboration with a set of malicious customers, learns nothing about a customer's spending pattern *beyond* the information that is available outside of the protocol. In our anonymity definition, which extends a definition of Camenisch *et al.* [90], the merchant interacts with either (1) a series of oracles implementing the real world protocols for customers $\mathcal{C}_1, \ldots, \mathcal{C}_N$, or (2) with a simulator $\mathcal{S}$ that performs the customer's part of the Pay protocol. In the latter experiment, we assume a simulator that has access to side information not normally available to participants in the real protocol, *e.g.,* a simulation trapdoor or control of a random oracle. We require that the simulator has the ability to simulate any customer without access to the customer's wallet, and without knowing the identity of the customer being simulated. Our definition holds if no adversary can determine whether she is in world (1) or (2). We stress that this definition implies anonymity because the simulator has no information about which party it is simulating.

Balance. The balance property consists of two separate games, one for the merchant and one for the customer. In both cases, assuming honest execution of the Resolve protocol, this property ensures that no colluding set of adversarial counterparties can extract more value from a channel than justified by (1) the party's initial channel funding, combined with (2) the set of legitimate payments made to (or by) the adversary. Because the merchant and customer have different interfaces, we define this property in terms of two slightly different games. In each game, the adversarial customer (resp. merchant) is given access to oracles that play the role of the merchant (resp. customer), and allows the parties to establish an arbitrary number of channels with chosen initial balances. The adversary may then initiate (resp. cause the other party to initiate) the Pay protocol repeatedly on adversarially-chosen payment amounts ϵ. Finally, the adversary can initiate channel closure with the counterparty to obtain channel closure messages $\mathsf{rc}_{\mathcal{C}}$, $\mathsf{rc}_{\mathcal{M}}$. The adversary *wins* if the output of the Resolve protocol is inconsistent with the total value funded and paid.

4.3 Technical Preliminaries

In this section we recall some basic building blocks that we will use in our constructions.

Commitment schemes. Let $\Pi_{\mathsf{commit}} = (\mathsf{CSetup}, \mathsf{Commit}, \mathsf{Decommit})$ be a commitment scheme where CSetup generates public parameters; on input parameters, a message M, and random coins r, Commit outputs a commitment $\mathcal{C}$; and $\mathsf{Decommit}$ on input parameters and a tuple (C, m, r) outputs 1 if $\mathcal{C}$ is a valid commitment to the message, or 0 otherwise. In our instantiations, we recommend using the Pedersen commitment scheme [93] based on the discrete logarithm assumption in a cyclic group.

Symmetric encryption schemes. Our constructions require an efficient symmetric encryption scheme as well as a one-time symmetric encryption scheme. We define a symmetric encryption scheme $\Pi_{\mathsf{symenc}} = (\mathsf{SymKeyGen}, \mathsf{SymEnc}, \mathsf{SymDec})$ where $\mathsf{SymKeyGen}$ outputs an ℓ-bit key. We also make use of a one-time encryption scheme $\Pi_{\mathsf{otenc}} = (\mathsf{OTKeyGen}, \mathsf{OTEnc}, \mathsf{OTDec})$. In practice, the encryption scheme can be implemented by encoding the plaintext as an element in a cyclic group $\mathbb{G}$ and multiplying by a random group element. In either case, our constructions require that the schemes provide IND-CPA security.

Pseudorandom Functions. Our unidirectional construction requires a pseudorandom function (PRF) F that supports efficient proofs of knowledge. For our purposes it is sufficient that the PRF be secure for a poly-size input space. In addition to the standard pseudorandomness property, ourprotocols require that the PRF should also possess a property we refer to as *strong pre-image resistance*. This property holds that, given access to an oracle implementing the function $F_s(\cdot)$ for a random seed s, no adversary can find an input point x and a pair (s', x') in the domain of the function such that $F_s(x) = F_{s'}(x')$ except with negligible probability. We propose to instantiate F using the Dodis-Yampolskiy PRF [94], the public parameters are a group $\mathbb{G}$ of prime order q with generator g. The seed is a random value $s \in \mathbb{Z}_q$ and the function is computed as $f_s(x) = g^{1/(s+x)}$ for

x in a polynomially-sized set. We show in Appendix C.5 that the Dodis-Yampolskiy PRF satisfies the strong pre-image resistance property.

Signatures with Efficient Protocols. Our schemes make use of a signature scheme $\Pi_{\mathsf{sig}} = (\mathsf{SigKeygen}, \mathsf{Sign}, \mathsf{Verify})$ with efficient protocols, as proposed by Camenisch and Lysyanskaya [95]. These schemes feature: (1) a protocol for a user to obtain a signature on the value(s) in a commitment without the signer learning anything about the message(s), and (2) a protocol for (non-interactively) proving knowledge of a signature. Several instantiations of these signatures have been proposed in the literature, including constructions based on the Strong RSA assumption [95] and various assumptions in bilinear groups [96, 97]. For security, we assume that all signatures satisfy the property of *existential unforgeability under chosen message attack* (EU-CMA).

Non-Interactive Zero-Knowledge Proofs. We use several standard results for non-interactively proving statements about committed values, such as (1) a proof of knowledge of a committed value, and (2) a proof that a committed value is in a range. When referring to the proofs above, we will use the notation of Camenisch and Stadler [34]. For instance, $PoK\{(x, r) : y = g^x h^r \ \wedge \ (1 \leq x \leq n)\}$ denotes a zero-knowledge proof of knowledge of integers x and r such that $y = g^x h^r$ holds and $1 \leq x \leq n$. All values not in enclosed in ()'s are assumed to be known to the verifier. Our protocols require a proof system that provides *simulation extractability*, which implies that there exists an efficient proof extractor that (under specific circumstances, such as the use of a simulation CRS) can extract the witness used by an adversary to construct a proof, even when the adversary is also supplied with simulated proofs. In practice we can conduct these proofs non-interactively using a variety of efficient proof techniques [96, 17, 29, 13, 98, 99, 100, 101, 102].

4.4 Protocols

In this section we present our main contribution, which consists of three protocols for implementing anonymous payment channels. Our first protocol in §4.4.1 is a unidirectional payment

channel based on e-cash techniques. Our second construction in §4.4.2 allows for bidirectional payments, with a more complex protocol for handling aborts. Finally, in §4.4.3 we propose an approach for third-party payments, in which two parties transmit payment via an *intermediary*.

4.4.1 Unidirectional payment channels

Our first construction modifies the compact e-cash construction of Camenisch *et al.* [90] to achieve efficient and *succinct* unidirectional payment channels. We now provide a brief overview of this construction.

Compact e-cash. In a compact e-cash scheme, a customer withdraws a fixed-size wallet capable of generating B coins. The customer's wallet is based on a tuple (k, sk, B): k is an (interactively generated) seed for a pseudorandom function F, sk is the customer's private key, and B is the number of coins in the wallet. Once signed by the merchant, this wallet can be used to generate up to B coins as follows: the i^{th} coin consists of a tuple (s, T, π) where s is a "serial number" computed as $s = F_k(i)$; T is a "double spend tag" computed such that, if the same coin is spent twice, the double spend tags can be combined to reveal the customer's key pk (or sk); and π is a non-interactive zero-knowledge proof of the following statements:

1. $0 < i \leq B$
2. The prover knows sk.
3. The prover has a signature on the wallet (k, sk, B).
4. The pair (s, T) is correctly structured with respect to the signed wallet.

This construction ensures that double spending is immediately detected by a verifier, since both transactions will share the serial number s.[5] The verifier can then recover the spender's public key by combining the double-spend tags. At the same time, the individual coin spends cannot be linked

[5] In the original compact e-cash construction [90], the key k was generated using an interactive protocol between the customer and bank, such that honest behavior by one party ensured that k was uniformly random. In our revised protocol below, k will be chosen only by the customer. This does not enable double-spending, provided that the PRF is deterministic and the proof system is sound.

to each other or to the user. Camenisch *et al.* [90] show how to construct the proof π efficiently using signatures and proof techniques secure under the Strong RSA or bilinear assumptions in the random oracle model. Subsequent work presents efficient proofs in the standard model [96, 103].

Achieving succinct closure. Let us recall our intuition for using compact e-cash in a unidirectional payment channel (see §4.1.3). In this proposal, the merchant plays the role of the bank and issues the customer a wallet of B coins, which she can then (anonymously) spend back to the merchant. To close a channel, the customer simply spends any unused coins "to herself", thus proving to the merchant that she retains no spending capability on the channel (since any subsequent attempt to spend those coins would be recognized by the merchant as a double spend). Unfortunately while compact e-cash provides a succinct wallet, this does not immediately lead to a succinct protocol for closing the channel — as the customer cannot simply reveal the wallet secrets without compromising the anonymity of previous coins spent on the channel. We require a mechanism to succinctly reveal only a fraction of the coins in a wallet, without revealing them all. At the same time, we wish to avoid complex proofs (*e.g.,* a proving cost that scales with $O(B)$).[6]

Our approach is to use the merchant to store the necessary information to verify channel closure. This requires a number of changes to the compact e-cash scheme of Camenisch *et al.* [90] (requiring a fresh analysis of the scheme, which we provide in §4.4.1). First, we design the customer's $\mathsf{Init}_{\mathcal{C}}$ algorithm so that the PRF seed k is generated solely by the customer, rather than interactively by the customer and the bank (merchant) as in [90]. The customer now commits to the wallet secrets, producing wCom, and embeds this into the customer's channel token $\mathsf{T}_{\mathcal{C}} := (\mathsf{wCom}, pk_c)$ where pk_c is a signature verification key. During the $\mathsf{Establish}$ protocol to obtaining the merchant's signature on wCom, the customer provides the merchant with a series of signed ciphertexts $(C_1, \ldots, C_B)$, each of which contains a coin spend tuple of the form (s, T, π') where π' is identical to the normal compact

[6]Indeed, an alternative proposal is to construct the coin serial numbers using a chained construction, where each s_i is computed as a one-way hash of the key used in the previous transaction. This would allow the customer to revoke the channel by posting a secret from one transaction. Unfortunately, proving the correctness of s_i using standard zero-knowledge techniques would then require $O(B)$ proving cost, and moreover, does not seem easy to accomplish using the efficient zero knowledge proof techniques we recommend in this work.

e-cash proof, but simply proves that s, T are correct with respect to wCom (which is not yet signed by the merchant). These ciphertexts are structured so that a key revealed for the j^{th} ciphertext will also open each subsequent ciphertext.

The key feature of this approach is that the merchant *does not need to know if these ciphertexts truly contain valid proofs* at the time the channel is opened. To reveal the remaining j coins in a channel, the customer reveals a key for the j^{th} ciphertext, which allows the merchant to "unlock" all of the remaining coin spends and verify them with respect to the commitment wCom embedded in the customer's channel token. If any ciphertext fails to open, or if the enclosed proof is not valid, the merchant can easily prove malfeasance by the customer and obtain the balance of the channel. This requires only symmetric encryption and a means to "chain" symmetric encryption keys – both of which can easily be constructed from standard building blocks.[7] Our schemes additionally require a one-time encryption algorithm OTEnc where the keyspace of the algorithm is also the range of the pseudorandom function F.

We now present the full scheme:

$\mathsf{Setup}(1^\lambda)$. On input λ, optionally generate CRS parameters for (1) a secure commitment scheme and (2) a non-interactive zero knowledge proof system. Output these as pp.

$\mathsf{KeyGen}(\mathsf{pp})$. Compute $(pk, sk) \leftarrow \Pi_{\mathsf{sig}}.\mathsf{SigKeygen}(1^\lambda)$.[8]

$\mathsf{Init}_{\mathcal{C}}(\mathsf{pp}, B_0^{\mathsf{cust}}, B_0^{\mathsf{merch}}, pk_c, sk_c)$. On input a keypair (pk_c, sk_c), uniformly sample two distinct PRF seeds k_1, k_2 and random coins r for the commitment scheme. Compute $\mathsf{wCom} = \mathsf{Commit}(sk_c, k_1, k_2, B_0^{\mathsf{cust}}; r)$. For $i = 1$ to B, sample $ck_i \leftarrow \mathsf{SymKeyGen}(1^\lambda)$ to form the vector $\vec{ck}$. Output $\mathsf{T}_{\mathcal{C}} = (\mathsf{wCom}, pk_c)$ and $csk_{\mathcal{C}} = (sk_c, k_1, k_2, r, B_0^{\mathsf{cust}}, \vec{ck})$.

$\mathsf{Init}_{\mathcal{M}}(\mathsf{pp}, B_0^{\mathsf{cust}}, B_0^{\mathsf{merch}}, pk_m, sk_m)$. Output $\mathsf{T}_{\mathcal{M}} = pk_m$, $csk_{\mathcal{M}} = (sk_m, B_0^{\mathsf{cust}})$.

$\mathsf{Refund}(\mathsf{pp}, \mathsf{T}_{\mathcal{M}}, csk_{\mathcal{C}}, w)$. Parse w (generated by the $\mathsf{Establish}$ and Pay protocols) to obtain

[7] For example, the necessary properties can be achieved using a secure commitment scheme and any secure symmetric encryption mechanism.

[8] For simplicity of exposition, we assume that pk can be derived from sk

$\vec{ck}$ and the current coin index i. Compute $\sigma \leftarrow \mathsf{Sign}(sk_c, \mathsf{refund} \| \mathsf{cID} \| i \| ck_i)$ (where cID uniquely identifies the channel being closed) and output $\mathsf{rc}_{\mathcal{C}} := (\mathsf{cID}, i, ck_i, \sigma)$.

$\mathsf{Refute}(\mathsf{pp}, \mathsf{T}_{\mathcal{C}}, \mathbf{S}, \mathsf{rc}_{\mathcal{C}})$. Parse the customer's channel closure message $\mathsf{rc}_{\mathcal{C}}$ as $(\mathsf{cID}, i, ck_i, \sigma)$ and verify cID and the signature σ. If the signature verifies, then obtain the ciphertexts $C_i, \ldots, C_B$ stored after the $\mathsf{Establish}$ protocol. For $j = i$ to B, compute $(j \| s_j \| u_j \| \pi_j^r \| ck_j \| \hat{\sigma}_j) \leftarrow \mathsf{SymDec}(ck_j, C_j)$ and verify the signature $\hat{\sigma}_j$ and the proof π_j^r. If (1) the signature $\hat{\sigma}_j$ or the proof π_j^r fail to verify, (2) any ciphertext fails to decrypt correctly, or (3) any of the decrypted values (s_j, u_j) match a valid spend containing (s_j, t_j) in $\mathbf{S}$ where $\mathsf{OTDec}(u_j, t_j) = pk_c$: record the invalid result into $\mathsf{rc}_{\mathcal{M}}$ along with cID and sign the result using sk_m so that it can be verified by the network. Otherwise set $\mathsf{rc}_{\mathcal{M}} = (\mathsf{accept})$ and sign with sk_m. Finally for each valid C_j, set $\mathbf{S} \leftarrow \mathbf{S} \cup (s_j, t_b, \pi)$ and output $\mathbf{S}$ as the new merchant state.

$\mathsf{Resolve}(\mathsf{pp}, \mathsf{T}_{\mathcal{C}}, \mathsf{T}_{\mathcal{M}}, \mathsf{rc}_{\mathcal{C}}, \mathsf{rc}_{\mathcal{M}})$. Parse the customer and merchant closure messages and verify all signatures. If any fail to verify, grant the balance of the channel to the opposing party. If $\mathsf{rc}_{\mathcal{C}} = (N, sk_N, \sigma)$ and $\mathsf{rc}_{\mathcal{M}} = \mathsf{accept}$ then set $B_{\mathsf{final}}^{\mathsf{cust}}$ to $(B_0^{\mathsf{cust}} - N) + 1$. Otherwise, evaluate the merchant closure message to determine whether the customer misbehaved. If so, assign the merchant the full balance of the channel.

We present the $\mathsf{Establish}$ and Pay protocols in Figure 4.3.

Security Analysis

Theorem 4.4.1. *The unidirectional channel scheme satisfies the properties of* anonymity *and* balance *under the assumption that* (1) *F is pseudorandom and provides strong pre-image resistance,* (2) *the commitment scheme is secure,* (3) *the zero-knowledge system is sound and zero-knowledge,* (4) *the signature scheme is existentially unforgeable under chosen message attack and signature extraction is blind, and* (5) *the symmetric encryption and one-time encryption scheme are each IND-CPA secure.*

We present a proof of Theorem 4.4.1 in Appendix C.3.

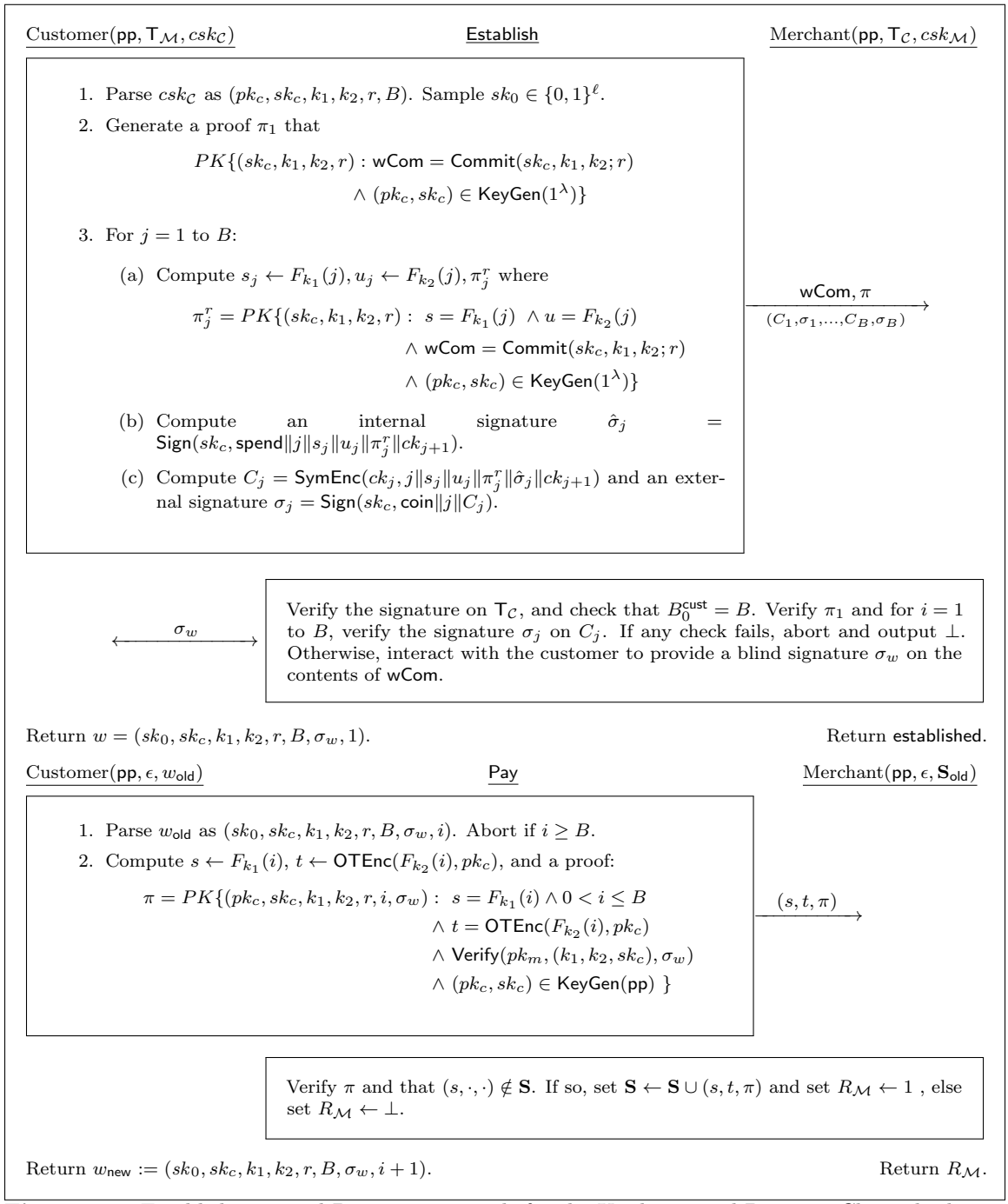


Figure 4.3: Establishment and Payment protocols for the Unidirectional Payment Channel scheme.

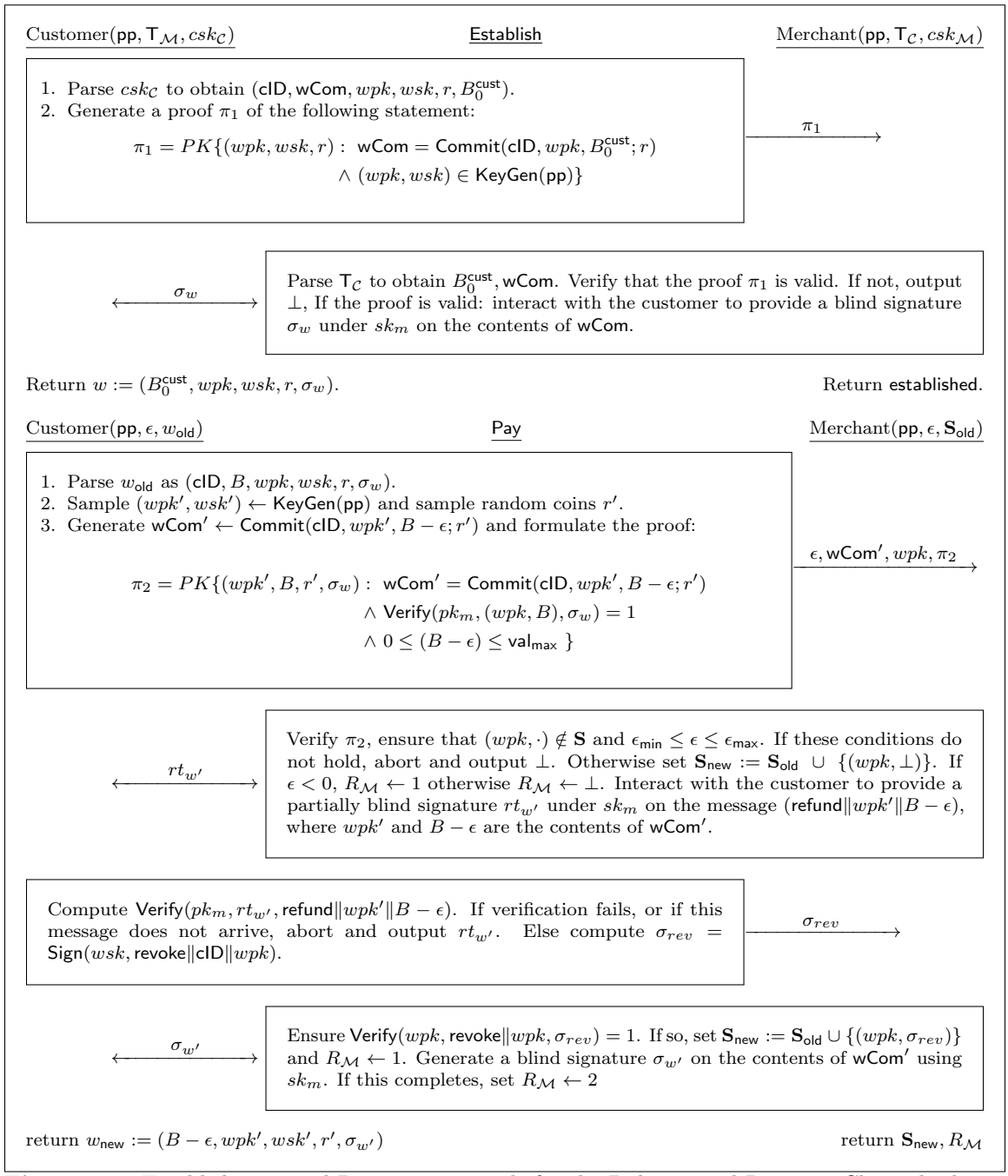


Figure 4.4: Establishment and Payment protocols for the Bidirectional Payment Channel scheme

4.4.2 Bidirectional payment channels

The key limitation of the above construction is that it is *unidirectional*, and only supports payments from a customer to a merchant. Additionally, it supports only fixed-value coins. In this

section we describe a construction that enables bidirectional payment channels which feature compact closure, compact wallets, and allow a single run of the Pay protocol to transfer arbitrary values (constrained by a maximum payment amount).

In this construction the customer's wallet is structured similarly to the previous construction: it consists of B_0^{cust}, and a random wallet public signature key wpk. The customer first commits to these values and sends the resulting commitment to the payment network. The wallet is activated when the customer and merchant interact to provide the customer with a blind signature on its contents.

The key difference from the first protocol is that, instead of conducting the payment ϵ using a series of individual coins, each payment has the customer (1) prove that it has a valid signed wallet with balance $B^{\mathsf{cust}} \geq \epsilon$ of currency in it, and (2) request a blind signature on a new wallet for the amount $B^{\mathsf{cust}} - \epsilon$ (and embedding a fresh wallet public key wpk_{new}). Notice that in this construction the value ϵ can be positive or negative. The customer uses a zero knowledge proof and signatures with efficient protocols to prove that the contents of the new requested wallet are constructed properly, that the balances of the new wallet differs from the original balance by ϵ, and that $(B^{\mathsf{cust}} - \epsilon) \geq 0$. At the conclusion of the transaction, the customer reveals wpk_{old} to assure the merchant that this wallet cannot be spent a second time, and the old wallet is invalidated by the customer signing a "revoked" message with wsk the corresponding private key. Closing the channel consists of the customer posting a valid wallet signed by the merchant to the blockchain.[9] The challenge in this construction is to simultaneously invalidate the existing wallet and sign the new one. If the merchant signs the new wallet before the old wallet is invalidated, then the customer can retain funds in the old wallet while continuing to use the new one. On the other hand, if the merchant can invalidate the old wallet before signing the new one, the customer has no way to close the channel if the merchant refuses to sign the new wallet.

To solve this, we separate the wallet — used in interactive payments — from the value that

[9] In the special case where the customer has not obtained a signature on the wallet from the merchant (*e.g.,* because the merchant never accepted the channel opening), it can simply post an opening of the wallet commitment.

is posted to perform channel closure and use a two phase protocol to obtain each of these values. Instead of revealing the most recent wallet w, $\mathcal{C}$ closes the channel using a *refund token* rt which specifies B^{cust}, the current wallet's public key, and a signature by the merchant. In phase one of Pay, the customer first obtains a signature on the refund token blindly from M. In the second phase, the customer invalidates the old wallet, and then the merchant signs the new wallet. If the merchant refuses to sign the new wallet, the customer can safely close the channel using rt.

We now describe the revised scheme. The protocols $\mathsf{Establish}$ and Pay are presented in Figure 4.4. The Setup and $\mathsf{Init}_{\mathcal{M}}$ algorithms are identical to the previous construction.

- $\mathsf{KeyGen}(\mathsf{pp})$. Compute $(pk, sk) \leftarrow \Pi_{\mathsf{sig}}.\mathsf{SigKeygen}(1^{\lambda})$.

- $\mathsf{Init}_{\mathcal{C}}(\mathsf{pp}, \mathsf{cID}, B_0^{\mathsf{cust}}, B_0^{\mathsf{merch}}, pk_c, sk_c)$. The customer generates the wallet commitment by sampling random coins r, computing an ephemeral keypair $(wpk, wsk) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$ and producing a commitment $\mathsf{wCom} = \mathsf{Commit}(\mathsf{cID}, wpk, B_0^{\mathsf{cust}}; r)$. It outputs the token $\mathsf{T}_{\mathcal{C}} = (pk_c, \mathsf{wCom})$ and retains the secrets $csk_{\mathcal{C}} = (\mathsf{wCom}, sk_c, \mathsf{cID}, wpk, wsk, r, B_0^{\mathsf{cust}})$.

- $\mathsf{Init}_{\mathcal{M}}(\mathsf{pp}, B_0^{\mathsf{cust}}, B_0^{\mathsf{merch}}, pk_m, sk_m)$. Output $\mathsf{T}_{\mathcal{M}} = pk_m$, $csk_{\mathcal{M}} = (sk_m, B_0^{\mathsf{cust}})$.

- $\mathsf{Refund}(\mathsf{pp}, \mathsf{T}_{\mathcal{M}}, csk_{\mathcal{C}}, w)$. If the customer has not yet invoked the Pay protocol, it sets $m := (\mathsf{refundUnsigned}, (\mathsf{cID}, wpk, B), r)$. Otherwise set $m := (\mathsf{refundToken}, (\mathsf{cID}, wpk, B), rt_w)$. Compute $\sigma = \mathsf{Sign}(sk_c, m)$. Output $\mathsf{rc}_{\mathcal{C}} = (m, \sigma)$.

- $\mathsf{Refute}(\mathsf{pp}, \mathsf{T}_{\mathcal{C}}, \mathbf{S}, \mathsf{rc}_{\mathcal{C}})$. If a merchant sees a channel closure message, it first parses $\mathsf{T}_{\mathcal{C}}$ to obtain pk_c. It parses $\mathsf{rc}_{\mathcal{C}}$ as (m, σ) and verifies the signature σ using pk_c. If this signature verifies, it parses m to obtain (cID, wpk, B) and verifies that cID matches the channel. Finally, if it has previously stored (wpk, σ_{rev}) in its state $\mathbf{S}$ then it outputs $\mathsf{rc}_{\mathcal{M}} = ((\mathsf{revoked}, \sigma_{rev}), \sigma)$ where σ is a valid signature on the message $(\mathsf{revoked}, \sigma_{rev})$ under sk_m. Otherwise it adds the new key wpk to its state $\mathbf{S}$.

- $\mathsf{Resolve}(\mathsf{pp}, \mathsf{T}_{\mathcal{C}}, \mathsf{T}_{\mathcal{M}}, \mathsf{rc}_{\mathcal{C}}, \mathsf{rc}_{\mathcal{M}})$.

Verify that both $\mathsf{rc}_{\mathcal{C}}, \mathsf{rc}_{\mathcal{M}}$ are correctly signed by the customer and merchant keys pk_c and pk_m respectively. Verify that both tokens contain the same cID and this matches the channel identifier from $\mathsf{T}_{\mathcal{C}}, \mathsf{T}_{\mathcal{M}}$. If either of the tokens fails this test, replace it with $\perp$. Let $B_{\mathsf{total}} = B_0^{\mathsf{cust}} + B_0^{\mathsf{merch}}$. If $\mathsf{rc}_{\mathcal{C}}$ is $\perp$, simply output all funds to the merchant.

1. Parse $\mathsf{T}_{\mathcal{C}}$ to obtain (pk_c, wCom).
2. Parse m as $(\mathsf{type}, (\mathsf{cID}, wpk, B), \mathsf{Token})$.
3. Parse m as $(\mathsf{revoked}, wpk, \sigma_{rev})$. Check that $\mathsf{Verify}(wpk, (\mathsf{revoke} \| \mathsf{cID} \| wpk)\sigma) = 1$. If any check fails, terminate and output $B_{\mathsf{final}}^{\mathsf{cust}} = B_{\mathsf{total}}$ and $B_{\mathsf{final}}^{\mathsf{merch}} = 0$.
4. Perform the following checks:
   (a) Check the refund's validity: If type is $\mathsf{refundUnsigned}$, check that $\mathsf{wCom} = \mathsf{Commit}(\mathsf{cID}, wpk, B, \mathsf{Token})$. If the merchant's token contains σ_{rev}
   Otherwise type is $\mathsf{refundToken}$, so check that Token is a valid refund token on (cID, wpk, B).
   If either check fails, terminate and output $B_{\mathsf{final}}^{\mathsf{cust}} = 0$ and $B_{\mathsf{final}}^{\mathsf{merch}} = B_{\mathsf{total}}$.
   (b) Check the refutation's validity: and check $\mathsf{Verify}(wpk, \mathsf{revoke} \| wpk, \sigma_{rev}) = 1$. If so, terminate and output $B_{\mathsf{final}}^{\mathsf{cust}} = 0$ and $B_{\mathsf{final}}^{\mathsf{merch}} = B_{\mathsf{total}}$. Otherwise return $B_{\mathsf{final}}^{\mathsf{cust}} = B$ and $B_{\mathsf{final}}^{\mathsf{merch}} = B_{\mathsf{total}} - B$ (i.e. pay the claimed B to $\mathcal{C}$ and the remainder to $\mathcal{M}$).

Security Analysis

As we noted in §4.1.3, the main limitation of the bidirectional protocol is the possibility that a malicious merchant may abort the protocol. The nature of the protocol ensures that a customer is not at risk of losing funds due to such an abort, since she may simply provide her refund token $rt_{w'}$ to the payment network in order to recover her balance. The main limitation therefore is to the customer's anonymity. A malicious merchant can place a customer into a situation where she cannot continue to spend, and must close her channel. This implicitly links the payment to the channel – a matter that is of only limited concern, if the channel is funded with anonymous currency.

Of more concern is the possibility that a malicious merchant will use aborts to reduce the anonymity set of the system, by causing several channels to enter a non-functional state. In practice, this attack will produce a visible signal at the payment network, allowing customers to halt use of the channel. However, within the context of our security proof we address this in a simpler way, by simply preventing the adversarial merchant from aborting during the Pay protocol.

Theorem 4.4.2. *The bidirectional channel scheme satisfies the properties of* anonymity *and* balance *under the restriction that the adversary does not abort during the* Pay *protocol, and the assumption that* (1) *the commitment scheme is secure,* (2) *the zero-knowledge system is simulation extractable and zero-knowledge,* (3) *the blind signature scheme is existentially unforgeable under chosen message attack, and* (4) *the one time signature scheme is existentially unforgeable under one time chosen message attack.*

We include a proof of Theorem 4.4.2 in Appendix C.4.

4.4.3 Bidirectional Third Party Payments

One of the main applications of the bidirectional construction above is to enable *third party payments*. In these payments, a first party **A** makes a payment of some positive value to a second party **B** via some intermediary **I** with whom both **A** and **B** have open channels. In this case, we assume that both **A** and **B** act as the customer for channel establishment, and **I** plays the role of the merchant. Our goal is that **I** does not learn the identities of the participants, or the amount being transferred (outside of side information she can learn from her channel state), nor should she trusted to safeguard the participants' funds. This construction stands in contrast to existing non-anonymous payment channel schemes [91, 8] where given the chain $\mathbf{A} \rightarrow \mathbf{I} \rightarrow \mathbf{B}$, the intermediary always learns both the amount and the participants.

The challenge in chaining payment channels is to make the payments *atomic*. That is, the payer **A** only wants to pay the intermediary **I** once **I** has paid the recipient **B**. But of course this places the intermediary at risk if **A** fails to complete the payment. Similarly, the payer risks losing

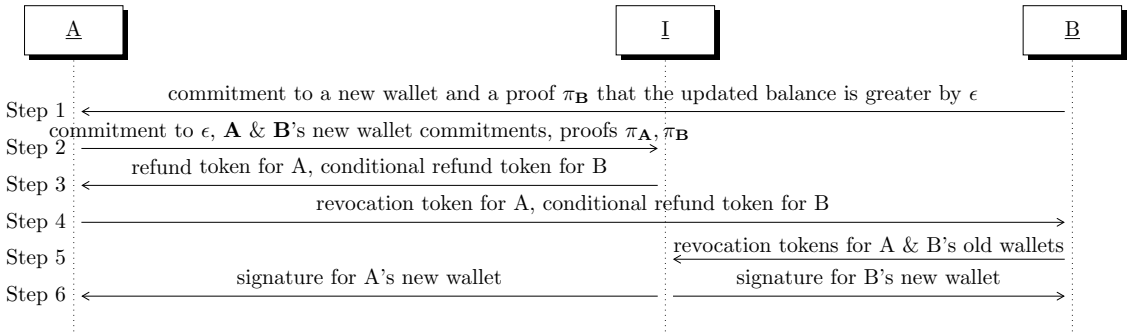


Figure 4.5: Outline of our third-party payments protocol. In practice, **A** can route all messages from **B** to **I**.

her funds to a dishonest intermediary. There is no purely cryptographic solution to this problem, since it's in essence fair exchange — a problem that has been studied extensively in multi-party protocols. However, there are known techniques for using blockchains to mediate aborts [104, 105]. This is our approach as well.

Recall from §4.4.2 that the Pay protocol occurs in two phases. The first portion is an exchange of *refund tokens* that can be used to reclaim escrowed funds. The second phase generates an anonymous wallet for subsequent payments. For a chained transaction from $\mathbf{A} \to \mathbf{I} \to \mathbf{B}$ to be secure, we need only ensure that the first phase of both legs completes or fails atomically.

We accomplish this by adding conditions to the refund tokens. These conditions are simple statements for the network to evaluate on examining a token during the Resolve protocol. Specifically, to prevent the recipient **B** from claiming funds from **I** if the payer **A** has not delivered a corresponding payment to **I**, we introduce the following conditions into **B**'s refund token:

1. **B** must produce a revocation message (*i.e.* a signature using **A**'s wsk) on **A**'s previous wallet.

2. **A** has not posted a revocation token containing wsk to the ledger.

By condition (1), once **B** forces a payment on $\mathbf{I} \to \mathbf{B}$, $\mathbf{A} \to \mathbf{I}$ cannot be reversed since **I** has the revocation token. By condition (2) if $\mathbf{A} \to \mathbf{I}$ has been already been reversed, **B** cannot force the payment $\mathbf{I} \to \mathbf{B}$ since wpk is already on the ledger.

Hiding the payment amount. Our third-party payment construction also provides an additional

useful feature. Since **I** acts only a passive participant in the transaction and does not maintain state for either channel, there is no need for for **I** to learn the amount being paid. Provided that both **A** and **B** agree on an amount ϵ (*i.e.,* both parties have sufficient funds in each of their channels), neither party need reveal ϵ to **I**: **I** need merely be assured that the balance of funds is conserved.

To hide the payment amount, we must modify the proof statement used to construct π_2 from the Pay protocol of Figure 4.4. Rather than revealing ϵ to the merchant, the customer **A** now commits to ϵ and uses this value as a secret input in computing the payment. Simultaneously, in the payment protocol conducted to adjust **B**'s wallet, **B** now proves that his wallet has been adjusted by $-\epsilon$.

To do this, we change the proof in the pay protocol to one that binds ϵ to a commitment but does not reveal it:

$$\begin{aligned}\pi_2 = PK\{(wpk', B, r', \sigma_w, \epsilon, r_\epsilon) : \\ \mathsf{wCom}' = \ & \mathsf{Commit}(wpk', B-\epsilon; r') \\ & \wedge \mathsf{Verify}(pk_m, (wpk, B), \sigma_w) = 1 \\ & \wedge \mathsf{vCom} = \mathsf{Commit}(\epsilon, r_\epsilon) \\ & \wedge 0 \leq (B-\epsilon) \leq \mathsf{val}_{\mathsf{max}} \ \}\end{aligned}$$

A can then prove to **I** that the two payments cancel or (if fee is non-zero), leave B with fee extra funds via a proof:

$$\begin{aligned}\pi_\epsilon = PK\{(\epsilon_{\mathbf{A}}, \epsilon_{\mathbf{B}}, r_{\epsilon_{\mathbf{A}}}, r_{\epsilon_{\mathbf{B}}}) : & \mathsf{vCom}_{\epsilon_A} = \mathsf{Commit}(\epsilon_{\mathbf{A}}; r_{\epsilon_{\mathbf{A}}}) \\ & \wedge \mathsf{vCom}_{\epsilon_{\mathbf{B}}} = \mathsf{Commit}(\epsilon_{\mathbf{B}}; r_{\epsilon_{\mathbf{B}}}) \\ & \wedge \epsilon_{\mathbf{A}} < \epsilon_{\mathsf{max}} \wedge -\epsilon_{\mathbf{B}} < \epsilon_{\mathsf{max}} \\ & \wedge \epsilon_{\mathbf{A}} + \epsilon_{\mathbf{B}} = fee\end{aligned}$$

The protocol. We now combine the process of updating both **A** and **B**'s wallet into a single protocol flow, which we outline in Figure 4.5. In detail, the steps are as follows:

1. **B** commits to ϵ and conducts the first move of the variable payment Pay protocol (Figure 4.4) (with the modified balance-hiding proof described above) and sends a commitment to its new wallet state wCom'_b, proof of correctness for the wallet, $\pi_{\mathbf{B}}$, and commitment randomness to **A**.

2. **A** completes it's own first move, generating $\mathsf{wCom}'_a, \pi_{\mathbf{A}}$ and additionally computes $\pi_{\mathbf{A}}$ attesting to the correct state of its original wallet and new wallet commitment. It sends these and **B**'s new wallet commitment and $\pi_{\mathbf{A}}$ to **I**.

3. **I**, after validating the proofs, issues **A** a refund token for its new wallet $rt_{w'_a}$ and **B** a conditional refund token $crt_{w'_b}^{\sigma_{rev}^{w_a}}$ as its new wallet. This token embeds the condition that **B** must produce a revocation token for **A**'s old wallet and that **A** must not have closed the channel already.

4. **A** completes its second move in the variable payment Pay protocol to generate $\sigma_{rev}^{w_a}$ the revocation token for its old wallet. It sends that and the $crt_{w'_b}^{\sigma_{rev}^{w_a}}$ to **B**.

5. **B** completes its second move to generate $\sigma_{rev}^{w_b}$ the revocation token for its old wallet. After validating that it now has a valid refund token by verifying $\sigma_{rev}^{w_a}$, it sends $\sigma_{rev}^{w_a}, \sigma_{rev}^{w_b}$ to **I**.

6. **I** completes the remaining moves of the variable payment Pay protocol with **A** and **B** individually, giving each a blind signature on their new wallets.

Anonymity and abort conditions.

In terms of anonymity, the execution of this protocol is no different in terms of the information revealed than two in parallel payments from $\mathbf{A} \rightarrow \mathbf{I}$ and $\mathbf{I} \rightarrow \mathbf{B}$. Our payment anonymity definition already allows this type of attack even for the two party case.

The main challenge in realizing this construction is the possibility that a malicious **I** can selectively abort the protocol during a transaction. This does not allow **I** to steal funds, but it does

force **A** and **B** to transmit messages to the network in order to recover their funds. This potentially links the payment attempt to **A** and **B**'s channels. Unfortunately, this seems fundamentally difficult to avoid in an interactive protocol.

We note that the anonymity threat is limited in practice by the fact that the channels themselves can be funded with an anonymous currency (*e.g.,* [23, 62, 15]), so linking two separate channels does not reveal the participant identifiers. More importantly, since the intermediary can use this abort technique only one time during the lifetime of a channel, there is no possibility for the merchant to link *separate* payments on the same channel. Finally, an intermediary who performs this abort technique will produce public evidence on the network, which allows participants to avoid the malicious intermediary.

4.4.4 From Third Party Payments to Payment Networks

It should be possible to extend the above protocol to allow payments of the form $\mathbf{A} \rightarrow \mathbf{I}_1 \rightarrow \ldots \rightarrow \mathbf{I}_n \rightarrow \mathbf{B}$ via techniques similar to those used in non-anonymous payment channel networks [91]. As discussed in §4.1.2, it is not possible to hide channel balances in such a setting. The general approach is as follows: we use "hash locks" to enforce that either all refund and revocation tokens are valid or none are. Specifically, we attach to both the fund and revocation tokens a condition that they can only be used if one party reveals x such that $y = \mathcal{H}(x)$, where x is picked by **A**. Because if one party releases x, all parties may close their channels, this forces the entire sequence of payments to either go through or not. As with Lightning, the timeouts for each channel must be carefully chosen. We leave the exact details of this approach to future work.

4.4.5 Hiding Channel Balances

Each of the constructions presented above has a privacy limitation: the balance of each payment channel is revealed when a channel is closed. While individuals can protect their identities and initial channel balances by using an anonymous currency mechanism to fund channels, the

	Customer		Merchant		
primitive	Establish(ms)	Pay(ms)	Setup(ms)	Establish(ms)	Pay(ms)
Bilinear CL-Sigs [97]	8.07 ± 0.13	100.13 ± 1.60	1433.51 ± 23.69	15.87 ± 0.27	82.32 ± 1.37
Algebraic MACs [106]	6.90 ± 0.17	37.61 ± 0.93	826.78 ± 19.26	11.97 ± 0.31	34.39 ± 0.88

Figure 4.6: Performance comparison of different implementations of BOLT bidirectional payment protocol. 1000 iterations on a single core of a Intel(R) Xeon(R) CPU E5-2695 v4 @ 2.10GHz. Customer setup is included in Establish.

information about channel balances leaks useful information to the network. We note, however, that in the case of *non-disputed* channel closure, even this information can be hidden from the public as follows. On channel closure, the customer posts a commitment to the final channel balance, along with a zero-knowledge proof that she possesses a valid channel closure token (ı.e., a signature on the channel balance in our bidirectional construction). In systems such as Zerocash [15], the final payment redeeming coins to the merchant and customer can be modified to include an additional statement: *the amounts paid in this transaction are consistent with the commitment, and do not exceed the initial funding transaction that created the channel.* We leave the precise details of such a construction to future work.

4.5 Implementation of the Bidirectional scheme

We now detail the integration of Bolt into a cryptocurrency and performance and cryptographic details of a concrete instantiation.

4.5.1 Integration with a Currency

In this section we consider the problem of integrating the bidirectional Bolt protocol into a Bitcoin like cryptocurrency in a *soft fork*: a protocol change which does not break backward compatibility with existing nodes. Recall that the bidirectional scheme requires that the channels be funded anonymously in order to protect against aborts linking the aborted payment to the channel opening (this does not hold if one wishes merely to prevent multiple payments on the same channel from being linked together). In these conditions, the anonymity of the payment channel is no better

than the anonymity of the underlying cryptocurrency. Of the Bitcoin derived currencies, Zerocash and ZCash [15, 107] provide a strong underlying anonymity layer. Anonymity tools for Bitcoin, such as Coinjoin [108], may also be sufficient in some circumstances and future improvements to Bitcoin may increase the achievable anonymity. The mechanism for deployment is compatible with either currency.

In Bitcoin and ZCash each transaction[10] consists of a set of inputs and a set of outputs. Inputs reference a previous transaction output and contain a `ScriptSig` authorizing use of the funds. Outputs specify the amount of the output and a `ScriptPubKey` specifying when the output can be spent. To evaluate a transaction the `ScriptPubKey` from the previous transaction and `ScriptSig` from the current transaction are combined and evaluated using a stack-based scripting language. In the simplest case, `ScriptPubKey` requires a signature under a specified public key to spend the funds and `ScriptSig` contains such a signature. However, more complex scripts are allowed including control flow such as if statements, time locks that enforce that a given number of blocks has elapsed since the transaction was created, and threshold signatures. As long as the combined script evaluates to True, spending is authorized.

Our soft fork approach involves adding a single opcode, `OP_BOLT` to the scripting language. This opcode has the power to

1. validate the commitment opening and blind signature on the commitment in a refund token, and

2. inspect the output of the transaction and enforce restrictions on it.

Most opcodes do not inspect transaction outputs. The notable exception to this rule are signature opcodes that may hash the entire transaction, including both inputs and outputs, in order to verify the signature. However, it is entirely possible to modify the ZCash and Bitcoin codebase to enable opcodes that do have access to transaction outputs in general.[11] Specifically, our new opcode

[10] We refer here to the "unshielded" transactions in ZCash. Shielded transactions function differently.

[11] In systems derived from Bitcoin core's source code, this requires some modification as the current architecture

will enforce two constraints on the outputs of a transaction closing the channel:

1. Verifying that there are two outputs: one paying the merchant his balance and the other paying the customer hers.

2. Verifying that the customer's payout is time locked such that it can be claimed by the merchant if the refund token has been invalidated (i.e. the customer tried to close on an old channel state).

To accomplish this, we implement the Pay protocol so that the revocation token is the private key corresponding to a Bitcoin (resp. transparent ZCash) address. When a channel is closed, the merchant's fraction of the channel balance is paid in a transaction that is spendable immediately. However, the customer's funds are not immediately spendable by the customer since we need to allow the merchant to dispute the closure. The merchant does this by signing a transaction with both his key and the revocation token key. If the merchant does not do so, the customer can claim the funds after some elapsed time. The script is given below:

```
OP_IF # If merchant
    OP_2 <rev-pubkey><merchant-pubkey> OP_2
    OP_CHECKMULTISIG #2 of 2 multi-sig check
OP_ELSE #If customer wait
    <delay> # delay to wait
    OP_CSV  # Timelock enforces delay
    OP_DROP
    <customer-pubkey> # key for customers funds
    OP_CHECKSIG
OP_ENDIF
```

abstraction in script evaluation provides a callback to get the transaction hash, not direct access to the transaction itself. However this is not a protocol limitation and indeed past versions of the codebase did expose direct access. Exposing direct access does not affect consensus rules in and of itself.

This approach greatly simplifies the implementation. Channel opening consists of posting a transaction with (previously anonymized) inputs containing the needed funds for escrow and a single output. This outputs `ScriptPubKey` contains the new opcode `OP_BOLT` and the channel parameters as one argument. To close the channel, the customer posts the appropriate transaction spending that output with a `ScriptSig` that contains what is required to satisfy `OP_BOLT`: the refund token and two outputs. One output for the merchant's share of the channel which is spendable immediately and one with the customer's spendable under the above time locked script. Finally, each party can post the appropriate transaction claiming their output. The merchant can dispute the channel closure and claim the funds immediately with `OP_FALSE <sig revocation-pubkey> <sig merchant-pubkey>`. On the other hand, the customer must wait and then claim with `OP_TRUE <sig customer-pubkey>`.

Simplified resolution At the cost of an extra round trip in the Pay protocol, we can eliminate the need to validate "blind" signatures in the resolution phase, instead opting to verify a simple commitment opening and a standard (e.g. ECDSA) signature on that commitment. We do this by having the refund token consist of a standard signature on the wallet commitment. Because it is a standard signature, refund token issuance can be linked to usage. This is not a problem if the token is used to handle an abort in the same protocol run as its issuance—our security model assumes the attacker can link a single transaction to channel open/closure. However, it cannot be safely used after that, i.e. in the first step of the next pay protocol, because the unblinded signature will link the current execution of the protocol to the previous one. To solve this, at the start of every run of the pay protocol, we request a fresh refund token on the current wallet before revealing anything. Because the contents of the refund token commitment are deterministically and provably generated from the existing wallet, issuing multiple of them has no other effect. However, it eliminates the need to ever use the old refund token. The only consequence is an additional round trip as we must wait for the refreshed refund token before we can publish.

4.5.2 Implementation

We provide two constructions of Bolt, one using signatures with efficient protocols [97] and the other using using Algebraic MACs [106]. We defer further discussion of primitive selection and usage to Appendix C.1 and move directly to presenting performance numbers in Figure 4.6.

4.6 Related Work

Anonymity and scaling for Bitcoin. A number of works have proposed additional privacy protections for Bitcoin. Zerocoin, Zerocash and similar works [23, 15] provide strong anonymity through the use of complex zero knowledge proofs. A separate line of works seek to increase anonymity by Bitcoin by mixing transactions (e.g. CoinJoin [108], CoinShuffle, CoinSwap). Like Bitcoin, each of these constructions require that all transactions are stored on the blockchain. Finally, recent work has proposed *probabilistic payments* as an alternative payment mechanism [109].

Privacy in payment channels As discussed in detail in the introduction, Heilman *et al.* [110] construct off-chain payments with 3^{rd} party privacy.

Lightning anonymity limitations. The Lightning Network [91] does not provide payment anonymity between pairs of channel participants – *i.e.,* a merchant can see the channel identity of every customer that initiates a payment. However, the protocol includes some limited anonymity protections for *path payments*. These operate on a principle similar to an onion routing network, by using multiple non-colluding intermediaries to obscure the origin and destination of a path. Unfortunately this proposal suffers from collusion problems: given the chain $A \rightarrow I_1 \rightarrow I_2 \rightarrow I_3 \rightarrow B$, only I_1 and I_3 must collude to recover the identities of A and B, since all transactions on the path share the same Hash Timelock Contract ID. Moreover, this security mechanism assumes there exist a network with sufficient path diversity for these protections to be viable. The practical viability of path routing in the Lightning payment network is a subject of some debate given the large amount of funds that would be tied up in maintaining open channels [111, 7]. It seems more likely that deployed

channels will rely on a star topology where clients and merchants interact via a one of a handful of highly-available parties, which is the situation we address in our constructions.

4.7 Acknowledgments

This research was supported in part by the National Science Foundation under awards CNS-1010928,CNS-1228443, and EFMA-1441209; The Office of Naval Research under contract N00014-14-1-0333; and the Mozilla Foundation.

4.8 Conclusion

In this work we showed how to construct anonymous payment channels between two mutually distrustful parties. Our protocols can be instantiated using efficient cryptographic primitives with no trusted third parties and (in many instantiations) no trusted setup. Payments of arbitrary value can be conducted directly between the parties, or via an intermediate connection who learns neither the participants identities nor the amount involved. Coupled with an decentralized anonymous payment scheme for funding the channels, they provide for private instantaneous anonymous payments without a trusted bank.

We leave two main open problems. The first is to investigate the necessary details for extending the third party payment protocol to support arbitrary paths consisting of $n > 3$ parties. Second, we did not consider the problem of performing payment resolution (in the event of a dispute) without revealing the final channel balance to the network.

Chapter 5

Conclusion

In this work we proposed a number of privacy preserving payment schemes for blockchain based currencies. These schemes leave open two principle problems:

1. The construction of a more efficient version of Zerocash using zkSNARKs and a better choice of primitives. This problem raises a more general question: how to construct cryptographic primitives such as hash functions and permutations that have efficient representations in zkSNARKs?

2. Privacy preserving scripting and smart contracts. Bitcoin supports a scripting language where complex authorization checks for spending a payment can be specified. Ethereum [112] takes this one step further and allows Turing complete smart contracts with almost arbitrary functionality. In both cases, these scripts are executed in the clear. Extending execution to the private setting and doing so efficiently is an interesting avenue for future research.

Appendix A

Zerocoin

A.1 Security Proofs

A.1.1 Proof of Theorem 2.3.1

Proof. Let $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ be an adversary that wins the Anonymity game with advantage ϵ. We approach this proof in two steps. In the first step we replace the proof π given to $\mathcal{A}$ with a simulated signature of knowledge.[1] In the second step, we replace the coins (c_0, c_1) and serial number S provided to $\mathcal{A}$ with random values sampled from the appropriate distributions. At each step we argue that $\mathcal{A}$'s advantage is only negligibly different than its advantage against the real protocol. Finally, we note that $\mathcal{A}$'s view in the final game is independent of the bit b, and therefore $\mathcal{A}$'s advantage must be negligible in the security parameter.

Replacing the SoK. We first argue that if the SoK π is computationally zero knowledge,[2] then π can be replaced with a simulated signature π' without substantially affecting $\mathcal{A}$'s advantage. More concretely, let $O_{SoK}(\cdot)$ be an oracle that on input a valid witness to the statement defined in the real protocol, flips a random bit $\hat{b}$ and if $b = 0$ outputs a SoK as in the real protocol, and if $b = 1$

[1] Our proofs assume the existence of an efficient simulator and extractor for the ZKSoK. See Appendix A.2.

[2] We state a relatively weak result here, and note that the same necessarily holds if π is perfect- or statistically zero knowledge.

outputs a simulated SoK π'. We show that if $\mathcal{A}$ wins the anonymity game with non-negligibly different probability when π is replaced with a simulated proof, we can construct a distinguisher that determines the bit $\hat{b}$ selected by the oracle O_{SoK}.

The simulation proceeds as follows: first, $\mathcal{A}'$ generates $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$ as in the real protocol. It next mints two coins $c_0, c_1 \in \mathsf{Mint}(\mathsf{pp})$ and stores the associated trapdoors (skc_0, skc_1). It provides the resulting values to $\mathcal{A}_1$, which outputs a set $\mathbf{C}$ and transaction string info using any strategy it wishes. Next $\mathcal{A}'$ samples $b \leftarrow \{0,1\}$ and executes the Spend algorithm on coin b as in the normal protocol, though with one modification: rather than execute the ZKSoK algorithm it instead queries the oracle O_{SoK} on the identical inputs to obtain π. It then provides (S, π) to $\mathcal{A}_2$. When $\mathcal{A}_2$ outputs a guess b', $\mathcal{A}'$ outputs 1 if $b = b'$.

Note that when O_{SoK} executes with $\hat{b} = 0$ (*i.e.,* generates a standard SoK) the input to $\mathcal{A}_2$ is distributed identically to the real protocol, thus $\mathcal{A}$'s advantage is ϵ and $\mathcal{A}'$ outputs 1 with probability $1/2 + \epsilon$. Thus it remains to consider the case where O_{SoK} outputs a simulated proof. In this case, if $\mathcal{A}$ wins the anonymity game with probability non-negligibly different than ϵ then $\mathcal{A}'$ would guess the bit $\hat{b}$ chosen O_{SoK} with non-negligible advantage. Thus $\mathcal{A}$'s advantage in winning the anonymity game must be at most $\epsilon + \mathrm{negl}(\lambda)$ when the SoK is replaced with a simulated SoK.

Modifying the values. We next consider the following modification: rather than generate the coins c_0, c_1 as in the real protocol, we instead sample two values c_0, c_1 at random from the set of prime numbers in the range $[A, B]$.[3] We also sample the value S at random from $\mathbb{Z}_q^*$ and simulate the protocol using these values. Finally, we simulate the SoK π. We observe that all values in this protocol are distributed identically to those in the previous game, and $\mathcal{A}$'s view is independent of the bit b. By implication, $\Pr[\, b = b' \,] = 1/2 + \mathrm{negl}(\lambda)$ and $\mathcal{A}$'s advantage in the anonymity game ϵ must be negligible. □

A.1.2 Proof of Theorem 2.3.2

[3] "Where A and B can be chosen with arbitrary polynomial dependence on the security parameter, as long as $2 < A$ and $B < A^2$." [113] For a full description, see [113, §3.2 and §3.3].

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 $\mathsf{pp} \leftarrow (N, u, p, q, g, h)$. For $i = 1$ to K, compute $(c_i, skc_i) \leftarrow \mathsf{Mint}(\mathsf{pp})$, where $skc_i = (S_i, r_i)$, and run $\mathcal{A}(\mathsf{pp}, c_1, \ldots, c_K)$. Answer each of $\mathcal{A}$'s queries to $\mathcal{O}_{spend}$ using the appropriate trapdoor information. Let $(S_1, \mathsf{info}_1), \ldots, (S_l, \mathsf{info}_l)$ 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 $(\pi'_i, S'_i, \mathsf{info}'_i, \mathbf{C}'_i)$. 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 $\textsc{Event}_{\textsc{Ext}}$.
2. If $c^*_j \notin \mathbf{C}'_j$, abort and signal $\textsc{Event}_{\textsc{Acc}}$.
3. If $c^*_j \in \{c_1, \ldots, c_K\}$:
   (a) If for some i, $(S'_j, r^*_j) = (S_i, r_i)$ and $\mathsf{info}'_j \neq \mathsf{info}_i$, abort and signal $\textsc{Event}_{\textsc{Forge}}$.
   (b) Otherwise if for some i, $(S'_j, r^*_j) = (S_i, r_i)$, abort and signal $\textsc{Event}_{\textsc{Col}}$.
   (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), $c_j^* = c_i \in \{c_1, \ldots, c_K\}$. 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 $\text{negl}_1(\lambda)$ be the (negligible) probability that the extractor fails on input π. By summation, $\Pr[\,\text{EVENT}_{\text{EXT}}\,] \leq (M+1)\text{negl}_1(\lambda)$. 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 $\Pr[\,\text{EVENT}_{\text{COL}}\,] \leq 1/l$.

Next, we argue that under the Strong RSA and Discrete Log assumptions, $\Pr[\,\text{EVENT}_{\text{ACC}}\,] \leq \text{negl}_2(\lambda)$ and $\Pr[\,\text{EVENT}_{\text{FORGE}}\,] \leq \text{negl}_3(\lambda)$. We show this in Lemmas A.1.1 and A.1.2 below. If $\mathcal{A}$ succeeds with advantage ϵ, then by summing the above probabilities we show that $\mathcal{B}$ succeeds with probability $\geq \epsilon - ((M+1)\text{negl}_1(\lambda) + \text{negl}_2(\lambda) + \text{negl}_3(\lambda) + 1/l)$. We conclude with the remaining Lemmas.

Lemma A.1.1. *Under the Strong RSA assumption,* $\Pr[\,\text{EVENT}_{\text{ACC}}\,] \leq \text{negl}_2(\lambda)$.

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 [36, §3] to solve the Strong RSA problem. For the

complete details we refer the reader to [36, §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 $\mathsf{pp} = (N, u, p, q, g, h)$. 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 [36, §3] and uses the resulting value to compute a solution to the Strong RSA instance. □

Lemma A.1.2. *Under the Discrete Logarithm assumption,* $\Pr[\,\text{EVENT}_{\text{FORGE}}\,] \leq \text{negl}_3(\lambda)$.

Proof sketch. The proof is similar to those used by earlier schemes, e.g., [34]. We now sketch it. 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 info' 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. □

□

A.2 Zero-Knowledge Proof Construction

The signature of knowledge

$$\pi = \mathsf{ZKSoK}[\mathsf{info}]\{(c, w, r) :$$

$$\mathsf{AccVerify}((N,u),A,c,w) = 1 \ \wedge \ c = g^S h^r\}$$

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 [113, §3.3 and Appendix A]. The latter is a double discrete log signature of knowledge that, although related to previous work [31, §5.3.3], is new (at least to us). 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
>
> $$\begin{aligned} \textbf{if } c[i] = 0 \textbf{ then } & s_i = r_i, s'_i = v_i; \\ \textbf{else } & s_i = r_i - z, s'_i = v_i - w b^{r_i - z}; \end{aligned}$$
>
> 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
>
> $$\begin{aligned} \textbf{if } c[i] = 0 \textbf{ then } & \bar{t_i} = g^{a^x b^{s_i}} h^{s'_i}; \\ \textbf{else } & \bar{t_i} = y_1^{b^{s_i}} h^{s'_i}; \end{aligned}$$
>
> and check whether $c = c'$.

Simulating and extracting. Our proofs in Appendix A.1 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., [34].

A.2.1 Proof Construction

The proof is constructed as follows:

> Given $y_1 = g_1^{a^x} g_2^w$ and $y_2 = g_3^x$, we want to prove that:
>
> $$\mathrm{Dlog}_a(\mathrm{Dlog}_{g_1}(y_1/g_2^w)) = \log_{g_3}(y_2) = x$$

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 $1 \leq i \leq l$, $t_i = g_1^{a^{r_i}} g_2^{v_i}$ and $t'_1 = g_3^{r_i}$. 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:

$$c = H(m \| y_1 \| y_2 \| a \| g_1 \| g_2 \| g_3 \| t_1 \| \ldots \| t_l \| t'_1 \| \ldots \| t'_l)$$

and

$$\begin{aligned} \textbf{if } c[i] = 0 \textbf{ then } s_i = r_i, s'_i = v_i; \\ \textbf{else } s_i = r_i - x, s'_i = v_i - w; \end{aligned}$$

To verify the signature it is sufficient to compute:

$$c' = H(m \| y_1 \| y_2 \| a \| g_1 \| g_2 \| g_3 \| \bar{t_1} \| \ldots \| \bar{t_l} \| \bar{t'_1} \| \ldots \| \bar{t'_l})$$

with

$$\begin{aligned} \textbf{if } c[i] = 0 \textbf{ then } \bar{t_i} = g_1^{a^{s_i}} g_2^{s'_i}, \bar{t'_i} = g_3^{s_i}; \\ \textbf{else } \bar{t_i} = y_1 g_1^{a^{s_i}} g_2^{s'_i}, \bar{t'_i} = y_2 g_3^{s_i}; \end{aligned}$$

and check whether $c = c'$.

A.2.2 HVZK

$\lambda_1 = p(|g_3|)$, $\lambda_2 = p(|g_2|)$

λ_i upper bound on size of x, w respectively ie $0 \leq x, w \leq 2^{\lambda_i}$

$\epsilon > 1$ constant

Let S_v be a simulator. Choose $c \in_R \{0, \ldots, 2^k - 1\}$. Choose $r_i = s_i \in_R \{0, \ldots, 2^{\epsilon\lambda_1} - 1\}$. Choose $v_i = s'_i \in_R \{0, \ldots, 2^{\epsilon\lambda_2} - 1\}$.

Using these values, the simulator computes:

$$t_i = \begin{cases} g_1^{a^{s_i}} g_2^{s'_i} & \text{if } c[i] = 0 \\ y_1 g_1^{a^{s_i}} g_2^{s'_i} & \text{else} \end{cases} \qquad t'_i = \begin{cases} g_3^{s_i} & \text{if } c[i] = 0 \\ y_2 g_3^{s_i} & \text{else} \end{cases}$$

for $i = 1, \ldots, l$. To prove that these values are statistically indistinguishable from a view of a

protocol run with the prover, it suffices to consider the probability distributions $P_{S_1,\ldots,S_l}(s_1,\ldots,s_l)$ of the s_i's of the prover and $P_{S'_1,\ldots,S'_l}(s'_1,\ldots,s'_l)$ of the s'_i's of the prover and $P_{R_1,\ldots,R_l}(r_1,\ldots,r_l)$ of the r_i's of the simulator and $P_{V_1,\ldots,V_l}(v_1,\ldots,v_l)$ of the v_i's of the simulator. The latter values are $\prod_{i=1}^{l} P_{R_i}(r_i)$ and $\prod_{i=1}^{l} P_{V_i}(v_i)$ where $P_{R_i}(r_i)$ is the uniform distribution over $\{0,\ldots,2^{\epsilon\lambda_1}-1\}$ and $P_{V_i}(v_i)$ is the uniform distributions over $\{0,\ldots,2^{\epsilon\lambda_2}-1\}$. If the prover chooses the r_i's uniformly at random from $\{0,\ldots,2^{\epsilon\lambda_1}-1\}$ and the v_i's uniformly at random from $\{0,\ldots,2^{\epsilon\lambda_2}-1\}$ and the secret key randomly from $\{0,\ldots,2^{\lambda}-1\}$ according to any distribution, we have

$$P_{S_i}(s) = \begin{cases} = 0 & \text{for } s < -(2^{\lambda_1}-1) \\ \leq 2^{-\epsilon\lambda_1} & \text{for } -(2^{\lambda_1}-1) \leq s < 0 \\ = 2^{-\epsilon\lambda_1} & \text{for } 0 \leq s \leq 2^{\epsilon\lambda_1}-2^{\lambda_1} \\ \leq 2^{-\epsilon\lambda_1} & \text{for } 2^{\epsilon\lambda_1}-2^{\lambda_1} < s \leq (2^{\epsilon\lambda_1}-1) \\ = 0 & \text{for } 2^{\epsilon\lambda_1}-1 < s \end{cases} \quad P_{V_i}(v) = \begin{cases} = 0 & \text{for } s < -(2^{\lambda_2}-1) \\ \leq 2^{-\epsilon\lambda_2} & \text{for } -(2^{\lambda_2}-1) \leq s < 0 \\ = 2^{-\epsilon\lambda_2} & \text{for } 0 \leq s \leq 2^{\epsilon\lambda_2}-2^{\lambda_2} \\ \leq 2^{-\epsilon\lambda_2} & \text{for } 2^{\epsilon\lambda_2}-2^{\lambda_2} < s \leq (2^{\epsilon\lambda_2}-1) \\ = 0 & \text{for } 2^{\epsilon\lambda_2}-1 < s \end{cases}$$

These hold for any distribution of $c[i]$ over $\{0,1\}$. Similarly, the probability $P_{S'_1,\ldots,S'_l}(s'_1,\ldots,s'_l)$ (resp. $P_{V'_1,\ldots,V'_l}(v'_1,\ldots,v'_l)$) is 0 if any s_i (resp. v_i) is smaller than $-(2^{\lambda}_1-1)$ (resp. $-(2^{\lambda}_2-1)$) or larger than $-(2^{\epsilon}\lambda_1-1)$ (resp. $-(2^{\epsilon}\lambda_2-1)$), equals $2^{-k\epsilon\lambda_1}$ (resp. $2^{-k\epsilon\lambda_2}$) if all s_i (resp. v_i) are in the range $\{0,\ldots,2^{\epsilon\lambda_1}-2^{\lambda_1}\}$ (resp. $\{0,\ldots,2^{\epsilon\lambda_2}-2^{\lambda_2}\}$, and is smaller or equal to $2^{-k\epsilon\lambda_1}$ (resp. $2^{-k\epsilon\lambda_2}$) in the other cases. Thus we have

$$\begin{aligned} \sum_{a\in(\mathbb{Z})^k} |P_{S'_1,\ldots,S'_l}(\alpha) - P_{R'_1,\ldots,R'_l}(\alpha)| &\leq \frac{2(2^{k\epsilon\lambda_1} - (2^{k\lambda_1} - 2^{\lambda_1}+1)^k)}{2^{k\epsilon\lambda_1}} \\ = \frac{2(2^{k\epsilon\lambda_1} - 2^{k\epsilon\lambda_1}(1+\frac{1-2^{\lambda_1}}{2^{\epsilon\lambda_1}})^k)}{2^{k\epsilon\lambda_1}} &\leq 2(1-(1+k\frac{1-2^{\lambda_1}}{2^{\epsilon\lambda_1}})) \\ = 2k\frac{2^{\lambda_1-1}}{2^{\epsilon\lambda_1}} &< \frac{2k}{(2^{\lambda_1})^{\epsilon-1}} \end{aligned}$$

and

$$\sum_{a\in(\mathbb{Z})^k} |P_{S'_1,\ldots,S'_l}(\alpha) - P_{R'_1,\ldots,R'_l}(\alpha)| \leq \frac{2(2^{k\epsilon\lambda_2} - (2^{k\lambda_2} - 2^{\lambda_2} + 1)^k)}{2^{k\epsilon\lambda_2}}$$
$$= \frac{2(2^{k\epsilon\lambda_2} - 2^{k\epsilon\lambda_2}(1 + \frac{1-2^{\lambda_2}}{2^{\epsilon\lambda_2}})^k)}{2^{k\epsilon\lambda_2}} \leq 2(1 - (1 + k\frac{1-2^{\lambda_2}}{2^{\epsilon\lambda_2}}))$$
$$= 2k\frac{2^{\lambda_2-1}}{2^{\epsilon\lambda_2}} < \frac{2k}{(2^{\lambda_2})^{\epsilon-1}}$$

For λ_1 and λ_2 as defined, the last term can be expressed as one over a polynomial in the input length, and therefore the two distributions are indistinguishable.

A.2.3 Soundness

We proceed as follows:

Without loss of generality, we assume that the j-th bits of c and $\tilde{c}$ differ and that $c[j] = 0$. Then we have

$$t_j = g_1^{a^{s_j}} g_2^{s'_j} = y_1 g_1^{a^{\tilde{s}_j}} g_2^{\tilde{s}'_j} = g_1^{a^x} g_2^w (g_1^{a^{\tilde{s}_j}} g_2^{\tilde{s}'_j})$$

and thus $a^x \equiv a^{s_j - \tilde{s}_j} \pmod{n}$ holds. Hence we can compute (in (Z)

$$x = s_j - \tilde{s}_j$$

$$w = s'_j - \tilde{s}'_j$$

A.3 Zero-Knowledge Proofs

A.3.1 Proof Construction

The proof is constructed as follows:

Given $y_1 = g_1^{a^x} g_2^w$ and $y_2 = g_3^x$, we want to prove that:

$$\text{Dlog}_a(\text{Dlog}_{g_1}(y_1/g_2^w)) = \log_{g_3}(y_2) = x$$

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 $1 \leq i \leq l$, $t_i = g_1^{a^{r_i}} g_2^{v_i}$ and $t'_1 = g_3^{r_i}$. 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:

$$c = H(m \| y_1 \| y_2 \| a \| g_1 \| g_2 \| g_3 \| t_1 \| \ldots \| t_l \| t'_1 \| \ldots \| t'_l)$$

and

$$\begin{aligned} \textbf{if } c[i] = 0 \textbf{ then } s_i = r_i, s'_i = v_i; \\ \textbf{else } s_i = r_i - x, s'_i = v_i - w; \end{aligned}$$

To verify the signature it is sufficient to compute:

$$c' = H(m \| y_1 \| y_2 \| a \| g_1 \| g_2 \| g_3 \| \bar{t_1} \| \ldots \| \bar{t_l} \| \bar{t'_1} \| \ldots \| \bar{t'_l})$$

with

$$\begin{aligned} \textbf{if } c[i] = 0 \textbf{ then } \bar{t_i} = g_1^{a^{s_i}} g_2^{s'_i}, \bar{t'_i} = g_3^{s_i}; \\ \textbf{else } \bar{t_i} = y_1 g_1^{a^{s_i}} g_2^{s'_i}, \bar{t'_i} = y_2 g_3^{s_i}; \end{aligned}$$

and check whether $c = c'$.

We now show that the above proof is Honest Verifier Zero Knowledge and Sound.

A.3.2 HVZK

$\lambda_1 = p(|g_3|)$, $\lambda_2 = p(|g_2|)$

λ_i upper bound on size of x, w respectively ie $0 \leq x, w \leq 2^{\lambda_i}$

$\epsilon > 1$ constant

Let S_v be a simulator. Choose $c \in_R \{0, \ldots, 2^k - 1\}$. Choose $r_i = s_i \in_R \{0, \ldots, 2^{\epsilon\lambda_1} - 1\}$. Choose

$v_i = s'_i \in_R \{0, \ldots, 2^{\epsilon\lambda_2} - 1\}$.

Using these values, the simulator computes:

$$t_i = \begin{cases} g_1^{a^{s_i}} g_2^{s_i'} & \text{if } c[i] = 0 \\ y_1 g_1^{a^{s_i}} g_2^{s_i'} & \text{else} \end{cases} \qquad t_i' = \begin{cases} g_3^{s_i} & \text{if } c[i] = 0 \\ y_2 g_3^{s_i} & \text{else} \end{cases}$$

for $i = 1, \ldots, l$. To prove that these values are statistically indistinguishable from a view of a protocol run with the prover, it suffices to consider the probability distributions $P_{S_1,\ldots,S_l}(s_1, \ldots, s_l)$ of the s_i's of the prover and $P_{S_1',\ldots,S_l'}(s_1', \ldots, s_l')$ of the s_i''s of the prover and $P_{R_1,\ldots,R_l}(r_1, \ldots, r_l)$ of the r_i's of the simulator and $P_{V_1,\ldots,V_l}(v_1, \ldots, v_l)$ of the v_i's of the simulator. The latter values are $\prod_{i=1}^{l} P_{R_i}(r_i)$ and $\prod_{i=1}^{l} P_{V_i}(v_i)$ where $P_{R_i}(r_i)$ is the uniform distribution over $\{0, \ldots, 2^{\epsilon\lambda_1} - 1\}$ and $P_{V_i}(v_i)$ is the uniform distributions over $\{0, \ldots, 2^{\epsilon\lambda_2} - 1\}$. If the prover chooses the r_i's uniformly at random from $\{0, \ldots, 2^{\epsilon\lambda_1} - 1\}$ and the v_i's uniformly at random from $\{0, \ldots, 2^{\epsilon\lambda_2} - 1\}$ and the secret key randomly from $\{0, \ldots, 2^{\lambda} - 1\}$ according to any distribution, we have

$$P_{S_i}(s) = \begin{cases} = 0 & \text{for } s < -(2^{\lambda_1} - 1) \\ \leq 2^{-\epsilon\lambda_1} & \text{for } -(2^{\lambda_1} - 1) \leq s < 0 \\ = 2^{-\epsilon\lambda_1} & \text{for } 0 \leq s \leq 2^{\epsilon\lambda_1} - 2^{\lambda_1} \\ \leq 2^{-\epsilon\lambda_1} & \text{for } 2^{\epsilon\lambda_1} - 2^{\lambda_1} < s \leq (2^{\epsilon\lambda_1} - 1) \\ = 0 & \text{for } 2^{\epsilon\lambda_1} - 1 < s \end{cases} \quad P_{V_i}(v) = \begin{cases} = 0 & \text{for } s < -(2^{\lambda_2} - 1) \\ \leq 2^{-\epsilon\lambda_2} & \text{for } -(2^{\lambda_2} - 1) \leq s < 0 \\ = 2^{-\epsilon\lambda_2} & \text{for } 0 \leq s \leq 2^{\epsilon\lambda_2} - 2^{\lambda_2} \\ \leq 2^{-\epsilon\lambda_2} & \text{for } 2^{\epsilon\lambda_2} - 2^{\lambda_2} < s \leq (2^{\epsilon\lambda_2} - 1) \\ = 0 & \text{for } 2^{\epsilon\lambda_2} - 1 < s \end{cases}$$

These hold for any distribution of $c[i]$ over $\{0, 1\}$. Similarly, the probability $P_{S_1',\ldots,S_l'}(s_1', \ldots, s_l')$ (resp. $P_{V_1',\ldots,V_l'}(v_1', \ldots, v_l')$) is 0 if any s_i (resp. v_i) is smaller than $-(2_1^{\lambda} - 1)$ (resp. $-(2_2^{\lambda} - 1)$) or larger than $-(2^{\epsilon}\lambda_1 - 1)$ (resp. $-(2^{\epsilon}\lambda_2 - 1)$), equals $2^{-k\epsilon\lambda_1}$ (resp. $2^{-k\epsilon\lambda_2}$) if all s_i (resp. v_i) are in the range $\{0, \ldots, 2^{\epsilon\lambda_1} - 2^{\lambda_1}\}$ (resp. $\{0, \ldots, 2^{\epsilon\lambda_2} - 2^{\lambda_2}\}$, and is smaller or equal to $2^{-k\epsilon\lambda_1}$ (resp. $2^{-k\epsilon\lambda_2}$) in the other cases. Thus we have

$$\sum_{a \in (\mathbb{Z})^k} |P_{S_1',\ldots,S_l'}(\alpha) - P_{R_1',\ldots,R_l'}(\alpha)| \leq \frac{2(2^{k\epsilon\lambda_1} - (2^{k\lambda_1} - 2^{\lambda_1} + 1)^k)}{2^{k\epsilon\lambda_1}}$$

$$= \frac{2(2^{k\epsilon\lambda_1} - 2^{k\epsilon\lambda_1}(1 + \frac{1-2^{\lambda_1}}{2^{\epsilon\lambda_1}})^k)}{2^{k\epsilon\lambda_1}} \leq 2(1 - (1 + k\frac{1-2^{\lambda_1}}{2^{\epsilon\lambda_1}}))$$
$$= 2k\frac{2^{\lambda_1 - 1}}{2^{\epsilon\lambda_1}} < \frac{2k}{(2^{\lambda_1})^{\epsilon-1}}$$

and

$$\sum_{a\in(\mathbb{Z})^k} |P_{S'_1,\ldots,S'_l}(\alpha) - P_{R'_1,\ldots,R'_l}(\alpha)| \leq \frac{2(2^{k\epsilon\lambda_2} - (2^{k\lambda_2} - 2^{\lambda_2} + 1)^k)}{2^{k\epsilon\lambda_2}}$$
$$= \frac{2(2^{k\epsilon\lambda_2} - 2^{k\epsilon\lambda_2}(1 + \frac{1-2^{\lambda_2}}{2^{\epsilon\lambda_2}})^k)}{2^{k\epsilon\lambda_2}} \leq 2(1 - (1 + k\frac{1-2^{\lambda_2}}{2^{\epsilon\lambda_2}}))$$
$$= 2k\frac{2^{\lambda_2 - 1}}{2^{\epsilon\lambda_2}} < \frac{2k}{(2^{\lambda_2})^{\epsilon-1}}$$

For λ_1 and λ_2 as defined, the last term can be expressed as one over a polynomial in the input length, and therefore the two distributions are indistinguishable.

A.3.3 Soundness

We proceed as follows:

Without loss of generality, we assume that the j-th bits of c and $\tilde{c}$ differ and that $c[j] = 0$. Then we have

$$t_j = g_1^{a^{s_j}} g_2^{s'_j} = y_1 g_1^{a^{\tilde{s}_j}} g_2^{\tilde{s}'_j} = g_1^{a^x} g_2^w (g_1^{a^{\tilde{s}_j}} g_2^{\tilde{s}'_j})$$

and thus $a^x \equiv a^{s_j - \tilde{s}_j} \pmod{n}$ holds. Hence we can compute (in (Z)

$$x = s_j - \tilde{s}_j$$

$$w = s'_j - \tilde{s}'_j$$

Appendix B

Zerocash

B.1 Completeness of DAP schemes

A DAP scheme $\Pi = (\mathsf{Setup}, \mathsf{CreateAddress}, \mathsf{Mint}, \mathsf{Pour}, \mathsf{VerifyTransaction}, \mathsf{Receive})$ is **complete** if no polynomial-size ledger sampler $\mathcal{S}$ can win the incompleteness experiment with more than negligible probability. In Section 3.3.4 we informally described this property; we now formally define it.

Definition B.1.1. *Let* $\Pi = (\mathsf{Setup}, \mathsf{CreateAddress}, \mathsf{Mint}, \mathsf{Pour}, \mathsf{VerifyTransaction}, \mathsf{Receive})$ *be a (candidate) DAP scheme. We say that* Π *is* ***complete*** *if, for every* $\mathrm{poly}(\lambda)$*-size ledger sampler* $\mathcal{S}$ *and sufficiently large* λ*,*

$$\mathsf{Adv}^{\mathsf{INCOMP}}_{\Pi,\mathcal{S}}(\lambda) < \mathrm{negl}(\lambda) \ ,$$

where $\mathsf{Adv}^{\mathsf{INCOMP}}_{\Pi,\mathcal{S}}(\lambda) := \Pr[\mathsf{INCOMP}(\Pi, \mathcal{S}, \lambda) = 1]$ *is* $\mathcal{S}$*'s advantage in the incompleteness experiment.*

We now describe the incompleteness experiment mentioned above. Given a (candidate) DAP scheme Π, a ledger sampler $\mathcal{S}$, and a security parameter λ, the (probabilistic) experiment $\mathsf{INCOMP}(\Pi, \mathcal{S}, \lambda)$ consists of an interaction between $\mathcal{S}$ and a challenger $\mathcal{C}$, terminating with a binary output by $\mathcal{C}$.

At the beginning of the experiment, $\mathcal{C}$ samples $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$ and sends pp to $\mathcal{S}$. Then, $\mathcal{S}$ sends $\mathcal{C}$ a ledger, two coins to be spent, and parameters for a pour transaction; more precisely, $\mathcal{S}$ sends (1) a ledger L; (2) two coins $\mathbf{c}_1^{\mathsf{old}}$, $\mathbf{c}_2^{\mathsf{old}}$; (3) two address secret keys $\mathsf{addr}_{\mathsf{sk},1}^{\mathsf{old}}, \mathsf{addr}_{\mathsf{sk},2}^{\mathsf{old}}$; (4) two values $v_1^{\mathsf{new}}, v_2^{\mathsf{new}}$; (5) new address key pairs $(\mathsf{addr}_{\mathsf{pk},1}^{\mathsf{new}}, \mathsf{addr}_{\mathsf{sk},1}^{\mathsf{new}}), (\mathsf{addr}_{\mathsf{pk},2}^{\mathsf{new}}, \mathsf{addr}_{\mathsf{sk},2}^{\mathsf{new}})$; (6) a public value v_{pub}; and (7) a transaction string info. Afterwards, $\mathcal{C}$ performs various checks on $\mathcal{S}$'s message.

Concretely, $\mathcal{C}$ first checks that $\mathbf{c}_1^{\mathsf{old}}$ and $\mathbf{c}_2^{\mathsf{old}}$ are valid unspent coins, i.e., checks that: (i) $\mathbf{c}_1^{\mathsf{old}}$ and $\mathbf{c}_2^{\mathsf{old}}$ are well-formed; (ii) their coin commitments $\mathsf{cm}_1^{\mathsf{old}}$ and $\mathsf{cm}_2^{\mathsf{old}}$ appear in (valid) transactions on L; (iii) their serial numbers $\mathsf{sn}_1^{\mathsf{old}}$ and $\mathsf{sn}_2^{\mathsf{old}}$ do *not* appear in (valid) transactions on L. Next, $\mathcal{C}$ checks that $v_1^{\mathsf{new}} + v_2^{\mathsf{new}} + v_{\mathsf{pub}} = v_1^{\mathsf{old}} + v_2^{\mathsf{old}}$ (i.e., the values suggested by $\mathcal{S}$ preserve balance) and $v_1^{\mathsf{old}} + v_2^{\mathsf{old}} \leq v_{\mathsf{max}}$ (i.e., the maximum value is not exceeded). If any of these checks fail, $\mathcal{C}$ aborts and outputs 0.

Otherwise, $\mathcal{C}$ computes rt, the Merkle-tree root over all coin commitments in L (appearing in valid transactions), and, for $i \in \{1, 2, \}$, path_i, the authentication path from commitment $\mathsf{cm}_i^{\mathsf{old}}$ to the root rt. Then, $\mathcal{C}$ attempts to spend $\mathbf{c}_1^{\mathsf{old}}$, $\mathbf{c}_2^{\mathsf{old}}$ as instructed by $\mathcal{S}$:

$$(\mathbf{c}_1^{\mathsf{new}}, \mathbf{c}_2^{\mathsf{new}}, \mathsf{tx}_{\mathsf{Pour}}) \leftarrow \mathsf{Pour}(\mathsf{pp}, \mathsf{rt}, \mathbf{c}_1^{\mathsf{old}}, \mathbf{c}_2^{\mathsf{old}}, \mathsf{addr}_{\mathsf{sk},1}^{\mathsf{old}}, \mathsf{addr}_{\mathsf{sk},2}^{\mathsf{old}}, \mathsf{path}_1, \mathsf{path}_2, v_1^{\mathsf{new}}, v_2^{\mathsf{new}}, \mathsf{addr}_{\mathsf{pk},1}^{\mathsf{new}}, \mathsf{addr}_{\mathsf{pk},2}^{\mathsf{new}}, v_{\mathsf{pub}}, \mathsf{info}) \ .$$

Finally, $\mathcal{C}$ outputs 1 if and only if any of the following conditions hold:

- $\mathsf{tx}_{\mathsf{Pour}} \neq (\mathsf{rt}, \mathsf{sn}_1^{\mathsf{old}}, \mathsf{sn}_2^{\mathsf{old}}, \mathsf{cm}_1^{\mathsf{new}}, \mathsf{cm}_2^{\mathsf{new}}, v_{\mathsf{pub}}, \mathsf{info}, *)$, where $\mathsf{cm}_1^{\mathsf{new}}, \mathsf{cm}_2^{\mathsf{new}}$ are the coin commitments of $\mathbf{c}_1^{\mathsf{new}}, \mathbf{c}_2^{\mathsf{new}}$; OR
- $\mathsf{tx}_{\mathsf{Pour}}$ is not valid, i.e., $\mathsf{VerifyTransaction}(\mathsf{pp}, \mathsf{tx}_{\mathsf{Pour}}, L)$ outputs 0; OR
- for some $i \in \{1, 2\}$, the coin $\mathbf{c}_i^{\mathsf{new}}$ is not returned by $\mathsf{Receive}(\mathsf{pp}, (\mathsf{addr}_{\mathsf{pk},i}^{\mathsf{new}}, \mathsf{addr}_{\mathsf{sk},i}^{\mathsf{new}}), L')$, where L' is the ledger obtained by appending $\mathsf{tx}_{\mathsf{Pour}}$ to L.

Remark. There is no need for the challenger $\mathcal{C}$ check that, in turn, both $\mathbf{c}_1^{\mathsf{new}}$ and $\mathbf{c}_2^{\mathsf{new}}$ are spendable, because this follows by induction. Namely, if $\mathbf{c}_1^{\mathsf{new}}, \mathbf{c}_2^{\mathsf{new}}$ were not spendable, a different sampler $\mathcal{S}'$ (that simulates $\mathcal{S}$ and then computes and outputs $\mathbf{c}_1^{\mathsf{new}}$ and $\mathbf{c}_2^{\mathsf{new}}$) would provide a counterexample to the above definition.

B.2 Security of DAP schemes

A DAP scheme $\Pi = (\mathsf{Setup}, \mathsf{CreateAddress}, \mathsf{Mint}, \mathsf{Pour}, \mathsf{VerifyTransaction}, \mathsf{Receive})$ is *secure* if it satisfies ledger indistinguishability, transaction non-malleability, and balance. (See Definition 3.3.2.) In Section 3.3.4 we informally described these three properties; we now formally define them.

Each of the definitions employs an experiment involving a (stateful) *DAP oracle* $\mathcal{O}^{\mathsf{DAP}}$ that receives and answers queries from an adversary $\mathcal{A}$ (proxied via a challenger $\mathcal{C}$, which performs the experiment-specific sanity checks). Below, we first describe how $\mathcal{O}^{\mathsf{DAP}}$ works.

The oracle $\mathcal{O}^{\mathsf{DAP}}$ is initialized by a list of public parameters pp and maintains state. Internally, $\mathcal{O}^{\mathsf{DAP}}$ stores: (i) L, a ledger; (ii) ADDR, a set of address key pairs; (iii) COIN, a set of coins. All of $L, \mathsf{ADDR}, \mathsf{COIN}$ start out empty. The oracle $\mathcal{O}^{\mathsf{DAP}}$ accepts different types of queries, and each query causes different updates to $L, \mathsf{ADDR}, \mathsf{COIN}$ and outputs. We now describe each type of query Q.

- $Q = (\textbf{CreateAddress})$

  1. Compute $(\mathsf{addr}_{\mathsf{pk}}, \mathsf{addr}_{\mathsf{sk}}) := \mathsf{CreateAddress}(\mathsf{pp})$.
  2. Add the address key pair $(\mathsf{addr}_{\mathsf{pk}}, \mathsf{addr}_{\mathsf{sk}})$ to ADDR.
  3. Output the address public key $\mathsf{addr}_{\mathsf{pk}}$.

  The ledger L and coin set COIN remain unchanged.

- $Q = (\textbf{Mint}, v, \mathsf{addr}_{\mathsf{pk}})$

  1. Compute $(\mathbf{c}, \mathsf{tx}_{\mathsf{Mint}}) := \mathsf{Mint}(\mathsf{pp}, v, \mathsf{addr}_{\mathsf{pk}})$.
  2. Add the coin $\mathbf{c}$ to COIN.
  3. Add the mint transaction $\mathsf{tx}_{\mathsf{Mint}}$ to L.
  4. Output $\perp$.

  The address set ADDR remains unchanged.

- $Q = (\textbf{Pour}, \mathsf{idx}_1^{\mathsf{old}}, \mathsf{idx}_2^{\mathsf{old}}, \mathsf{addr}_{\mathsf{pk},1}^{\mathsf{old}}, \mathsf{addr}_{\mathsf{pk},2}^{\mathsf{old}}, \mathsf{info}, v_1^{\mathsf{new}}, v_2^{\mathsf{new}}, \mathsf{addr}_{\mathsf{pk},1}^{\mathsf{new}}, \mathsf{addr}_{\mathsf{pk},2}^{\mathsf{new}}, v_{\mathsf{pub}})$

  1. Compute rt, the root of a Merkle tree over all coin commitments in L.

2. For each $i \in \{1, 2\}$:
   (a) Let $\mathsf{cm}_i^{\mathsf{old}}$ be the $\mathsf{idx}_i^{\mathsf{old}}$-th coin commitment in L.
   (b) Let tx_i be the mint/pour transaction in L that contains $\mathsf{cm}_i^{\mathsf{old}}$.
   (c) Let $\mathbf{c}_i^{\mathsf{old}}$ be the first coin in COIN with coin commitment $\mathsf{cm}_i^{\mathsf{old}}$.
   (d) Let $(\mathsf{addr}_{\mathsf{pk},i}^{\mathsf{old}}, \mathsf{addr}_{\mathsf{sk},i}^{\mathsf{old}})$ be the first key pair in ADDR with $\mathsf{addr}_{\mathsf{pk},i}^{\mathsf{old}}$ being $\mathbf{c}_i^{\mathsf{old}}$'s address.
   (e) Compute path_i, the authentication path from $\mathsf{cm}_i^{\mathsf{old}}$ to rt.
3. Compute $(\mathbf{c}_1^{\mathsf{new}}, \mathbf{c}_2^{\mathsf{new}}, \mathsf{tx}_{\mathsf{Pour}}) := \mathsf{Pour}(\mathsf{pp}, \mathsf{rt}, \mathbf{c}_1^{\mathsf{old}}, \mathbf{c}_2^{\mathsf{old}}, \mathsf{addr}_{\mathsf{sk},1}^{\mathsf{old}}, \mathsf{addr}_{\mathsf{sk},2}^{\mathsf{old}}, \mathsf{path}_1, \mathsf{path}_2, v_1^{\mathsf{new}}, v_2^{\mathsf{new}}, \mathsf{addr}_{\mathsf{pk},1}^{\mathsf{new}}, \mathsf{addr}_{\mathsf{pk},2}^{\mathsf{new}}, v_{\mathsf{pub}}, \mathsf{info})$.
4. Verify that $\mathsf{VerifyTransaction}(\mathsf{pp}, \mathsf{tx}_{\mathsf{Pour}}, L)$ outputs 1.
5. Add the coin $\mathbf{c}_1^{\mathsf{new}}$ to COIN.
6. Add the coin $\mathbf{c}_2^{\mathsf{new}}$ to COIN.
7. Add the pour transaction $\mathsf{tx}_{\mathsf{Pour}}$ to L.
8. Output $\perp$.

If any of the above operations fail, the output is $\perp$ (and $L, \mathsf{ADDR}, \mathsf{COIN}$ remain unchanged).

- $Q = (\mathbf{Receive}, \mathsf{addr}_{\mathsf{pk}})$

1. Look up $(\mathsf{addr}_{\mathsf{pk}}, \mathsf{addr}_{\mathsf{sk}})$ in ADDR. (If no such key pair is found, abort.)
2. Compute $(\mathbf{c}_1, \ldots, \mathbf{c}_n) \leftarrow \mathsf{Receive}(\mathsf{pp}, (\mathsf{addr}_{\mathsf{pk}}, \mathsf{addr}_{\mathsf{sk}}), L)$.
3. Add $\mathbf{c}_1, \ldots, \mathbf{c}_n$ to COIN.
4. Output $(\mathsf{cm}_1, \ldots, \mathsf{cm}_n)$, the corresponding coin commitments.

The ledger L and address set ADDR remain unchanged.

- $Q = (\mathbf{Insert}, \mathsf{tx})$

1. Verify that $\mathsf{VerifyTransaction}(\mathsf{pp}, \mathsf{tx}, L)$ outputs 1. (Else, abort.)
2. Add the mint/pour transaction tx to L.
3. Run **Receive** for all addresses $\mathsf{addr}_{\mathsf{pk}}$ in ADDR; this updates the COIN with any coins that might have been sent to honest parties via tx.

4. Output $\perp$.

The address set ADDR remains unchanged.

Remark. The oracle $\mathcal{O}^{\mathsf{DAP}}$ provides $\mathcal{A}$ with two ways to cause a pour transaction to be added to L. If $\mathcal{A}$ has already obtained address public keys $\mathsf{addr}_{\mathsf{pk},1}$ and $\mathsf{addr}_{\mathsf{pk},2}$ (via previous **CreateAddress** queries), then $\mathcal{A}$ can use a **Pour** query to elicit a pour transaction $\mathsf{tx}_{\mathsf{Pour}}$ (despite not knowing address secret keys $\mathsf{addr}_{\mathsf{sk},1}, \mathsf{addr}_{\mathsf{sk},2}$ corresponding to $\mathsf{addr}_{\mathsf{pk},1}, \mathsf{addr}_{\mathsf{pk},2}$). Alternatively, if $\mathcal{A}$ has himself generated both address public keys, then $\mathcal{A}$ knows corresponding address secret keys, and can invoke Pour "in his head" to obtain a pour transaction $\mathsf{tx}_{\mathsf{Pour}}$, which he can add to L by using an **Insert** query. In the first case, both addresses belong to honest users; in the second, both to $\mathcal{A}$.

But what about pour transactions where one address belongs to an honest user and one to $\mathcal{A}$? Such pour transactions might arise from MPC computations (e.g., to make matching donations). The ledger oracle $\mathcal{O}^{\mathsf{DAP}}$, as defined above, does not support such queries. While extending the definition is straightforward, for simplicity we leave handling such queries to future work.

B.2.1 Ledger indistinguishability

Ledger indistinguishability is characterized by an experiment $\mathsf{L\text{-}IND}$, which involves a polynomial-size adversary $\mathcal{A}$ attempting to break a given (candidate) DAP scheme.

Definition B.2.1. *Let* $\Pi = (\mathsf{Setup}, \mathsf{CreateAddress}, \mathsf{Mint}, \mathsf{Pour}, \mathsf{VerifyTransaction}, \mathsf{Receive})$ *be a (candidate) DAP scheme. We say that* Π *is* $\mathsf{L\text{-}IND}$ ***secure*** *if, for every* $\mathrm{poly}(\lambda)$*-size adversary* $\mathcal{A}$ *and sufficiently large* λ*,*

$$\mathsf{Adv}^{\mathsf{L\text{-}IND}}_{\Pi,\mathcal{A}}(\lambda) < \mathrm{negl}(\lambda) \ ,$$

where $\mathsf{Adv}^{\mathsf{L\text{-}IND}}_{\Pi,\mathcal{A}}(\lambda) := 2 \cdot \Pr[\mathsf{L\text{-}IND}(\Pi, \mathcal{A}, \lambda) = 1] - 1$ *is* $\mathcal{A}$*'s advantage in the* $\mathsf{L\text{-}IND}$ *experiment.*

We now describe the $\mathsf{L\text{-}IND}$ experiment mentioned above. Given a (candidate) DAP scheme Π, adversary $\mathcal{A}$, and security parameter λ, the (probabilistic) experiment $\mathsf{L\text{-}IND}(\Pi, \mathcal{A}, \lambda)$ consists of an interaction between $\mathcal{A}$ and a challenger $\mathcal{C}$, terminating with a binary output by $\mathcal{C}$.

At the beginning of the experiment, $\mathcal{C}$ samples $b \in \{0,1\}$ at random, samples $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$, and sends pp to $\mathcal{A}$; next, $\mathcal{C}$ initializes (using pp) two separate DAP oracles $\mathcal{O}_0^{\mathsf{DAP}}$ and $\mathcal{O}_1^{\mathsf{DAP}}$ (i.e., the two oracles have separate ledgers and internal tables).

The experiment proceeds in steps and, at each step, $\mathcal{C}$ provides to $\mathcal{A}$ two ledgers $(L_{\mathsf{Left}}, L_{\mathsf{Right}})$, where $L_{\mathsf{Left}} := L_b$ is the current ledger in $\mathcal{O}_b^{\mathsf{DAP}}$ and $L_{\mathsf{Right}} := L_{1-b}$ the one in $\mathcal{O}_{1-b}^{\mathsf{DAP}}$; then $\mathcal{A}$ sends to $\mathcal{C}$ a pair of queries (Q, Q'), which must be of the *same* type (i.e., one of **CreateAddress**, **Mint**, **Pour**, **Receive**, **Insert**). The challenger $\mathcal{C}$ acts differently depending on the query type, as follows.

- If the query type is **Insert**, $\mathcal{C}$ forwards Q to $\mathcal{O}_b^{\mathsf{DAP}}$, and Q' to $\mathcal{O}_{1-b}^{\mathsf{DAP}}$. This allows $\mathcal{A}$ to insert his own transactions directly in L_{Left} and L_{Right}.
- For any other query type, $\mathcal{C}$ first ensures that Q, Q' are *publicly consistent* (see below) and then forwards Q to $\mathcal{O}_0^{\mathsf{DAP}}$, and Q' to $\mathcal{O}_1^{\mathsf{DAP}}$; letting (a_0, a_1) be the two oracle answers, $\mathcal{C}$ replies to $\mathcal{A}$ with (a_b, a_{1-b}). This allows $\mathcal{A}$ to elicit behavior from honest users. However note that $\mathcal{A}$ does not know the bit b, and hence the mapping between $(L_{\mathsf{Left}}, L_{\mathsf{Right}})$ and (L_0, L_1); in other words, $\mathcal{A}$ does not know if he elicits behavior on (L_0, L_1) or on (L_1, L_0).

At the end of the experiment, $\mathcal{A}$ sends $\mathcal{C}$ a guess $b' \in \{0,1\}$. If $b = b'$, $\mathcal{C}$ outputs 1; else, $\mathcal{C}$ outputs 0.

Public consistency. As mentioned above, $\mathcal{A}$ sends $\mathcal{C}$ pairs of queries (Q, Q'), which must be of the same type and publicly consistent, a property that we now define. If Q, Q' are both of type **CreateAddress** or **Receive**, then they are always publicly consistent. In the special case of **CreateAddress** we require that both oracles generate the same address. If they are both of type **Mint**, then the minted value in Q must equal that in Q'. Finally, if they are both of type **Pour**, the following restrictions apply.

First, Q, Q' must be individually well-formed; namely, (i) the coin commitments referenced by Q (via the two indices $\mathsf{idx}_1^{\mathsf{old}}, \mathsf{idx}_2^{\mathsf{old}}$) must correspond to coins $\mathbf{c}_1^{\mathsf{old}}, \mathbf{c}_2^{\mathsf{old}}$ that appear in the ledger oracle's coin table COIN; (ii) the two coins $\mathbf{c}_1^{\mathsf{old}}, \mathbf{c}_2^{\mathsf{old}}$ must be unspent (i.e. their serial numbers must not appear in a valid pour transactions on the corresponding oracle's ledger); (iii) the address public

keys specified in Q must match those in $\mathbf{c}_1^{\mathsf{old}}, \mathbf{c}_2^{\mathsf{old}}$; and (iv) the balance equation must hold (i.e., $v_1^{\mathsf{new}} + v_2^{\mathsf{new}} + v_{\mathsf{pub}} = v_1^{\mathsf{old}} + v_2^{\mathsf{old}}$).

Furthermore, Q, Q' must be jointly consistent with respect to public information and $\mathcal{A}$'s view; namely: (i) the public values in Q and Q' must equal; (ii) the transaction strings in Q and Q' must equal; (iii) for each $i \in \{1, 2\}$, if the i-th recipient addresses in Q is not in ADDR (i.e., belongs to $\mathcal{A}$) then v_i^{new} in *both* Q and Q' must equal (and vice versa for Q'); and (iv) for each $i \in \{1, 2\}$, if the i-th index in Q references (in L_0) a coin commitment contained in a transaction that was posted via an **Insert** query, then the corresponding index in Q' must reference (in L_1) a coin commitment that also appears in a transaction posted via an **Insert** query and, moreover, v_i^{old} in *both* Q and Q' must equal (and vice versa for Q'). The challenger $\mathcal{C}$ learns v_i^{old} by looking-up the corresponding coin $\mathbf{c}_i^{\mathsf{old}}$ in the oracle's coin set COIN. (v) for each $i \in \{1, 2\}$ if the i-th index in Q must not reference a coin that has previously been spent.

B.2.2 Transaction non-malleability

Transaction non-malleability is characterized by an experiment $\mathsf{TR\text{-}NM}$, which involves a polynomial-size adversary $\mathcal{A}$ attempting to break a given (candidate) DAP scheme.

Definition B.2.2. *Let* $\Pi = (\mathsf{Setup}, \mathsf{CreateAddress}, \mathsf{Mint}, \mathsf{Pour}, \mathsf{VerifyTransaction}, \mathsf{Receive})$ *be a (candidate) DAP scheme. We say that* Π *is* $\mathsf{TR\text{-}NM}$ ***secure*** *if, for every* $\mathrm{poly}(\lambda)$*-size adversary* $\mathcal{A}$ *and sufficiently large* λ*,*

$$\mathsf{Adv}^{\mathsf{TR\text{-}NM}}_{\Pi,\mathcal{A}}(\lambda) < \mathrm{negl}(\lambda) \ ,$$

where $\mathsf{Adv}^{\mathsf{TR\text{-}NM}}_{\Pi,\mathcal{A}}(\lambda) := \Pr[\mathsf{TR\text{-}NM}(\Pi, \mathcal{A}, \lambda) = 1]$ *is* $\mathcal{A}$*'s advantage in the* $\mathsf{TR\text{-}NM}$ *experiment.*

We now describe the $\mathsf{TR\text{-}NM}$ experiment mentioned above. Given a (candidate) DAP scheme Π, adversary $\mathcal{A}$, and security parameter λ, the (probabilistic) experiment $\mathsf{TR\text{-}NM}(\Pi, \mathcal{A}, \lambda)$ consists of an interaction between $\mathcal{A}$ and a challenger $\mathcal{C}$, terminating with a binary output by $\mathcal{C}$.

At the beginning of the experiment, $\mathcal{C}$ samples $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$ and sends pp to $\mathcal{A}$; next,

$\mathcal{C}$ initializes a DAP oracle $\mathcal{O}^{\mathsf{DAP}}$ with pp and allows $\mathcal{A}$ to issue queries to $\mathcal{O}^{\mathsf{DAP}}$. At the end of the experiment, $\mathcal{A}$ sends $\mathcal{C}$ a pour transaction tx^*, and $\mathcal{C}$ outputs 1 if and only if the following conditions hold. Letting $\mathcal{T}$ be the set of pour transactions generated by $\mathcal{O}^{\mathsf{DAP}}$ in response to **Pour** queries, there exists $\mathsf{tx} \in \mathcal{T}$ such that: (i) $\mathsf{tx}^* \neq \mathsf{tx}$; (ii) $\mathsf{VerifyTransaction}(\mathsf{pp}, \mathsf{tx}^*, L') = 1$, where L' is the portion of the ledger preceding tx;[1] and (iii) a serial number revealed in tx^* is also revealed in tx.

B.2.3 BAL

BAL is characterized by an experiment BAL, which involves a polynomial-size adversary $\mathcal{A}$ attempting to break a given (candidate) DAP scheme.

Definition B.2.3. *Let* $\Pi = (\mathsf{Setup}, \mathsf{CreateAddress}, \mathsf{Mint}, \mathsf{Pour}, \mathsf{VerifyTransaction}, \mathsf{Receive})$ *be a (candidate) DAP scheme. We say that* Π *is* BAL ***secure*** *if, for every* $\mathrm{poly}(\lambda)$*-size adversary* $\mathcal{A}$ *and sufficiently large* λ*,*

$$\mathsf{Adv}^{\mathsf{BAL}}_{\Pi,\mathcal{A}}(\lambda) < \mathrm{negl}(\lambda) \ ,$$

where $\mathsf{Adv}^{\mathsf{BAL}}_{\Pi,\mathcal{A}}(\lambda) := \Pr[\mathsf{BAL}(\Pi, \mathcal{A}, \lambda) = 1]$ *is* $\mathcal{A}$*'s advantage in the* BAL *experiment.*

We now describe the BAL experiment mentioned above. Given a (candidate) DAP scheme Π, adversary $\mathcal{A}$, and security parameter λ, the (probabilistic) experiment $\mathsf{BAL}(\Pi, \mathcal{A}, \lambda)$ consists of an interaction between $\mathcal{A}$ and a challenger $\mathcal{C}$, terminating with a binary output by $\mathcal{C}$.

At the beginning of the experiment, $\mathcal{C}$ samples $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$, and sends pp to $\mathcal{A}$; next, $\mathcal{C}$ (using pp) initializes a DAP oracle $\mathcal{O}^{\mathsf{DAP}}$ and allows $\mathcal{A}$ to issue queries to $\mathcal{O}^{\mathsf{DAP}}$. At the conclusion of the experiment, $\mathcal{A}$ sends $\mathcal{C}$ a set of coins S_{coin}. Recalling that ADDR is the set of addresses returned by **CreateAddress** queries (i.e., addresses of "honest" users), $\mathcal{C}$ computes the following five quantities.

- v_{Unspent}, the total value of all spendable coins in S_{coin}. The challenger $\mathcal{C}$ can check if a coin $\mathbf{c} \in S_{\mathsf{coin}}$ is spendable as follows: mint a fresh coin $\mathbf{c}'$ of value 0 (via a **Mint** query) and check if a corresponding **Pour** query consuming $\mathbf{c}, \mathbf{c}'$ yields a pour transaction $\mathsf{tx}_{\mathsf{Pour}}$ that is valid.

[1] That is, L' is the longest ledger prefix that can be used to spend at least one of the coins spent in tx.

- v_{Mint}, the total value of all coins minted by $\mathcal{A}$. To compute v_{Mint}, the challenger $\mathcal{C}$ sums up the values of all coins that (i) were minted via **Mint** queries using addresses not in ADDR, or (ii) whose mint transactions were directly placed on the ledger via **Insert** queries.

- $v_{\mathsf{ADDR}\to\mathcal{A}}$, the total value payments received by $\mathcal{A}$ from addresses in ADDR. To compute $v_{\mathsf{ADDR}\to\mathcal{A}}$, the challenger $\mathcal{C}$ looks up all pour transactions placed on the ledger via **Pour** queries and sums up the values that were transferred to addresses not in ADDR.

- $v_{\mathcal{A}\to\mathsf{ADDR}}$, the total value of payments sent by $\mathcal{A}$ to addresses in ADDR. To compute $v_{\mathcal{A}\to\mathsf{ADDR}}$, the challenger $\mathcal{C}$ first deduces the set $S' \subseteq \mathsf{COIN}$ of all coins received by honest parties and then sums up the values of coins in S'. (Note that $\mathcal{C}$ can compute S' by selecting all coins in COIN that are both tied to an address in ADDR and arose from transactions placed on the ledger by **Insert** queries.)

- v_{Basecoin}, the total value of public outputs placed by $\mathcal{A}$ on the ledger. To compute v_{Basecoin}, the challenger $\mathcal{C}$ looks up all pour transactions placed on the ledger by **Insert** and sums up the corresponding v_{pub} values.

At the end of the experiment, $\mathcal{C}$ outputs 1 if $v_{\mathsf{Unspent}} + v_{\mathsf{Basecoin}} + v_{\mathcal{A}\to\mathsf{ADDR}} > v_{\mathsf{Mint}} + v_{\mathsf{ADDR}\to\mathcal{A}}$; else, $\mathcal{C}$ outputs 0.

Remark. There are two methods for $\mathcal{A}$ to spend more public-output money than he owns: (i) by directly inserting transactions on the ledger, and (ii) by asking honest parties to create such transactions. The first method is accounted for in the computation of v_{Basecoin}, while the second method is accounted for in the computation of $v_{\mathcal{A}\to\mathsf{ADDR}}$ (since $\mathcal{A}$ must first pay the honest party).

B.3 Proof of Theorem 3.4.1

We prove Theorem 3.4.1. We omit a formal proof of the completeness claim; one can verify that the DAP scheme's completeness follows, in a straightforward way, from the completeness of the

construction's building blocks. Next, we argue security via three separate proofs, respectively showing that our construction satisfies (i) ledger indistinguishability, (ii) transaction non-malleability, and (iii) balance.

B.3.1 Proof of ledger indistinguishability

We describe a simulation $\ni_{\mathsf{sim}}$ in which the adversary $\mathcal{A}$ interacts with a challenger $\mathcal{C}$, as in the L-IND experiment. However $\ni_{\mathsf{sim}}$ differs from the L-IND experiment in a critical way: all answers sent to $\mathcal{A}$ are computed *independently* of the bit b, so that $\mathcal{A}$'s advantage in $\ni_{\mathsf{sim}}$ is 0. The remainder of the proof is devoted to showing that $\mathsf{Adv}^{\mathsf{L\text{-}IND}}_{\Pi,\mathcal{A}}(\lambda)$ (i.e., $\mathcal{A}$'s advantage in the L-IND experiment) is at most negligibly different than $\mathcal{A}$'s advantage in $\ni_{\mathsf{sim}}$.

The simulation. The simulation $\ni_{\mathsf{sim}}$ works as follows. First, after sampling $b \in \{0,1\}$ at random, $\mathcal{C}$ samples $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$, with the following modification: the zk-SNARK keys are generated as $(\mathsf{pk}_{\mathtt{POUR}}, \mathsf{vk}_{\mathtt{POUR}}, \mathsf{trap}) \leftarrow \mathsf{Sim}(1^\lambda, C_{\mathtt{POUR}})$, to obtain the zero-knowledge trapdoor trap. Then, as in the L-IND experiment, $\mathcal{C}$ sends pp to $\mathcal{A}$, and then initializes two separate DAP oracles $\mathcal{O}_0^{\mathsf{DAP}}$ and $\mathcal{O}_1^{\mathsf{DAP}}$.

Afterwards, as in L-IND, $\ni_{\mathsf{sim}}$ proceeds in steps and, at each step, $\mathcal{C}$ provides to $\mathcal{A}$ two ledgers $(L_{\mathsf{Left}}, L_{\mathsf{Right}})$, where $L_{\mathsf{Left}} := L_b$ is the current ledger in $\mathcal{O}_b^{\mathsf{DAP}}$ and $L_{\mathsf{Right}} := L_{1-b}$ the one in $\mathcal{O}_{1-b}^{\mathsf{DAP}}$; then $\mathcal{A}$ sends to $\mathcal{C}$ a message (Q, Q'), which consist of two (publicly-consistent) queries of the same type. The challenger $\mathcal{C}$ acts differently depending on the query type, as follows.

- *Answering* **CreateAddress** *queries.* In this case, $Q = Q' =$ **CreateAddress**.

  To answer Q, $\mathcal{C}$ behaves as in L-IND, except for the following modification: after obtaining $(\mathsf{addr}_{\mathsf{pk}}, \mathsf{addr}_{\mathsf{sk}}) \leftarrow \mathsf{CreateAddress}(\mathsf{pp})$, $\mathcal{C}$ replaces a_{pk} in $\mathsf{addr}_{\mathsf{pk}}$ with a random string of the appropriate length; then, $\mathcal{C}$ stores $\mathsf{addr}_{\mathsf{sk}}$ in a table and returns $\mathsf{addr}_{\mathsf{pk}}$ to $\mathcal{A}$.

  Afterwards, $\mathcal{C}$ does the same for Q'.

- *Answering* **Mint** *queries.* In this case, $Q = (\mathbf{Mint}, v, \mathsf{addr}_{\mathsf{pk}})$ and $Q' = (\mathbf{Mint}, v, \mathsf{addr}'_{\mathsf{pk}})$.

  To answer Q, $\mathcal{C}$ behaves as in L-IND, except for the following modification: the Mint algorithm

computes the commitment k as $\mathsf{COMM}_r(\tau \| \rho)$, for a random string τ of the appropriate length, instead of as $\mathsf{COMM}_r(a_{\mathsf{pk}} \| \rho)$, where a_{pk} is the value specified in $\mathsf{addr}_{\mathsf{pk}}$.

Afterwards, $\mathcal{C}$ does the same for Q'.

- *Answering* **Pour** *queries.* In this case, Q and Q' both have the form $(\mathbf{Pour}, \mathsf{cm}_1^{\mathsf{old}}, \mathsf{cm}_2^{\mathsf{old}}, \mathsf{addr}_{\mathsf{pk},1}^{\mathsf{old}}, \mathsf{addr}_{\mathsf{pk},2}^{\mathsf{old}}, \mathsf{info}, v_1^{\mathsf{new}}, v_2^{\mathsf{new}}, \mathsf{addr}_{\mathsf{pk},1}^{\mathsf{new}}, \mathsf{addr}_{\mathsf{pk},2}^{\mathsf{new}}, v_{\mathsf{pub}}^{\mathsf{new}})$.

  To answer Q, $\mathcal{C}$ modifies the way some values are computed:

  1. Compute rt_i by accumulating all of the valid coin commitments on L_i.
  2. Set v_{pub} and info to the corresponding input values.
  3. For each $j \in \{1, 2\}$:
     (a) Sample a uniformly random $\mathsf{sn}_j^{\mathsf{old}}$.
     (b) If $\mathsf{addr}_{\mathsf{pk},j}^{\mathsf{new}}$ is an address generated by a previous query to **CreateAddress**, (i) sample a coin commitment $\mathsf{cm}_j^{\mathsf{new}}$ on a random input, (ii) run $\mathcal{K}_{\mathsf{enc}}(\mathsf{pp}_{\mathsf{enc}}) \to (\mathsf{pk}_{\mathsf{enc}}, \mathsf{sk}_{\mathsf{enc}})$ and compute $\mathbf{C}_j^{\mathsf{new}} := \mathcal{E}_{\mathsf{enc}}(\mathsf{pk}_{\mathsf{enc}}, r)$ for a random r of suitable length.
     (c) Otherwise, calculate $(\mathsf{cm}_i^{\mathsf{new}}, \mathbf{C}_i^{\mathsf{new}})$ as in the Pour algorithm.[2]
  4. Set h_1 and h_2 to be random strings of the appropriate length.
  5. Compute all remaining values as in the Pour algorithm
  6. The pour proof is computed as $\pi_{\mathtt{POUR}} := \mathsf{Sim}(\mathsf{trap}, x)$, where $x := (\mathsf{rt}, \mathsf{sn}_1^{\mathsf{old}}, \mathsf{sn}_2^{\mathsf{old}}, \mathsf{cm}_1^{\mathsf{new}}, \mathsf{cm}_2^{\mathsf{new}}, v_{\mathsf{pub}}, h_1, h_2)$.

  Afterwards, $\mathcal{C}$ does the same for Q'.

- *Answering* **Receive** *queries.* In this case, $Q = (\mathbf{Receive}, \mathsf{addr}_{\mathsf{pk}})$ and $Q' = (\mathbf{Receive}, \mathsf{addr}'_{\mathsf{pk}})$. The answer to each query proceeds as in the $\mathsf{L\text{-}IND}$ experiment.

- *Answering* **Insert** *queries.* In this case, $Q = (\mathbf{Insert}, \mathsf{tx})$ and $Q = (\mathbf{Insert}, \mathsf{tx}')$. The answer to each query proceeds as in the $\mathsf{L\text{-}IND}$ experiment.

In each of the above cases, the response to $\mathcal{A}$ is computed independently of the bit b. Thus, when $\mathcal{A}$ outputs a guess b', it must be the case that $\Pr[\, b = b' \,] = 1/2$, i.e., $\mathcal{A}$'s advantage in $\mathfrak{O}_{\mathsf{sim}}$ is 0.

[2]Note that by the restrictions of the experiment, the value v_i^{new} is identical between Q_{Left} and Q_{Right}.

Proof that the simulation is indistinguishable from the real experiment. We now describe a sequence of hybrid experiments $(\ni_{\mathsf{real}}, \ni_1, \ni_2, \ni_3, \ni_{\mathsf{sim}})$ in each of which a challenger $\mathcal{C}$ conducts a modification of the L-IND experiment with $\mathcal{A}$. We define $\ni_{\mathsf{real}}$ to be the original L-IND experiment, and $\ni_{\mathsf{sim}}$ to be the simulation described above.

With a slight abuse of notation, given experiment $\ni$, we define $\mathsf{Adv}^{\ni}$ to be the absolute value of the difference between (i) the L-IND advantage of $\mathcal{A}$ in $\ni$ and (ii) the L-IND advantage of $\mathcal{A}$ in $\ni_{\mathsf{real}}$. Also, let

- $q_{\mathbf{CA}}$ be the total number of **CreateAddress** queries issued by $\mathcal{A}$,
- $q_{\mathbf{P}}$ be the total number of **Pour** queries issued by $\mathcal{A}$, and
- $q_{\mathbf{M}}$ be the total number of **Mint** queries issued by $\mathcal{A}$.

Finally, define $\mathsf{Adv}^{\mathsf{Enc}}$ to be $\mathcal{A}$'s advantage in Enc's IND-CCA and IK-CCA experiments, $\mathsf{Adv}^{\mathsf{PRF}}$ to be $\mathcal{A}$'s advantage in distinguishing the pseudorandom function PRF from a random one, and $\mathsf{Adv}^{\mathsf{COMM}}$ to be $\mathcal{A}$'s advantage against the hiding property of COMM.

We now describe each of the hybrid experiments.

- **Experiment $\ni_1$.** The experiment $\ni_1$ modifies $\ni_{\mathsf{real}}$ by simulating the zk-SNARKs. More precisely, we modify $\ni_{\mathsf{real}}$ so that $\mathcal{C}$ simulates each zk-SNARK proof, as follows. At the beginning of the experiment, instead of invoking $\mathsf{KeyGen}(1^\lambda, C_{\mathtt{POUR}})$, $\mathcal{C}$ invokes $\mathsf{Sim}(1^\lambda, C_{\mathtt{POUR}})$ and obtains $(\mathsf{pk}_{\mathtt{POUR}}, \mathsf{vk}_{\mathtt{POUR}}, \mathsf{trap})$. At each subsequent invocation of the Pour algorithm, $\mathcal{C}$ computes $\pi_{\mathtt{POUR}} \leftarrow \mathsf{Sim}(\mathsf{trap}, x)$, without using any witnesses, instead of using Prove. Since the zk-SNARK system is perfect zero knowledge, the distribution of the simulated $\pi_{\mathtt{POUR}}$ is identical to that of the proofs computed in $\ni_{\mathsf{real}}$. Hence $\mathsf{Adv}^{\ni_1} = 0$.

- **Experiment $\ni_2$.** The experiment $\ni_2$ modifies $\ni_1$ by replacing the ciphertexts in a pour transaction by encryptions of random strings. More precisely, we modify $\ni_1$ so that, each time $\mathcal{A}$ issues a **Pour** query where one of the output addresses $(\mathsf{addr}^{\mathsf{new}}_{\mathsf{pk},1}, \mathsf{addr}^{\mathsf{new}}_{\mathsf{pk},2})$ is in the set of addresses previously generated by a **CreateAddress** query, the two ciphertexts $\mathbf{C}^{\mathsf{new}}_1, \mathbf{C}^{\mathsf{new}}_2$ are generated as follows:

(i) $(\mathsf{pk}^{\mathsf{new}}_{\mathsf{enc}}, \mathsf{sk}^{\mathsf{new}}_{\mathsf{enc}}) \leftarrow \mathcal{K}_{\mathsf{enc}}(\mathsf{pp}_{\mathsf{enc}})$; (ii) for each $j \in \{1,2\}$, $\mathbf{C}^{\mathsf{new}}_j := \mathcal{E}_{\mathsf{enc}}(\mathsf{pk}^{\mathsf{new}}_{\mathsf{enc},j}, r)$ where r is a message sampled uniformly from the plaintext space of the encryption scheme. By Lemma B.3.1 (see below), $|\mathsf{Adv}^{\Game_2} - \mathsf{Adv}^{\Game_1}| \leq 4 \cdot q_{\mathbf{P}} \cdot \mathsf{Adv}^{\mathsf{Enc}}$.

- **Experiment $\Game_3$.** The experiment $\Game_3$ modifies $\Game_2$ by replacing all PRF-generated values with random strings. More precisely, we modify $\Game_2$ so that:
  - each time $\mathcal{A}$ issues a **CreateAddress** query, the value a_{pk} within the returned $\mathsf{addr}_{\mathsf{pk}}$ is substituted with a random string of the same length;
  - each time $\mathcal{A}$ issues a **Pour** query, each of the serial numbers $\mathsf{sn}^{\mathsf{old}}_1, \mathsf{sn}^{\mathsf{old}}_2$ in $\mathsf{tx}_{\mathsf{Pour}}$ is substituted with a random string of the same length, and h_{info} with a random string of the same length.

  By Lemma B.3.2 (see below), $|\mathsf{Adv}^{\Game_3} - \mathsf{Adv}^{\Game_2}| \leq q_{\mathbf{CA}} \cdot \mathsf{Adv}^{\mathsf{PRF}}$.

- **Experiment $\Game_{\mathsf{sim}}$.** The experiment $\Game_{\mathsf{sim}}$ is already described above. For comparison, we explain how it differs from $\Game_3$: the coin commitments are replaced with commitments to random inputs. More precisely, we modify $\Game_3$ so that:
  - each time $\mathcal{A}$ issues a **Mint** query, the coin commitment cm in $\mathsf{tx}_{\mathsf{Mint}}$ is substituted with a commitment to a random input; and
  - each time $\mathcal{A}$ issues a **Pour** query, then, for each $j \in \{1,2\}$, if the output address $\mathsf{addr}^{\mathsf{new}}_{\mathsf{pk},j}$ is in the set of addresses previously generated by an **CreateAddress** query, $\mathsf{cm}^{\mathsf{new}}_j$ is substituted with a commitment to a random input.

  By Lemma B.3.3 (see below), $|\mathsf{Adv}^{\Game_{\mathsf{sim}}} - \mathsf{Adv}^{\Game_3}| \leq (q_{\mathbf{M}} + 4 \cdot q_{\mathbf{P}}) \cdot \mathsf{Adv}^{\mathsf{COMM}}$.

As argued above, the responses provided to $\mathcal{A}$ in $\Game_{\mathsf{sim}}$ are independent of the bit b, so that $\mathsf{Adv}^{\Game_{\mathsf{sim}}} = 0$. Then, by summing over $\mathcal{A}$'s advantages in the hybrid experiments, we can bound $\mathcal{A}$'s advantage in $\Game_{\mathsf{real}}$ by

$$\mathsf{Adv}^{\mathsf{L\text{-}IND}}_{\Pi,\mathcal{A}}(\lambda) \leq 4 \cdot q_{\mathbf{P}} \cdot \mathsf{Adv}^{\mathsf{Enc}} + q_{\mathbf{CA}} \cdot \mathsf{Adv}^{\mathsf{PRF}} + (q_{\mathbf{M}} + 4 \cdot q_{\mathbf{P}}) \cdot \mathsf{Adv}^{\mathsf{COMM}} \ ,$$

which is negligible in λ. This concludes the proof of ledger indistinguishability. Below, we sketch

proofs for the lemmas used above (Lemma B.3.1, Lemma B.3.2, and Lemma B.3.3).

Lemma B.3.1. *Let* $\mathsf{Adv}^{\mathsf{Enc}}$ *be the maximum of:*

- $\mathcal{A}$*'s advantage in the* IND-CCA *experiment against the encryption scheme* Enc*, and*
- $\mathcal{A}$*'s advantage in the* IK-CCA *experiment against the encryption scheme* Enc*.*

Then after $q_{\mathbf{P}}$ **Pour** *queries,* $|\mathsf{Adv}^{\eth_2} - \mathsf{Adv}^{\eth_1}| \leq 4 \cdot q_{\mathbf{P}} \cdot \mathsf{Adv}^{\mathsf{Enc}}$*.*

Proof sketch. Define $\epsilon := \mathsf{Adv}^{\eth_2} - \mathsf{Adv}^{\eth_1}$. Using $\mathcal{A}$, we first show how to construct a solver with advantage $\geq \frac{\epsilon}{2 \cdot q_{\mathbf{P}}}$ in the IK-CCA or IND-CCA experiments. We use a hybrid $\mathbf{H}$, intermediate between $\eth_1$ and $\eth_2$; concretely, $\mathbf{H}$ modifies $\eth_1$ so that each ciphertext (where the corresponding public key appears in the set generated by a **CreateAddress** query) is replaced with the encryption of the same plaintext, but under a new, random public key generated via the $\mathcal{K}_{\mathsf{enc}}$ algorithm. (For comparison, $\eth_2$ modifies $\mathbf{H}$ so that each plaintext is replaced with a random plaintext drawn from the plaintext space.) We now argue that $\mathcal{A}$'s advantage in distinguishing $\mathbf{H}$ and $\eth_1$ is at most $2 \cdot q_{\mathbf{P}} \cdot \mathsf{Adv}^{\mathsf{Enc}}$, and so is for distinguishing $\eth_2$ and $\mathbf{H}$. Overall, we deduce that $|\mathsf{Adv}^{\eth_2} - \mathsf{Adv}^{\eth_1}| \leq 4 \cdot q_{\mathbf{P}} \cdot \mathsf{Adv}^{\mathsf{Enc}}$.

First, we discuss $\mathbf{H}$ and $\eth_1$. For some $j \in \{1, \ldots, q_{\mathbf{CA}}\}$, when $\mathcal{A}$ makes the j-th query of the form **CreateAddress**, query the IK-CCA challenger to obtain two public keys $(\mathsf{pk}_{\mathsf{enc},0}, \mathsf{pk}_{\mathsf{enc},1})$ and return $\mathsf{pk}_{\mathsf{enc}} := \mathsf{pk}_{\mathsf{enc},0}$ in the response to $\mathcal{A}$. At the time $\mathcal{A}$ issues a **Pour** query that results in the i-th ciphertext $\mathbf{C}_i$ being encrypted under $\mathsf{pk}_{\mathsf{enc}}$, query the IK-CCA challenger on the corresponding plaintext m and receive $\mathbf{C}^* = \mathcal{E}_{\mathsf{enc}}(\mathsf{pk}_{\mathsf{enc},\bar{b}}, m)$ where $\bar{b}$ is the bit chosen by the IK-CCA challenger. Substitute $\mathbf{C}_i := \mathbf{C}^*$ and write the resulting $\mathsf{tx}_{\mathsf{Pour}}$ to the Ledger. When $\mathcal{A}$ outputs b' we return this guess as our guess in the IK-CCA experiment. We note that when $\bar{b} = 0$ then $\mathcal{A}$'s view of the interaction is distributed identically to that of $\eth_1$, and when $\bar{b}$ is 1 then $\mathcal{A}$'s view represents an intermediate hybrid where one key has been substituted. By a standard hybrid argument over each of the $2 \cdot q_{\mathbf{P}}$ ciphertexts, we note that over the random coins of the experiment, our solver must succeed in the IK-CCA experiment with advantage $\geq \frac{\epsilon}{2 \cdot q_{\mathbf{P}}}$. If we assume a maximum adversarial advantage $\mathsf{Adv}^{\mathsf{Enc}}$ against the IK-CCA experiment for the encryption scheme, then we get that

$\left|\mathsf{Adv}^{\mathbf{H}} - \mathsf{Adv}^{\ni_2}\right| \leq 2 \cdot q_{\mathbf{P}} \cdot \mathsf{Adv}^{\mathsf{Enc}}$.

Next, we discuss $\ni_2$ and $\mathbf{H}$; the argument is similar to the above one. This time, rather than replacing the key used to encrypt, we replace the plaintext with a random message drawn from the plaintext space; this final distribution is the same as in $\ni_2$. We omit the formal description of the resulting $\mathsf{IND\text{-}CCA}$ solver (which essentially follows the pattern above), and simply note that $\left|\mathsf{Adv}^{\ni_2} - \mathsf{Adv}^{\mathbf{H}}\right| \leq 2 \cdot q_{\mathbf{P}} \cdot \mathsf{Adv}^{\mathsf{Enc}}$. $\square$

Lemma B.3.2. *Let* $\mathsf{Adv}^{\mathsf{PRF}}$ *be* $\mathcal{A}$*'s advantage in distinguishing the pseudorandom function* PRF *from a random function. Then, after* $q_{\mathbf{CA}}$ **CreateAddress** *queries,* $|\mathsf{Adv}^{\ni_3} - \mathsf{Adv}^{\ni_2}| \leq q_{\mathbf{CA}} \cdot \mathsf{Adv}^{\mathsf{PRF}}$.

Proof sketch. We first describe a hybrid $\mathbf{H}$, intermediate between $\ni_2$ and $\ni_3$, in which all values computed using the first (rather than all) oracle-generated key a_{sk} are replaced with random strings. Then, we show that $\mathcal{A}$'s advantage in distinguishing between $\mathbf{H}$ and $\ni_2$ is at most $\mathsf{Adv}^{\mathsf{PRF}}$. Finally, we extend the argument to all $q_{\mathbf{CA}}$ oracle-generated keys (corresponding to what happens in $\ni_3$).

We now describe $\mathbf{H}$. On receiving $\mathcal{A}$'s first **CreateAddress** query, replace the public address $\mathsf{addr}_{\mathsf{pk}} = (a_{\mathsf{pk}}, \mathsf{pk}_{\mathsf{enc}})$ with $\mathsf{addr}_{\mathsf{pk}} = (\tau, \mathsf{pk}_{\mathsf{enc}})$ where τ is a random string of the appropriate length. On each subsequent **Pour** query, for each $i \in \{1, 2\}$, if $\mathsf{addr}^{\mathsf{old}}_{\mathsf{pk},i} = \mathsf{addr}_{\mathsf{pk}}$ then:

1. in the output $\mathsf{tx}_{\mathsf{Pour}}$, replace $\mathsf{sn}^{\mathsf{old}}_i$ with a random string of appropriate length;
2. in the output $\mathsf{tx}_{\mathsf{Pour}}$, replace each of h_1, h_2 with a random string of appropriate length.
3. simulate the zk-SNARK proof $\pi_{\mathtt{POUR}}$ for the new transaction.

Note that the above modifications do not affect the computation of the zk-SNARK proof $\pi_{\mathtt{POUR}}$, because $\pi_{\mathtt{POUR}}$ is simulated with the help of a trapdoor.

We now argue that $\mathcal{A}$'s advantage in distinguishing between $\mathbf{H}$ and $\ni_2$ is at most $\mathsf{Adv}^{\mathsf{PRF}}$. Let a_{sk} be the random, secret seed for PRF generated by the oracle in answering the first **CreateAddress** query. In $\ni_2$ (as in $\ni_{\mathsf{real}}$):

- $a_{\mathsf{pk}} := \mathsf{PRF}^{\mathrm{addr}}_{a_{\mathsf{sk}}}(0)$;
- for each $i \in \{1, 2\}$, $\mathsf{sn}_i := \mathsf{PRF}^{\mathrm{sn}}_{a_{\mathsf{sk}}}(\rho)$ for a random (and not previously used) ρ

- for each $i \in \{1, 2\}$, $h_i := \mathsf{PRF}^{\mathrm{pk}}_{a_{\mathsf{sk}}}(i \| h_{\mathsf{Sig}})$ and, with overwhelming probability, h_{Sig} is unique.

Moreover, each of $\mathsf{PRF}^{\mathrm{addr}}_{a_{\mathsf{sk}}}, \mathsf{PRF}^{\mathrm{sn}}_{a_{\mathsf{sk}}}, \mathsf{PRF}^{\mathrm{pk}}_{a_{\mathsf{sk}}}$ are constructed from $\mathsf{PRF}_{a_{\mathsf{sk}}}$ as specified in Section 3.4.1. Note that, with overwhelming probability, no two calls to $\mathsf{PRF}_{a_{\mathsf{sk}}}$ are made on the same input. First, even identical inputs passed to $\mathsf{PRF}^{\mathrm{addr}}_{a_{\mathsf{sk}}}, \mathsf{PRF}^{\mathrm{sn}}_{a_{\mathsf{sk}}}, \mathsf{PRF}^{\mathrm{pk}}_{a_{\mathsf{sk}}}$ produce different underlying calls to $\mathsf{PRF}_{a_{\mathsf{sk}}}$. Second, within each construction, there is exactly one call to $\mathsf{PRF}^{\mathrm{addr}}_{a_{\mathsf{sk}}}$, and the calls to $\mathsf{PRF}^{\mathrm{sn}}_{a_{\mathsf{sk}}}$ are each by definition unique. Finally, with overwhelming probability, the calls to $\mathsf{PRF}^{\mathrm{pk}}_{a_{\mathsf{sk}}}$ from different transactions each reference a distinct digest h_{Sig}, and, within a given transaction, the two calls each begin with a distinct prefix.

Now let $\mathcal{O}$ be an oracle that implements either $\mathsf{PRF}_{a_{\mathsf{sk}}}$ or a random function. We show that if $\mathcal{A}$ distinguishes $\mathbf{H}$ from $\ni_2$ with probability ϵ, then we can construct a distinguisher for the two cases of $\mathcal{O}$. In either case we use $\mathcal{O}$ to generate all values computed using $\mathsf{PRF}^{\mathrm{addr}}_{a_{\mathsf{sk}}}, \mathsf{PRF}^{\mathrm{sn}}_{a_{\mathsf{sk}}}, \mathsf{PRF}^{\mathrm{pk}}_{a_{\mathsf{sk}}}$. Clearly, when $\mathcal{O}$ implements $\mathsf{PRF}_{a_{\mathsf{sk}}}$, the distribution of the experiment is identical to $\ni_2$; instead, when $\mathcal{O}$ implements a random function, the distribution of the experiment is identical to $\mathbf{H}$. Thus, $\mathcal{A}$'s advantage is at most $\mathsf{Adv}^{\mathsf{PRF}}$.

Finally, by a standard hybrid argument, we extend the above to all $q_{\mathbf{CA}}$ oracle-generated addresses; then, $\mathcal{A}$'s differential distinguishing advantage is at most $q_{\mathbf{CA}} \cdot \mathsf{Adv}^{\mathsf{PRF}}$. Because this final hybrid is equal to $\ni_3$, we deduce that $|\mathsf{Adv}^{\ni_3} - \mathsf{Adv}^{\ni_2}| \leq q_{\mathbf{CA}} \cdot \mathsf{Adv}^{\mathsf{PRF}}$. $\square$

Lemma B.3.3. *Let* $\mathsf{Adv}^{\mathsf{COMM}}$ *be* $\mathcal{A}$*'s advantage against the hiding property of* COMM*. After* $q_{\mathbf{M}}$ **Mint** *queries and* $q_{\mathbf{P}}$ **Pour** *queries,* $|\mathsf{Adv}^{\ni_{\mathsf{sim}}} - \mathsf{Adv}^{\ni_3}| \leq (q_{\mathbf{M}} + 4 \cdot q_{\mathbf{P}}) \cdot \mathsf{Adv}^{\mathsf{COMM}}$.

Proof sketch. We only provide a short sketch, because the structure of the argument is similar to the one used to prove Lemma B.3.2 above.

For the first **Mint** or **Pour** query, replace the "internal" commitment $k := \mathsf{COMM}_r(a_{\mathsf{pk}} \| \rho)$ with a random string of the appropriate length. Since ρ is random (and unique), then $\mathcal{A}$'s advantage in distinguishing this modified experiment from $\ni_2$ is at most $\mathsf{Adv}^{\mathsf{COMM}}$. Then, if we similarly modify all $q_{\mathbf{M}}$ **Mint** queries and all $q_{\mathbf{P}}$ **Pour** queries, by replacing the resulting $q_{\mathbf{M}} + 2 \cdot q_{\mathbf{P}}$ internal

commitments with random strings, we can bound $\mathcal{A}$'s advantage by $(q_{\mathbf{M}} + 2 \cdot q_{\mathbf{P}}) \cdot \mathsf{Adv}^{\mathsf{COMM}}$.

Next, in a similar vein, if replace the coin commitment in the first **Pour** with a commitment to a random value, then $\mathcal{A}$'s advantage in distinguishing this modified experiment from the above one is at most $\mathsf{Adv}^{\mathsf{COMM}}$. Then, if we similarly modify all $q_{\mathbf{P}}$ **Pour** queries, by replacing the resulting $2 \cdot q_{\mathbf{P}}$ coin commitments with random strings, we obtain the experiment $\ni_{\mathsf{sim}}$, and deduce that $|\mathsf{Adv}^{\ni_{\mathsf{sim}}} - \mathsf{Adv}^{\ni_3}| \leq (q_{\mathbf{M}} + 4 \cdot q_{\mathbf{P}}) \cdot \mathsf{Adv}^{\mathsf{COMM}}$. $\square$

B.3.2 Proof of transaction non-malleability

Letting $\mathcal{T}$ be the set of pour transactions generated by $\mathcal{O}^{\mathsf{DAP}}$ in response to **Pour** queries, recall that $\mathcal{A}$ wins the TR-NM experiment whenever it outputs tx^* such that there exists $\mathsf{tx}' \in \mathcal{T}$ such that: (i) $\mathsf{tx}^* \neq \mathsf{tx}'$; (ii) $\mathsf{VerifyTransaction}(\mathsf{pp}, \mathsf{tx}^*, L') = 1$, where L' is the portion of the ledger preceding tx'; and (iii) a serial number revealed in tx^* is also revealed in tx'. Being a pour transaction, tx^* has the form $(\mathsf{rt}, \mathsf{sn}_1^{\mathsf{old}}, \mathsf{sn}_2^{\mathsf{old}}, \mathsf{cm}_1^{\mathsf{new}}, \mathsf{cm}_2^{\mathsf{new}}, v_{\mathsf{pub}}, \mathsf{info}, *)$, where $* := (\mathsf{pk}_{\mathsf{sig}}, h_1, h_2, \pi_{\texttt{POUR}}, \mathbf{C}_1, \mathbf{C}_2, \sigma)$; set $h_{\mathsf{Sig}} := \mathsf{CRH}(\mathsf{pk}_{\mathsf{sig}})$. Let $\mathsf{pk}'_{\mathsf{sig}}$ be the corresponding public key in tx' and set $h'_{\mathsf{Sig}} := \mathsf{CRH}(\mathsf{pk}'_{\mathsf{sig}})$.

Define $\epsilon := \mathsf{Adv}^{\mathsf{TR\text{-}NM}}_{\Pi,\mathcal{A}}(\lambda)$, and let $\mathcal{Q}_{\mathbf{CA}} = \{a_{\mathsf{sk},1}, \ldots, a_{\mathsf{sk},q_{\mathbf{CA}}}\}$ be the set of internal address keys created by $\mathcal{C}$ in response to $\mathcal{A}$'s **CreateAddress** queries. Let $\mathcal{Q}_{\mathbf{P}} = (\mathsf{pk}_{\mathsf{sig},1}, \ldots, \mathsf{pk}_{\mathsf{sig},q_{\mathbf{P}}})$ be the set of signature public keys created by $\mathcal{C}$ in response to $\mathcal{A}$'s **Pour** queries. We decompose the event in which $\mathcal{A}$ wins into the following four disjoint events.

- $\textsc{Event}_{\mathsf{sig}}$: $\mathcal{A}$ wins, and there is $\mathsf{pk}''_{\mathsf{sig}} \in \mathcal{Q}_{\mathbf{P}}$ such that $\mathsf{pk}_{\mathsf{sig}} = \mathsf{pk}''_{\mathsf{sig}}$.
- $\textsc{Event}_{\mathsf{col}}$: $\mathcal{A}$ wins, the above event does not occur, and there is $\mathsf{pk}''_{\mathsf{sig}} \in \mathcal{Q}_{\mathbf{P}}$ such that $h_{\mathsf{Sig}} = \mathsf{CRH}(\mathsf{pk}''_{\mathsf{sig}})$.
- $\textsc{Event}_{\mathsf{mac}}$: $\mathcal{A}$ wins, the above two events do not occur, and $h_i = \mathsf{PRF}^{\mathrm{pk}}_a(i \| h_{\mathsf{Sig}})$ for some $i \in \{1, 2\}$ and $a \in \mathcal{Q}_{\mathbf{CA}}$.
- $\textsc{Event}_{\mathsf{key}}$: $\mathcal{A}$ wins, the above three events do not occur, and $h_i \neq \mathsf{PRF}^{\mathrm{pk}}_a(i \| h_{\mathsf{Sig}})$ for all $i \in \{1, 2\}$ and $a \in \mathcal{Q}_{\mathbf{CA}}$.

Clearly, $\epsilon = \Pr[\textsc{Event}_{\mathsf{sig}}] + \Pr[\textsc{Event}_{\mathsf{col}}] + \Pr[\textsc{Event}_{\mathsf{key}}] + \Pr[\textsc{Event}_{\mathsf{mac}}]$. Hence, to show that

ϵ is negligible in λ, it suffices to argue that each of these probabilities is negligible in λ.

Bounding the probability of $\text{Event}_{\mathsf{sig}}$. Define $\epsilon_1 := \Pr[\text{EVENT}_{\mathsf{sig}}]$. Let σ be the signature in tx^*, and σ'' be the signature in the first pour transaction $\mathsf{tx}'' \in \mathcal{T}$ that contains $\mathsf{pk}''_{\mathsf{sig}}$. When $\text{EVENT}_{\mathsf{sig}}$ occurs, since $\mathsf{pk}_{\mathsf{sig}} = \mathsf{pk}''_{\mathsf{sig}}$, the two signatures are with respect to the same public key. Moreover, since tx^* is valid, $\mathcal{V}_{\mathsf{sig}}(\mathsf{pk}_{\mathsf{sig}}, m, \sigma) = 1$ where m is everything in tx^* but for σ. Let m'' consist of all elements in tx'' but for σ''. Observe that whenever $\mathsf{tx}^* \neq \mathsf{tx}''$ we also have $(m, \sigma) \neq (m'', \sigma'')$. We use this fact below to show that $\mathcal{A}$ forges a signature with non-negligible probability.

First, we argue that, conditioned on $\text{EVENT}_{\mathsf{sig}}$, $\mathsf{tx}^* \neq \mathsf{tx}''$ with overwhelming probability; we do so by way of contradiction. First, since $\mathcal{A}$ wins, by definition there is $\mathsf{tx}' \in \mathcal{T}$ such that $\mathsf{tx}^* \neq \mathsf{tx}'$ and yet each of tx^* and tx' share one serial number. Therefore: (i) $\mathsf{tx}^* \neq \mathsf{tx}'$; and (ii) if $\mathsf{tx}^* = \mathsf{tx}''$ then tx'' and tx' also share a serial number. However the probability that tx' and tx'' share a serial number is bounded by the probability $\tilde{p}$ that $\mathcal{T}$ contains two transactions that share the same serial number. Because each serial number is computed as $\mathsf{PRF}^{\text{sn}}_{a_{\mathsf{sk}}}(\rho)$, where ρ is random, $\tilde{p}$ is negligible. We conclude that $\mathsf{tx}^* \neq \mathsf{tx}''$ with all but negligible probability.

Next, we describe an algorithm $\mathcal{B}$, which uses $\mathcal{A}$ as a subroutine, that wins the SUF-1CMA game against Sig with probability $\epsilon_1 / q_{\mathbf{P}}$. After receiving a verification key $\mathsf{pk}''_{\mathsf{sig}}$ from the SUF-1CMA challenger, the algorithm $\mathcal{B}$ performs the following steps.

1. $\mathcal{B}$ selects a random index $j \leftarrow \{1, \ldots, q_{\mathbf{P}}\}$.
2. $\mathcal{B}$ conducts the TR-NM experiment with $\mathcal{A}$, except that, when $\mathcal{A}$ issues the j-th **Pour** query, $\mathcal{B}$ executes Pour as usual, but modifies the resulting pour transaction tx'' as follows: (i) it substitutes $\mathsf{pk}''_{\mathsf{sig}}$ for the signature public key in tx''; (ii) it queries the SUF-1CMA challenger to obtain σ'' on the appropriate message m''; and (iii) it substitutes σ'' for the signature in tx''.
3. When $\mathcal{A}$ outputs tx^*, $\mathcal{B}$ looks into tx^* to obtain $\mathsf{pk}_{\mathsf{sig}}$, m, and σ.
4. If $\mathsf{pk}_{\mathsf{sig}} \neq \mathsf{pk}''_{\mathsf{sig}}$ then $\mathcal{B}$ aborts; otherwise $\mathcal{B}$ outputs (m, σ) as a forgery for Sig.

Note that tx'' has the same distribution has an "untampered" pour transaction; thus, all transactions returned to $\mathcal{A}$ are distributed as in the TR-NM experiment. Since the index j is selected at random,

$\mathcal{B}$ succeeds in the experiment with probability at least $\epsilon_1/q_{\mathbf{P}}$. Because Sig is $\mathsf{SUF\text{-}1CMA}$, ϵ_1 must be negligible in λ.

Bounding the probability of $\mathbf{Event_{col}}$. Define $\epsilon_2 := \Pr[\,\textsc{Event}_{\mathsf{col}}\,]$. When $\textsc{Event}_{\mathsf{col}}$ occurs, $\mathcal{A}$ receives a transaction tx' containing a public key $\mathsf{pk}''_{\mathsf{sig}}$, and subsequently outputs a transaction tx^* containing a public key $\mathsf{pk}_{\mathsf{sig}}$ such that (i) $\mathsf{pk}_{\mathsf{sig}} \neq \mathsf{pk}''_{\mathsf{sig}}$, but (ii) $\mathsf{CRH}(\mathsf{pk}_{\mathsf{sig}}) = \mathsf{CRH}(\mathsf{pk}'_{\mathsf{sig}})$. In particular, $\mathcal{A}$ finds collisions for CRH with probability ϵ_2. Because CRH is collision resistant, ϵ_2 must be negligible in λ.

Bounding the probability of $\mathbf{Event_{mac}}$. Define $\epsilon_3 := \Pr[\,\textsc{Event}_{\mathsf{mac}}\,]$. We first define an experiment $\beth_1$, which modifies the $\mathsf{TR\text{-}NM}$ experiment as follows. When $\mathcal{C}$ samples $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$, the sub-call to $(\mathsf{pk}_{\mathtt{POUR}}, \mathsf{vk}_{\mathtt{POUR}}) \leftarrow \mathsf{KeyGen}(1^\lambda, C_{\mathtt{POUR}})$ is replaced by $(\mathsf{pk}_{\mathtt{POUR}}, \mathsf{vk}_{\mathtt{POUR}}, \mathsf{trap}) \leftarrow \mathsf{Sim}(1^\lambda, C_{\mathtt{POUR}})$, so to obtain the zero-knowledge trapdoor trap. Afterwards, each time $\mathcal{A}$ issues a **Pour** query, $\mathcal{C}$ replaces the zk-SNARK proof in the resulting pour transaction with a simulated proof, obtained by running $\mathsf{Sim}(\mathsf{trap}, x)$ for an appropriate input x. Because the zk-SNARK is perfect zero knowledge, $\Pr[\,\textsc{Event}_{\mathsf{mac}}\,] = \epsilon_3$ in the $\beth_1$ experiment as well.

Assume by way of contradiction that ϵ_3 is non-negligible. We now show how to construct an attacker $\mathcal{B}$, which uses $\mathcal{A}$ as a subroutine, that distinguishes PRF from a random function RAND with non-negligible probability. The algorithm $\mathcal{B}$, which has access either to $\mathcal{O} = \mathsf{PRF}$ or $\mathcal{O} = \mathsf{RAND}$, "interfaces" between $\mathcal{A}$ and $\mathcal{C}$ in the experiment $\beth_1$ above, as follows.

1. First, $\mathcal{B}$ selects a random index $j \leftarrow \{1, \ldots, q_{\mathbf{CA}}\}$, which identifies $a_{\mathsf{sk},j} \in \mathcal{Q}_{\mathbf{CA}}$.
2. Next, $\mathcal{B}$ uses the oracle $\mathcal{O}$ instead of $\mathsf{PRF}_{a_{\mathsf{sk},j}}$, i.e., anytime a value needs to be computed depending on $\mathsf{PRF}_{a_{\mathsf{sk},j}}(z)$, for some z, $\mathcal{O}(z)$ is used instead. (For instance, the public address key $a_{\mathsf{pk},j}$ is one such value.)
3. Finally, after $\mathcal{A}$ outputs tx^*:
   (a) if $\mathcal{O}$ has been previously evaluated the expression "$\mathsf{PRF}^{\mathrm{pk}}_{a_{\mathsf{sk},j}}(i \| h_{\mathsf{Sig}})$" using $\mathcal{O}$, $\mathcal{B}$ aborts and outputs 1;
   (b) otherwise, $\mathcal{B}$ evaluates the expression "$\mathsf{PRF}^{\mathrm{pk}}_{a_{\mathsf{sk},j}}(i \| h_{\mathsf{Sig}})$" by using $\mathcal{O}$; if the result equals

h_i, $\mathcal{B}$ outputs 1, else it outputs 0.

Conducting the above strategy does not require knowledge of $a_{\mathsf{sk},j}$ because, having the simulation trapdoor, $\mathcal{B}$ does not need witnesses to generate (valid) zk-SNARK proofs.

We now argue that $\left|\Pr\left[\,\mathcal{B}^{\mathsf{PRF}}(1^\lambda)=1\,\right]-\Pr\left[\,\mathcal{B}^{\mathsf{RAND}}(1^\lambda)=1\,\right]\right|$ is non-negligible.

- *Case 1:* $\mathcal{O}=\mathsf{RAND}$. Observe that:

$$\Pr\left[\,\mathcal{B}^{\mathsf{RAND}}(1^\lambda)=1 \mid \mathcal{B}^{\mathsf{RAND}}(1^\lambda) \text{ does not abort}\,\right]=2^{-\omega}\ .$$

  where ω is the output length of PRF. Hence:

$$\Pr\left[\,\mathcal{B}^{\mathsf{RAND}}(1^\lambda)=1\,\right]=\left(1-\Pr\left[\,\mathcal{B}^{\mathsf{RAND}}(1^\lambda) \text{ aborts}\,\right]\right)\cdot 2^{-\omega}+\Pr\left[\,\mathcal{B}^{\mathsf{RAND}}(1^\lambda) \text{ aborts}\,\right].$$

- *Case 2:* $\mathcal{O}=\mathsf{PRF}$. In this case the distribution of the simulation is identical to that of $\Im_1$, and $\mathcal{B}$ has set $a_{\mathsf{sk},j}$ equal to the seed used by $\mathcal{O}$. Recall that, when $\textsc{Event}_{\mathsf{mac}}$ holds, $h_i=\mathsf{PRF}^{\mathrm{pk}}_a(i\|h_{\mathsf{Sig}})$ for some $a\in\mathcal{Q}_{\mathbf{CA}}$. Since $\mathcal{A}$'s view of the experiment is independent of j, the probability that $a=a_{\mathsf{sk},j}$ is at least $1/q_{\mathbf{CA}}$, and the probability that $h_i=\mathsf{PRF}^{\mathrm{pk}}_{a_{\mathsf{sk},j}}(i\|h_{\mathsf{Sig}})$ is at least $\epsilon_3/q_{\mathbf{CA}}$. Hence:

$$\Pr\left[\,\mathcal{B}^{\mathsf{PRF}}(1^\lambda)=1 \mid \mathcal{B}^{\mathsf{PRF}}(1^\lambda) \text{ does not abort}\,\right]=\epsilon_3/q_{\mathbf{CA}}\ .$$

  Thus:

$$\Pr\left[\,\mathcal{B}^{\mathsf{PRF}}(1^\lambda)=1\,\right]=\left(1-\Pr\left[\,\mathcal{B}^{\mathsf{PRF}}(1^\lambda) \text{ aborts}\,\right]\right)\cdot \epsilon_3/q_{\mathbf{CA}}+\Pr\left[\,\mathcal{B}^{\mathsf{PRF}}(1^\lambda) \text{ aborts}\,\right].$$

Clearly, $2^{-\omega}$ is negligible; moreover, if ϵ_3 is non-negligible, then so is $|\epsilon_3/q_{\mathbf{CA}}|$. Thus, to show that $\left|\Pr\left[\,\mathcal{B}^{\mathsf{PRF}}(1^\lambda)=1\,\right]-\Pr\left[\,\mathcal{B}^{\mathsf{RAND}}(1^\lambda)=1\,\right]\right|$ is non-negligible, it suffices to show that each of $\Pr\left[\,\mathcal{B}^{\mathsf{RAND}}(1^\lambda) \text{ aborts}\,\right]$ and $\Pr\left[\,\mathcal{B}^{\mathsf{PRF}}(1^\lambda) \text{ aborts}\,\right]$ is negligible.

To do so, recall that $\mathcal{B}$ aborts if and only if it has previously evaluated the expression

"$\mathsf{PRF}^{\mathrm{pk}}_{a_{\mathsf{sk},j}}(i\|h_{\mathsf{Sig}})$" using $\mathcal{O}$ prior to receiving $\mathcal{A}$'s output. First note that $\mathcal{B}$'s only calls to $\mathcal{O}$ occur when it evaluates the functions $\mathsf{PRF}^{\mathrm{addr}}, \mathsf{PRF}^{\mathrm{sn}}$ and $\mathsf{PRF}^{\mathrm{pk}}$. Moreover, due to the construction of these functions it is not possible to evaluate the expression $\mathsf{PRF}^{\mathrm{pk}}_{a_{\mathsf{sk},j}}(i\|h_{\mathsf{Sig}})$ using any calls to $\mathsf{PRF}^{\mathrm{addr}}$ or $\mathsf{PRF}^{\mathrm{sn}}$. Thus $\mathcal{B}$ aborts if and only if it has previously queried $\mathsf{PRF}^{\mathrm{pk}}$ on the expression $\mathsf{PRF}^{\mathrm{pk}}_{a_{\mathsf{sk},j}}(i\|h_{\mathsf{Sig}})$. However it is easy to see that this cannot happen under the conditions of $\textsc{Event}_{\mathsf{mac}}$, since such a query would imply the condition $\textsc{Event}_{\mathsf{sig}}$ or $\textsc{Event}_{\mathsf{col}}$, each of which is excluded by $\textsc{Event}_{\mathsf{mac}}$. Hence the probability of either condition occurring is 0.

Bounding the probability of $\mathbf{Event}_{\mathsf{key}}$. Define $\epsilon_4 := \Pr[\,\textsc{Event}_{\mathsf{key}}\,]$, and let $\mathcal{E}$ be the zk-SNARK extractor for $\mathcal{A}$. Assume by way of contradiction that ϵ_4 is non-negligible. We construct an algorithm $\mathcal{B}$ that finds collisions for $\mathsf{PRF}^{\mathrm{sn}}$ with non-negligible probability (contradicting the fact that $\mathsf{PRF}^{\mathrm{sn}}$ is collision resistant). The algorithm $\mathcal{B}$ works as follows.

1. Run $\mathcal{A}$ (simulating its interaction with the challenger $\mathcal{C}$) to obtain tx^*.
2. Run $\mathcal{E}(\mathsf{pk}_{\mathtt{POUR}}, \mathsf{vk}_{\mathtt{POUR}})$ to obtain a witness a for the zk-SNARK proof $\pi_{\mathtt{POUR}}$ in tx^*.
3. If a is not a valid witness for the instance $x := (\mathsf{rt}, \mathsf{sn}^{\mathsf{old}}_1, \mathsf{sn}^{\mathsf{old}}_2, \mathsf{cm}^{\mathsf{new}}_1, \mathsf{cm}^{\mathsf{new}}_2, v_{\mathsf{pub}}, h_{\mathsf{Sig}}, h_1, h_2)$, abort and output 0.
4. Parse a as $(\mathsf{path}_1, \mathsf{path}_2, \mathbf{c}^{\mathsf{old}}_1, \mathbf{c}^{\mathsf{old}}_2, \mathsf{addr}^{\mathsf{old}}_{\mathsf{sk},1}, \mathsf{addr}^{\mathsf{old}}_{\mathsf{sk},2}, \mathbf{c}^{\mathsf{new}}_1, \mathbf{c}^{\mathsf{new}}_2)$.
5. For each $i \in \{1,2\}$, parse $\mathbf{c}^{\mathsf{old}}_i$ as $(\mathsf{addr}^{\mathsf{old}}_{\mathsf{pk},i}, v^{\mathsf{old}}_i, \rho^{\mathsf{old}}_i, r^{\mathsf{old}}_i, s^{\mathsf{old}}_i, \mathsf{cm}^{\mathsf{old}}_i)$.
6. For each $i \in \{1,2\}$, parse $\mathsf{addr}^{\mathsf{old}}_{\mathsf{sk},i}$ as $(a^{\mathsf{old}}_{\mathsf{sk},i}, \mathsf{sk}^{\mathsf{old}}_{\mathsf{enc},i})$.

   (Note that, since a is a valid witness, $\mathsf{sn}^{\mathsf{old}}_i = \mathsf{PRF}^{\mathrm{sn}}_{a^{\mathsf{old}}_{\mathsf{sk},i}}(\rho^{\mathsf{old}}_i)$ for all $i \in \{1,2\}$.)
7. For each $i \in \{1,2\}$:
   (a) Look for a pour transaction $\mathsf{tx} \in \mathcal{T}$ that contains $\mathsf{sn}^{\mathsf{old}}_i$.
   (b) If one tx is found, let $\overline{a_{\mathsf{sk}}}$ and $\overline{\rho}$ be the seed and input used to compute $\mathsf{sn}^{\mathsf{old}}_i$ in tx; thus, $\mathsf{sn}^{\mathsf{old}}_i = \mathsf{PRF}^{\mathrm{sn}}_{\overline{a_{\mathsf{sk}}}}(\overline{\rho})$. If $a^{\mathsf{old}}_{\mathsf{sk},i} \neq \overline{a_{\mathsf{sk}}}$, output $\big((a^{\mathsf{old}}_{\mathsf{sk},i}, \rho^{\mathsf{old}}_i), (\overline{a_{\mathsf{sk}}}, \overline{\rho})\big)$ as a collision for $\mathsf{PRF}^{\mathrm{sn}}$.

Note that, whenever $\textsc{Event}_{\mathsf{key}}$ holds:

- the proof $\pi_{\mathtt{POUR}}$ is valid and, with all but negligible probability, the witness a is valid;
- the serial number $\mathsf{sn}^{\mathsf{old}}_1$ or $\mathsf{sn}^{\mathsf{old}}_2$ appears in some previous pour transaction in $\mathcal{T}$;

- whenever a is valid, it holds that $h_1 = \mathsf{PRF}^{\mathrm{pk}}_{a^{\mathsf{old}}_{\mathsf{sk},1}}(h_{\mathsf{Sig}})$ and $h_2 = \mathsf{PRF}^{\mathrm{pk}}_{a^{\mathsf{old}}_{\mathsf{sk},2}}(h_{\mathsf{Sig}})$, so that it cannot be that $a^{\mathsf{old}}_{\mathsf{sk},1} = a^{\mathsf{old}}_{\mathsf{sk},2} = \overline{a_{\mathsf{sk}}}$ (as this contradicts the conditions of the event $\textsc{Event}_{\mathsf{key}}$).

Overall, we conclude that $\mathcal{B}$ finds a collision for $\mathsf{PRF}^{\mathrm{sn}}$ with probability $\epsilon_4 - \mathrm{negl}(\lambda)$.

B.3.3 Proof of balance

Define $\epsilon := \mathsf{Adv}^{\mathsf{BAL}}_{\Pi,\mathcal{A}}(\lambda)$; our goal is to show that ϵ is negligible in λ. Recall that ADDR is the set of addresses returned by $\mathcal{A}$'s **CreateAddress** queries.

Augmenting the ledger with witnesses. We modify the BAL experiment in a way that does not affect $\mathcal{A}$'s view: the challenger $\mathcal{C}$ computes, for each pour transaction $\mathsf{tx}_{\mathsf{Pour}}$ on the ledger L (maintained by the oracle $\mathcal{O}^{\mathsf{DAP}}$), a witness $a = (\mathsf{path}_1, \mathsf{path}_2, \mathbf{c}^{\mathsf{old}}_1, \mathbf{c}^{\mathsf{old}}_2, \mathsf{addr}^{\mathsf{old}}_{\mathsf{sk},1}, \mathsf{addr}^{\mathsf{old}}_{\mathsf{sk},2}, \mathbf{c}^{\mathsf{new}}_1, \mathbf{c}^{\mathsf{new}}_2)$ for the zk-SNARK instance $x = (\mathsf{rt}, \mathsf{sn}^{\mathsf{old}}_1, \mathsf{sn}^{\mathsf{old}}_2, \mathsf{cm}^{\mathsf{new}}_1, \mathsf{cm}^{\mathsf{new}}_2, v_{\mathsf{pub}}, h_{\mathsf{Sig}}, h_1, h_2)$ corresponding to $\mathsf{tx}_{\mathsf{Pour}}$.[3] In this way, $\mathcal{C}$ obtains an *augmented ledger* $(L, \vec{a})$, where a_i is a witness for the zk-SNARK instance x_i of the i-th pour transaction in L. Note that we can parse $(L, \vec{a})$ as a list of matched pairs $(\mathsf{tx}_{\mathsf{Pour}}, a)$ where $\mathsf{tx}_{\mathsf{Pour}}$ is a pour transaction in L and a is its corresponding witness.

The discussion below is relative to the above modification of the BAL experiment.

Balanced ledgers. We say that an augmented ledger $(L, \vec{a})$ is *balanced* if the following holds.

I. Each $(\mathsf{tx}_{\mathsf{Pour}}, a)$ in $(L, \vec{a})$ contains openings (i.e., decommitments) of two distinct coin commitments $\mathsf{cm}^{\mathsf{old}}_1$ and $\mathsf{cm}^{\mathsf{old}}_2$; also, each $\mathsf{cm}^{\mathsf{old}}_i$ is the output coin commitment of a pour or mint transaction that precedes $\mathsf{tx}_{\mathsf{Pour}}$ on L.

II. No two $(\mathsf{tx}_{\mathsf{Pour}}, a)$ and $(a', \mathsf{tx}'_{\mathsf{Pour}})$ in $(L, \vec{a})$ contain openings of the same coin commitment.

III. Each $(\mathsf{tx}_{\mathsf{Pour}}, a)$ in $(L, \vec{a})$ contains openings of $\mathsf{cm}^{\mathsf{old}}_1, \mathsf{cm}^{\mathsf{old}}_2, \mathsf{cm}^{\mathsf{new}}_1, \mathsf{cm}^{\mathsf{new}}_2$ to values $v^{\mathsf{old}}_1, v^{\mathsf{old}}_2$, $v^{\mathsf{new}}_1, v^{\mathsf{new}}_2$ (respectively), with the condition that $v^{\mathsf{old}}_1 + v^{\mathsf{old}}_2 = v^{\mathsf{new}}_1 + v^{\mathsf{new}}_2 + v_{\mathsf{pub}}$.

IV. For each $(\mathsf{tx}_{\mathsf{Pour}}, a)$ in $(L, \vec{a})$ and for each $i \in \{1, 2\}$, the following conditions hold:

[3] Concretely, for pour transactions in L not inserted by $\mathcal{A}$, $\mathcal{C}$ simply retains the witness a internally used by $\mathcal{O}^{\mathsf{DAP}}$ to generate the transaction. As for the (valid) pour transactions inserted by $\mathcal{A}$, $\mathcal{C}$ uses the zk-SNARK multi-instance knowledge extractor corresponding to $\mathcal{A}$; see Section 3.2.1. (If knowledge extraction fails, $\mathcal{C}$ aborts and outputs 1. However, this only happens with negligible probability.)

(a) If $\mathsf{cm}_i^{\mathsf{old}}$ is also the output of a mint transaction $\mathsf{tx}_{\mathsf{Mint}}$ on L, then the public value v in $\mathsf{tx}_{\mathsf{Mint}}$ is equal to v_i^{old}.

(b) If $\mathsf{cm}_i^{\mathsf{old}}$ is also the output of a pour transaction $\mathsf{tx}'_{\mathsf{Pour}}$ on L, then its witness a' contains an opening of $\mathsf{cm}_i^{\mathsf{old}}$ to a value v' that is equal to v_i^{old}.

V. For each $(\mathsf{tx}_{\mathsf{Pour}}, a)$ in $(L, \vec{a})$, where $\mathsf{tx}_{\mathsf{Pour}}$ was inserted by $\mathcal{A}$, it holds that, for each $i \in \{1, 2\}$, if $\mathsf{cm}_i^{\mathsf{old}}$ is the output of an earlier mint or pour transaction tx', then the public address of the i-th output of tx' is not contained in ADDR.

Intuitively, the above conditions ensure that, in L, $\mathcal{A}$ did not spend money that was not previously minted, or paid to an address under $\mathcal{A}$'s control. Concretely, one can prove by induction that if $(L, \vec{a})$ is balanced then $v_{\mathsf{Unspent}} + v_{\mathsf{Basecoin}} + v_{\mathcal{A}\to\mathsf{ADDR}} > v_{\mathsf{Mint}} + v_{\mathsf{ADDR}\to\mathcal{A}}$.

In light of the above, it suffices to argue that the augmented ledger induced by the (modified) BAL experiment is balanced with all but negligible probability. Suppose, by way of contradiction, that is is not the case: $\mathcal{A}$ induces, with non-negligible probability, an augmented ledger $(L, \vec{a})$ that is *not* balanced. We distinguish between five cases, corresponding to which one of the above conditions does not hold with non-negligible probability. In each case, we show how to reach a contradiction, concluding the proof.

$\mathcal{A}$ violates Condition I. Suppose that $\Pr[\,\mathcal{A}$ wins but violates Condition I$\,]$ is non-negligible. By construction of $\mathcal{O}^{\mathsf{DAP}}$, every $(\mathsf{tx}_{\mathsf{Pour}}, a)$ in $(L, \vec{a})$ for which $\mathsf{tx}_{\mathsf{Pour}}$ was not inserted by $\mathcal{A}$ satisfies Condition I; thus, the violation can only originate from a pair $(\mathsf{tx}_{\mathsf{Pour}}, a)$ in $(L, \vec{a})$ for which $\mathsf{tx}_{\mathsf{Pour}}$ was inserted by $\mathcal{A}$ and such that: (i) $\mathsf{cm}_1^{\mathsf{old}} = \mathsf{cm}_2^{\mathsf{old}}$; or (ii) there is $i \in \{1, 2\}$ such that $\mathsf{cm}_i^{\mathsf{old}}$ has no corresponding output coin commitment in any pour or mint transaction that precedes $\mathsf{tx}_{\mathsf{Pour}}$ on L.

Observe that the validity of $\mathsf{tx}_{\mathsf{Pour}}$ implies that:

- The two serial numbers $\mathsf{sn}_1^{\mathsf{old}}$ and $\mathsf{sn}_2^{\mathsf{old}}$ are distinct. Moreover, recalling that each $\mathsf{sn}_i^{\mathsf{old}}$ equals $\mathsf{PRF}^{\mathsf{sn}}_{a_{\mathsf{sk},i}^{\mathsf{old}}}(\rho_i^{\mathsf{old}})$, this also implies that $(a_{\mathsf{sk},1}^{\mathsf{old}}, \rho_1^{\mathsf{old}}) \neq (a_{\mathsf{sk},2}^{\mathsf{old}}, \rho_2^{\mathsf{old}})$.
- The witness a contains two valid authentication paths $\mathsf{path}_1, \mathsf{path}_2$ for a Merkle tree constructed

using only coin commitments of transactions preceding $\mathsf{tx}_{\mathsf{Pour}}$ in L.

In either (i) or (ii), we reach a contradiction. Indeed:

(i) If $\mathsf{cm}_1^{\mathsf{old}} = \mathsf{cm}_2^{\mathsf{old}}$, then the fact that $\mathsf{sn}_1^{\mathsf{old}} \neq \mathsf{sn}_2^{\mathsf{old}}$ implies that the witness a contains two distinct openings of $\mathsf{cm}_1^{\mathsf{old}}$ (the first opening contains $(a_{\mathsf{sk},1}^{\mathsf{old}}, \rho_1^{\mathsf{old}})$, while the second opening contains $(a_{\mathsf{sk},2}^{\mathsf{old}}, \rho_2^{\mathsf{old}})$). This violates the binding property of the commitment scheme COMM.

(ii) If there is $i \in \{1, 2\}$ such that $\mathsf{cm}_i^{\mathsf{old}}$ does not previously appear in L, then path_i is an invalid authentication path, and thus yields a collision in the function CRH. This violates the collision resistance of CRH.

$\mathcal{A}$ violates Condition II. Suppose that $\Pr[\,\mathcal{A}$ wins but violates Condition II$\,]$ is non-negligible. Observe that, when Condition II is violated, L contains two pour transactions $\mathsf{tx}_{\mathsf{Pour}}, \mathsf{tx}'_{\mathsf{Pour}}$ spending the same coin commitment cm, and revealing two serial numbers sn and sn'. Since $\mathsf{tx}_{\mathsf{Pour}}, \mathsf{tx}'_{\mathsf{Pour}}$ are valid, it must be the case that $\mathsf{sn} \neq \mathsf{sn}'$. However (as argued already above), if both transactions spend cm but produce different serial numbers, then the corresponding witnesses a, a' contain different openings of cm. This contradicts the binding property of the commitment scheme COMM.

$\mathcal{A}$ violates Condition III. Suppose that $\Pr[\,\mathcal{A}$ wins but violates Condition III$\,]$ is non-negligible. In this case, the contradiction is immediate: whenever Condition III is violated, the equation $v_1^{\mathsf{old}} + v_2^{\mathsf{old}} = v_1^{\mathsf{new}} + v_2^{\mathsf{new}} + v_{\mathsf{pub}}$ does not hold, and thus, by construction of the statement $\mathtt{POUR}$, the soundness of the zk-SNARK is violated as well.

$\mathcal{A}$ violates Condition IV. Suppose that $\Pr[\,\mathcal{A}$ wins but violates Condition IV$\,]$ is non-negligible. Observe that, when Condition IV is violated, L contains:

- a pour transaction $\mathsf{tx}_{\mathsf{Pour}}$ in which a coin commitment $\mathsf{cm}^{\mathsf{old}}$ is opened to a value v^{old}; and also
- a (mint or pour) transaction tx' that opens $\mathsf{cm}^{\mathsf{old}}$ to a value v' different from v^{old}.

This contradicts the binding property of the commitment scheme COMM.

$\mathcal{A}$ violates Condition V. Suppose that $\Pr[\,\mathcal{A}$ wins but violates Condition V$\,]$ is non-negligible. Observe that, when Condition V is violated, L contains an inserted pour transaction $\mathsf{tx}_{\mathsf{Pour}}$ that spends the output of a previous transaction tx' whose public address $\mathsf{addr}_{\mathsf{pk}} = (a_{\mathsf{pk}}, \mathsf{pk}_{\mathsf{enc}})$ lies in

ADDR; moreover, the witness associated to tx' contains a_{sk} such that $a_{\mathsf{pk}} = \mathsf{PRF}^{\mathrm{addr}}_{a_{\mathsf{sk}}}(0)$. We omit the full argument, but one can verify that, in this case, we can construct a new adversary $\mathcal{B}$ that uses $\mathcal{A}$ to distinguish, with non-negligible probability, PRF from a random function.

Appendix C

Bolt

C.1 Choice of cryptographic primitives

We now describe in depth our choice of cryptographic primitives:

C.1.1 Possible building blocks

Signatures with efficient protocols are the core building block of anonymous credentials and are a well studied primitive with many solutions offering various performance, security, and feature trade offs. One of the most efficient schemes that offers a full set of features and provable security is the bilinear variant of CL-signatures due to Camenisch and Lysyanskaya [97]. An implementation exists in Charm [39].

We are aware of two other candidate signature schemes with available implementations from petLib [114] that are aimed at providing increased performance with reduced functionality. The first is used in the construction of Lightweight Anonymous Credentials [115]. Here signatures can only be shown anonymously once. Second, Algebraic MACs are used in [106], to build a limited form of anonymous credential. Because it uses a MAC not a signature, only the issuer can verify "signed" messages. This requires some modification to our protocol since closure of a channel currently

requires public verification of the refund token.

C.1.2 Selecting the signature scheme

The scheme from Anonymous Credentials Light [115] is the fastest for issuing and showing, with most operations taking less than $0.01ms$. However, a registration phase must be completed for the set of messages that can be signed. This must be repeated every time the set changes and takes $100ms$. Because the refund token rt is selected at random and changes on every instance, this process must be done on every payment. Moreover, even if this were made far faster, the registration process reveals the message set. It may be possible to patch Bolt to accommodate this or modify the credential scheme to remove the restriction, but the $100ms$ cost is too high to pay per payment. The remaining two schemes are more promising with most operations taking less than $30ms$ each.

C.1.3 Implementation

We build two completely distinct implementations of the bidirectional payment protocol. One using bilinear CL-Sigs and the other using Algebraic MACs. Our approach mirrors the construction of a credential scheme: we present a commitment to the wallet and a proof that it is signed and then use Schnorr proofs [17] to prove the balance of the new wallet commitment is correct with respect to the old wallet. We then blindly obtain a signature on the new wallet. For the range proof, we use a technique reminiscent of [100]: we decompose the balance into bytes and prove we have a signature issued on each byte. This allows us to reuse the code and primitives from the signature scheme rather than using a separate range proof which would introduce more cryptographic assumptions, more code, and more dependencies.

The results are given in Table 4.6. The implementation based on Algebraic MACs is approximately twice as fast as the CL-Sig approach. It should be noted, however, that there is far more room for optimization in the CL-signature library. While both are implemented in Python, the implementation of Algebraic MACs use only elliptic curve operations via openSSL. As such, the

principle overhead is from calling native code from Python. On the other hand, the CL-signature implementation uses symmetric bilinear pairings with an implementation from the PBC library [116]. Use of asymmetric pairings and a faster pairing library such as RELIC [117] would give a marked improvement.

C.1.4 Adapting channel closure to avoid public verification of credentials

Closing a disputed channel currently requires the blockchain to verify that the refund token is signed. For our faster construction, this is impossible since the key remains secret and the "signature" is actually a MAC. There are two solutions to this: 1) we can, as outlined in paragraph 4.5.1 opt to have the blockchain validate conventional signatures at the cost of an extra round trip in pay. 2) We can allow the merchant to prove that a purported MAC is invalid.

The MAC itself consists of $u, u' = u^{\mathcal{H}_x(m)}$ where $\mathcal{H}_x(m)$ is a keyed and deterministic hash function. Unfortunately, u is chosen at random so the MAC is not unique and it is not sufficient to reveal the correct MAC on the message and prove its correctness.[1] Instead, we must prove that $log_u(u') \neq log_v(v')$ (i.e. that $u_x^{\mathcal{H}}(m) \neq u'$) and that the revealed MAC v, v') is correct. Camenisch and Shoup give an extremely efficient proof for discrete log inequality [118] where only one discrete log is known to the prover and none known to the verifier. We implement this full proof of invalid MAC combining the prove of MAC validity and discrete log inequality. It takes approximately $14ms$ to generate and verify. We note that as this proof includes the actual valid MAC on the forged refund token, it is necessary for the blockchain to blacklist this MAC and not accept it. However, since the refund token can never be used in payments, we need not add extra steps to the pay protocol.

[1] Counter-intuitively, despite being built on a MAC, Keyed-Verification Anonymous Credentials include an efficient zk-proof of validity of a MAC that effectively transforms the MAC into a (non blind) signature. Since this proof is somewhat expensive, it is only used to verify the correctness of issued credentials.

C.2 Security Definitions

In this section we provide formal security definitions for an anonymous payment channel scheme.

C.2.1 Payment anonymity

Let $\mathcal{A}$ be an adversary playing the role of merchant. We consider an experiment involving P "customers", each interacting with the merchant as follows. First, $\mathcal{A}$ is given pp, then outputs $\mathsf{T}_{\mathcal{M}}$. Next $\mathcal{A}$ issues the following queries in any order:

Initialize channel for $\mathcal{C}_i$. When $\mathcal{A}$ makes this query on input $B^{\mathsf{cust}}, B^{\mathsf{merch}}$, it obtains the commitment $\mathsf{T}^i_{\mathcal{C}}$, generated as $(\mathsf{T}^i_{\mathcal{C}}, csk^i_{\mathcal{C}}) \xleftarrow{R} \mathsf{Init}_{\mathcal{C}}(\mathsf{pp}, B^{\mathsf{cust}}, B^{\mathsf{merch}})$.

Establish channel with $\mathcal{C}_i$. In this query, $\mathcal{A}$ executes the $\mathsf{Establish}$ protocol with $\mathcal{C}_i$ as:

$$\mathsf{Establish}(\{\mathcal{C}(\mathsf{pp}, \mathsf{T}_{\mathcal{M}}, csk^i_{\mathcal{C}})\}, \{\mathcal{A}(state)\}$$

Where *state* is the adversary's state. Let us denote the customer's output as w_i, where w_i may be $\perp$.

Payment from $\mathcal{C}_i$. In this query, if $w_i \neq \perp$, then $\mathcal{A}$ executes the Pay protocol for an amount ϵ with $\mathcal{C}_i$ as:

$$\mathsf{Pay}(\{\mathcal{C}(\mathsf{pp}, \epsilon, w_i)\}, \{\mathcal{A}(state)\})$$

Where *state* is the adversary's state. We denote the customer's output as w_i, where w_i may be $\perp$.

Finalize with $\mathcal{C}_i$. When $\mathcal{A}$ makes this query, it obtains the closure message $\mathsf{rc}^i_{\mathcal{C}}$, computed as $\mathsf{rc}_{\mathcal{C}} \xleftarrow{R} \mathsf{Refund}(\mathsf{pp}, w_i)$.

We say that $\mathcal{A}$ is *legal* if $\mathcal{A}$ never asks to spend from a wallet where $w_i = \perp$ or where w_i is undefined, and where $\mathcal{A}$ never asks $\mathcal{C}_i$ to spend unless the customer has sufficient balance to complete the spend.

Let *auxparams* be an auxiliary trapdoor not available to the participants of the real protocol. We require the existence of a simulator $\mathcal{S}^{X-Y(\cdot)}(\mathsf{pp}, auxparams, \cdot)$ such that for all $\mathsf{T}_{\mathcal{M}}$, no allowed adversary $\mathcal{A}$ can distinguish the following two experiments with non-negligible advantage:

Real experiment. In this experiment, all responses are computed as described above.

Ideal experiment. In this experiment, the Commitment, Establishment and Finalize queries are handled using the procedure described abvove. However, in the Payment query, $\mathcal{A}$ does not interact with $\mathcal{C}_i$ but instead interacts with $\mathcal{S}^{X-Y(\cdot)}(\mathsf{pp}, auxparams, \cdot)$.

As in [90] we note that this definition preserves anonymity because the simulator $\mathcal{S}$ does not know the identity of the user i for which he is spending the coin.

C.2.2 Payment Balance

$\mathcal{A}$ interacts with a collection of P honest customers $\mathcal{C}_1, \ldots, \mathcal{C}_P$ and Q honest merchants $\mathcal{M}_1, \ldots, \mathcal{M}_Q$. Initialize the counters $\mathsf{bal}_{\mathcal{A}} \leftarrow 0, \mathsf{claimed}_{\mathcal{A}} \leftarrow 0$. Let $\mathsf{pp} \leftarrow \mathsf{Setup}(1^{\lambda})$. For each merchant $i \in [1, Q]$, at the start of the game let $(pk_{\mathcal{M}_i}, sk_{\mathcal{M}_i}) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$. Give pp and $(pk_{\mathcal{M}_1}, \ldots, pk_{\mathcal{M}_Q})$ to $\mathcal{A}$. Now $\mathcal{A}$ may issue the following queries in any order:

Initialize channel for $\mathcal{C}_i$ (resp. $\mathcal{M}_i$) When $\mathcal{A}$ makes this query on input $(\mathcal{P}_i, B_0^{\mathsf{cust}}, B_0^{\mathsf{merch}})$, it obtains the commitment $\mathsf{T}_{\mathcal{C}_i}$ (resp. $\mathsf{T}_{\mathcal{M}_i}$) computed as follows:

- If the party $\mathcal{P}_i$ is a customer: First compute $(pk_{\mathcal{C}_i}, sk_{\mathcal{C}_i}) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$, then $(\mathsf{T}_{\mathcal{C}_i}, csk_{\mathcal{C}}^i) \xleftarrow{R} \mathsf{Init}_{\mathcal{C}}(\mathsf{pp}, B_0^{\mathsf{cust}}, B_0^{\mathsf{merch}}, pk_{\mathcal{C}_i}, sk_{\mathcal{C}_i})$. Set $\mathsf{bal}_{\mathcal{A}} \leftarrow \mathsf{bal}_{\mathcal{A}} + B_0^{\mathsf{merch}}$.
- If the party $\mathcal{P}_i$ is a merchant: Compute $(\mathsf{T}_{\mathcal{M}_i}, csk_{\mathcal{M}_i}) \xleftarrow{R} \mathsf{Init}_{\mathcal{M}}(\mathsf{pp}, B_0^{\mathsf{cust}}, B_0^{\mathsf{merch}}, pk_{\mathcal{M}_i}, csk_{\mathcal{M}}^i)$. Set $\mathsf{bal}_{\mathcal{A}} \leftarrow \mathsf{bal}_{\mathcal{A}} + B_0^{\mathsf{cust}}$.

Establish channel with $\mathcal{C}_i$ (resp. $\mathcal{M}_i$). When $\mathcal{A}$ specifies $(\mathcal{P}_i, \mathsf{T}_{\mathcal{A}})$, and $\mathcal{A}$ has previously initialized a channel with party $\mathcal{P}_i$, execute the $\mathsf{Establish}$ protocol with $\mathcal{C}_i$ (resp. $\mathcal{M}_i$) using the following input:

- If $\mathcal{P}_i$ is a customer: $\mathsf{Establish}(\{\mathcal{C}_i(\mathsf{pp}, \mathsf{T}_{\mathcal{A}}, csk^i_{\mathcal{C}})\}, \{\mathcal{A}(state)\} \rightarrow w_i$ (or $\perp$).
- If $\mathcal{P}_i$ is a merchant: $\mathsf{Establish}(\{\mathcal{A}(state)\}, \{\mathcal{M}(\mathsf{pp}, \mathsf{T}_{\mathcal{A}}, csk^i_{\mathcal{M}})\} \rightarrow \mathsf{established}$ (or $\perp$).

Where *state* is the adversary's state.

Payment from $\mathcal{C}_i$ (resp. to $\mathcal{M}_i$). In this query, $\mathcal{A}$ specifies $(\mathcal{P}_i, \epsilon)$ where ϵ may be positive or negative. If $\mathcal{A}$ has previously conducted the $\mathsf{Establish}$ protocol with this party and the party's output was not $\perp$, then execute the Pay protocol with $\mathcal{A}$ as:

- If $\mathcal{P}_i$ is a customer: $\mathsf{Pay}(\{\mathcal{C}_i(\mathsf{pp}, \epsilon, w_i)\}, \{\mathcal{A}(state)\}) \rightarrow w_i$ (or $\perp$). If the customer's output is not $\perp$, set $\mathsf{bal}_{\mathcal{A}} \leftarrow \mathsf{bal}_{\mathcal{A}} + \epsilon$.
- If $\mathcal{P}_i$ is a merchant: $\mathsf{Pay}(\{\mathcal{A}(state)\}, \{\mathcal{M}_i(\mathsf{pp}, \epsilon, \mathbf{S}_i)\}) \rightarrow \mathbf{S}_i$ (or $\perp$). If the merchant's output is not $\perp$, $\mathsf{bal}_{\mathcal{A}} \leftarrow \mathsf{bal}_{\mathcal{A}} - \epsilon$.

Where *state* is the adversary's state.

Finalize with $\mathcal{C}_i$ (resp. $\mathcal{M}_i$) When $\mathcal{A}$ makes this query on input $\mathcal{P}_i$ and optional input $\mathsf{rc}_{\mathcal{C}}$, if it has previously established a channel with $\mathcal{P}_i$, it obtains a closure message as:

- If $\mathcal{P}_i$ is a customer: if $\mathcal{A}$ has previously established a channel with $\mathcal{P}_i$ and has not previously Finalized on this party, compute $\mathsf{rc}_{\mathcal{C}} \xleftarrow{R} \mathsf{Refund}(\mathsf{pp}, w_i)$, store $\mathsf{rc}_{\mathcal{C}}$, and return $\mathsf{rc}_{\mathcal{C}}$ to $\mathcal{A}$.
- If $\mathcal{P}_i$ is a merchant: if $\mathcal{A}$ has previously established a channel with $\mathcal{P}_i$ and has not previously Finalized on this party, compute $\mathsf{rc}_{\mathcal{M}} \xleftarrow{R} \mathsf{Refute}(\mathsf{pp}, \mathbf{S}_i, \mathsf{rc}_{\mathcal{C}})$.

If the adversary provided $\mathsf{rc}_{\mathcal{M}}$ and $\mathsf{rc}_{\mathcal{C}}$ is stored, compute $(B^{\mathsf{cust}}_{\mathsf{final}}, B^{\mathsf{merch}}_{\mathsf{final}}) \leftarrow \mathsf{Resolve}(\mathsf{pp}, \mathsf{T}_{\mathcal{C}}, \mathsf{T}_{\mathcal{M}}, \mathsf{rc}_{\mathcal{C}}, \mathsf{rc}_{\mathcal{M}})$ and update $\mathsf{claimed}_{\mathcal{A}} \leftarrow \mathsf{claimed}_{\mathcal{A}} + B^{\mathsf{merch}\ (\text{resp. cust})}_{\mathsf{final}}$.

We say that $\mathcal{A}$ is *legal* if $\mathcal{A}$ never asks to spend from a wallet where $w_i = \perp$ or where w_i is undefined, and where $\mathcal{A}$ never asks $\mathcal{C}_i$ to spend unless the customer has sufficient balance to complete the spend. We say that $\mathcal{A}$ wins the game if at the conclusion of $\mathcal{A}$'s queries, we have $\mathsf{claimed}_{\mathcal{A}} > \mathsf{bal}_{\mathcal{A}}$.

C.3 Proof of Security for Unidirectional Scheme

Proof of Theorem 4.4.1

Proof. The proof of Theorem 4.4.1 requires two separate arguments: (1) that the scheme satisfies the *anonymity* property and (2) that the scheme satisfies the *balance* property. We begin by addressing anonymity.

C.3.1 Anonymity

To prove that the scheme satisfies the anonymity property, we must describe a simulator $\mathcal{S}^{X-Y(\cdot)}(\mathsf{pp}, auxparams, \cdot)$ such that for all $\mathsf{T}_{\mathcal{M}}$, no allowed adversary $\mathcal{A}$ can distinguish the Real experiment from the Ideal experiment with non-negligible advantage. Recall that in the Ideal experiment (as in the Real experiment), when the adversary $\mathcal{A}$ queries on channel initialization, establishment or closure, the customer answers these queries by honestly running the appropriate algorithms. When the adversary triggers a customer to initiate the Pay protocol, in the Real experiment the adversary runs the protocol honestly. In the Ideal experiment, the customer's side of the protocol is conducted by $\mathcal{S}$.

For all allowed adversaries $\mathcal{A}$, the simulator $\mathcal{S}$ operates as follows. First, if required by the zero-knowledge proof system, we generate a *simulation* CRS for the zero-knowledge proof system, and embed this in pp.[2] When $\mathcal{A}$ calls the simulator on a legal transaction, the simulator emulates the customer's side of the Pay protocol, but with the following changes. First, for $j = 1$ to B, the simulator $\mathcal{S}$ employs the ZK simulation algorithm to simulate each of the zero knowledge proofs π. It generates s_i by sampling a random element in the range of F. Finally, it samples a random key k' for the one-time encryption scheme, samples a random public key pk' by running the KeyGen algorithm, and sets $t := \mathsf{OTEnc}(k', pk')$.

To prove that the Real and Ideal experiments are indistinguishable, we will begin with Real

[2] This is necessary for certain proof systems such as [99].

experiment, and modify elements via a series of games until we arrive at the Ideal experiment conducted using our simulator $\mathcal{S}$. Let $\text{negl}_1, \ldots, \text{negl}_3$ be negligible functions. For notational convenience, let $\mathsf{Adv}\textbf{Game i}$ be $\mathcal{A}$'s advantage in distinguishing the output of **Game i** from **Game 0**, *i.e.,* the Real distribution.

Game 0. This is the real experiment.

Game 1. In this game, each NIZK π issued during the Pay protocol is simulated. If the proof system is zero-knowledge, then $\mathsf{Adv}\textbf{Game 1} \leq \text{negl}_1(\lambda)$.

Game 2. In this game, each serial number s presented to $\mathcal{A}$ in the Pay protocol, and each encryption key k used to construct the value t, is replaced with a random value in the range of the pseudorandom function F. In Lemma C.3.1 we show that if the F is a PRF, the commitment scheme is hiding, and the committing encryption is IND-CPA, then $\mathsf{Adv}\textbf{Game 2} - \mathsf{Adv}\textbf{Game 1} \leq \text{negl}_2(\lambda)$.

Game 3. In this game, each value t presented to $\mathcal{A}$ in the Pay protocol is constructed by sampling a random $(pk'_c, sk'_c) \leftarrow \mathsf{KeyGen}(1^\lambda)$, then encrypting pk'_c. Under the assumption that OTEnc is IND-CPA for a unique, random key k, then $\mathsf{Adv}\textbf{Game 3} - \mathsf{Adv}\textbf{Game 2} \leq \text{negl}_3(\lambda)$.

By summation over the individual hybrids, we have that $\mathsf{Adv}\textbf{Game 3}$ is negligible in the security parameter. Since the distribution of **Game 3** is identical to the Ideal experiment conducted with our simulator $\mathcal{S}$, this concludes the main proof. We now sketch proofs of the remaining Lemmas.

Lemma C.3.1 (Replacement of the s, k_t values.)**.** For all p.p.t. distinguishers $\mathcal{A}$ the distribution of **Game 1** (in which each value s, t is generated as in the real protocol) is computationally indistinguishable from the distribution of **Game 2** (in which each s and the key k_t used to encrypt t is a random element) if (1) F is a PRF, (2) the wallet commitment scheme is hiding, and (3) the committing symmetric encryption scheme $(\mathsf{SymEnc}, \mathsf{SymDec})$ is IND-CPA secure.

Proof sketch. Let $\mathcal{A}$ be an allowed adversary that outputs 1 with non-negligibly different probability when playing **Game 2** and **Game 1**. We use $\mathcal{A}$ to construct three separate distinguishers $\mathcal{B}_1, \mathcal{B}_2, \mathcal{B}_3$ where at least one of the following is true: (1) $\mathcal{B}_1$ distinguishes the PRF F from a random function with non-negligible advantage, (2) $\mathcal{B}_2$ succeeds against the IND-CPA security of the committing symmetric encryption scheme $(\mathsf{SymEnc}, \mathsf{SymDec})$ with non-negligible advantage, or (3) $\mathcal{B}_3$ succeeds against the hiding property of the commitment scheme with non-negligible advantage.

Let us define a series of intermediate hybrids $H_0 = \textbf{Game 1}, \ldots, H_P = \textbf{Game 2}$, and in each Hybrid $i = 1$ to P, the output of the Pay protocol for a single customer $\mathcal{C}_i$ is modified in the manner of **Game 2**. Given an allowed adversary $\mathcal{A}$ that distinguishes **Game 1** from **Game 2** with non-negligible probability, there must exist an adversary $\mathcal{A}'$ that for some $i \in \{1, \ldots, P\}$, distinguishes one pair of hybrids H_i and H_{i-1} with non-negligible probability. Given such an adversary we now define several more hybrids, and argue that for each of these hybrids the adversary $\mathcal{A}$ must distinguish each from the previous hybrid with at most negligible probabilty.

I.1 Replace the proof π_1 issued by $\mathcal{C}_i$ during the $\mathsf{Establish}$ protocol with a simulated proof. If the proof system is zero knowledge, then $\mathcal{A}$'s advantage in distinguishing this hybrid from the previous hybrid is negligible in λ.

I.2 Replace wCom with a commitment to random values k_1', k_2'. If an adversary distinguishes this hybrid from the previous with non-negligible advantage, then this implies a distinguisher $\mathcal{B}_3$ that succeeds with non-negligible advantage against the hiding property of the commitment scheme. Since we assume the commitment scheme is secure, this bounds the difference between the hybrids to be negligible in λ. (Note that the NIZK proof π_1 is simulated and thus independent of wCom and k_1', k_2'.)

I.3 Replace each s, k_t in the Pay protocol with a value computed using k_1', k_2'. If an adversary distinguishes this hybrid from the previous hybrid with non-negligible advantage,

then by Lemma C.3.2 this implies an attacker against $\mathcal{B}_2$ that wins the IND-CPA game with non-negligible advantage against (SymEnc, SymDec). Since we assume the encryption scheme to be IND-CPA secure, this bound the difference between the hybrids to be negligible in λ. (Recall that the NIZK proof generated in the Pay protocol is simulated.)

I.4 Replace each s, k_t in the Pay protocol with a random element in the range of F. If an adversary distinguishes this hybrid from the previous hybrid with non-negligible advantage, then this implies the existence of $\mathcal{B}_1$ that distinguishes F from a random function, hence under the assumption that F is a PRF, this bounds the difference between the hybrids to be negligible in λ.

I.5 Replace the commitment wCom and proof π_1 with the original distribution from Game 2. Under the assumption that the proof system is zero-knowledge, and the commitment scheme is hiding, the difference between this hybrid and the previous is negligible in λ.

Note that the final hybrid is identical to **Game 2**. Under the assumptions that the proof system is zero knowledge, that F is a PRF, the committing encryption scheme is IND-CPA secure, and the commitment scheme is hiding, the difference between $\mathcal{A}$'s probability of outputting 1 in **Game 2** and **Game 1** is negligible in λ. □

Lemma C.3.2 (Replacement of the wallet secrets.)**.** For all p.p.t. adversaries $\mathcal{A}$ no adversary can distinguish the intermediate hybrid **I.2** from hybrid **I.3** with non-negligible probability if (SymEnc, SymDec) is IND-CPA secure.

Proof sketch. Let $\mathcal{A}$ be an allowed adversary that outputs 1 with non-negligibly different probability in hybrid **I.2** from hybrid **I.3**. We show that $\mathcal{A}$ implies an adversary $\mathcal{B}_2$ such that $\mathcal{B}_2$ succeeds in the IND-CPA game against the encryption scheme (SymEnc, SymDec) with non-negligible advantage. We now describe this adversary.

If $\mathcal{A}'$ distinguishes **I.2** from hybrid **I.3** with non-negligible advantage, we construct $\mathcal{B}_2$ that

succeeds with non-negligible advantage against the LOR-CPA security of the symmetric encryption scheme (a definition that is equivalent to IND-CPA security [119]). $\mathcal{B}_2$ begins with the distribution of **I.2** and first picks a random integer $J \in \{0, \ldots, B\}$ and for $d = 1$ to J: queries the LOR encryption oracle on the pair $(s_d \| u_d \| \pi_d^r, s'_d \| u'_d \| \pi_d'^r)$ where the left input is structured as in **I.2** and the right input is structured as in **I.3**. Given the resulting ciphertexts $C_1, \ldots, C_J$, $\mathcal{A}_1$ now generates the remaining ciphertexts $C'_{J+1}, \ldots, C'_B$ by querying the LOR oracle such that both inputs are constructed as in hybrid H_j. It then constructs the ciphertext vector for customer i as $(C_1, \ldots, C_J, C'_{j+1}, \ldots, C'_B)$ and gives this to $\mathcal{A}'$ as $\mathcal{C}_i$'s output in the $\mathsf{Establish}$ protocol. Note that if the LOR oracle chooses the left input, the distribution of this vector is as in **I.2**, and if it chooses the right input, the distribution is as in **I.3**. When $\mathcal{A}'$ finalizes the channel with $\mathcal{C}_i$, check that the final balance of the customer is $B - J$, and if not, abort. Otherwise, finalize the channel and output $\mathcal{A}'$'s guess as the guess for the LOR-CPA oracle. Note that the abort probability is at most $1/P$, for P polynomial in λ. $\square$

C.3.2 Balance

We now sketch a proof that the scheme satisfies the Balance definition if the zero-knowledge proof system is simulation extractable, the commitment scheme is binding, and the signature schemes are EU-CMA secure. The primary observation in our proof is that if $\mathcal{A}$, acting as a customer, is able to succeed in obtaining $\mathsf{claimed}_{\mathcal{A}} > \mathsf{bal}_{\mathcal{A}}$, this implies that one of the following conditions is true: (1) $\mathcal{A}$ has successfully paid more than B_0^{cust} coins on a given channel, (2) during the Finalization process, $\mathcal{A}$ has successfully claimed more than the remaining number of coins on a given channel and the honest merchant is not able to produce evidence of fraud. Similarly, a legal adversary $\mathcal{A}$ that wins the game must succeed in either (3) producing evidence (as a merchant) of a doubly-spent coin, even when the customer has behaved honestly, or (4) producing evidence of an invalid ciphertext opening.

Let us first describe a simulated experiment, which is identical to the real protocol interaction but with the following differences. First, if necessary we configure the proof system to allow for the extraction of witnesses, and embed any resulting CRS into pp. Whenever $\mathcal{A}$ initiates the Pay protocol

(acting as a customer) to send a successful (accepted) payment, we then extract the witness used to construct the proof π and abort the experiment if the extractor does not produce a valid witness. In addition, we abort if $\mathcal{A}$ is able to submit more than B coins for any given channel (identified by the witness), or if the attacker is able to submit a signature forgery (*i.e.,* submit a signature that was not granted through the Establish protocol). Finally, if the attacker Finalizes the channel, extracting more than the remaining number of coins available on a given channel, we abort (this implies that $\mathcal{A}$ has produced an additional spend value with respect to the commitment wCom). We simulate all proofs issued during the Pay and Establish protocols. Whenever the adversary, acting as a merchant, posts a channel closure message $\mathsf{rc}_{\mathcal{M}}$ such that Resolve executed on the customer and merchant inputs outputs $B_{\mathsf{final}}^{\mathsf{merch}} > 0$, where the final balance is inconsistent with the actual remaining balance, we abort the protocol. We note that if there exists an adversary $\mathcal{A}$ who succeeds in winning the Balance game with non-negligible advantage, then this implies an attacker with the ability to distinguish the real experiment from the simulated experiment with non-negligible advantage. We show that such an attacker represents a contradiction, assuming that the proof system is sound and the signature scheme is EU-CMA. Consider the following hybrids:

Game 0. This is the real experiment.

Game 1. This game is identical to **Game 0** except that we extract on every valid proof π_1 in the Establish protocol, every proof π in the Pay protocol, and every proof π_r^j that the customer reveals as a result of a channle Finalization. We abort if the extractor ever fails to produce a valid witness. Under the assumption that the proof system is sound, the abort probability is negligible. Thus $\mathsf{Adv}\textbf{Game 1} \leq \mathrm{negl}_1(\lambda)$.

Game 2. This game is identical to **Game 1** except that we abort if the customer ever presents a collision in wCom (*e.g.,* in the witness to any proof of knowledge). Assuming that the commitment scheme is binding, $\mathsf{Adv}\textbf{Game 2} - \mathsf{Adv}\textbf{Game 1} \leq \mathrm{negl}_2(\lambda)$.

Game 3. This game is identical to **Game 2** except that we abort if $\mathcal{A}$ is able to successfully submit

$B' > B$ coins on a given channel. Note that the serial number s is computed as a function of the secret key k_1 and the coin index $0 \leq i < B$. Thus, there are at most B distinct values of s for any given (signed) PRF seed k_1. Thus, for this abort to occur, it must be the case that $\mathcal{A}$ has forged a signature σ_w that was not issued during the Establish protocol. If this occurs, we obtain an adversary $\mathcal{B}$ that succeeds against the EU-CMA security of the signature. Since we assume that the signature scheme is EU-CMA secure, then we obtain $\mathsf{Adv}\textbf{Game 3} - \mathsf{Adv}\textbf{Game 2} \leq \mathrm{negl}_3(\lambda)$.

Game 4. This game is identical to **Game 3**, except that we abort if $\mathcal{A}$ ever produces a ciphertext C_j that contains a witness to the proof statement with an opening of the commitment wCom that does not match with the corresponding values used in the Pay protocol. If this occurs, this implies an attacker that violates the binding property of the commitment scheme. Since we assume that the commitment scheme is binding, then we obtain $\mathsf{Adv}\textbf{Game 4} - \mathsf{Adv}\textbf{Game 3} \leq \mathrm{negl}_4(\lambda)$.

Game 5. This game is identical to **Game 4** except that whenever $\mathcal{A}$ presents signed evidence that the customer has supplied an invalid ciphertext (that does not decrypt with key ck_j), we abort. Since no customer ever outputs invalid ciphertexts or keys, this implies that the adversary has constructed a forged signature using the signature scheme. This implies that we can use $\mathcal{A}$ to win the EU-CMA game against the signature scheme. Thus, under the assumption that the signature scheme is EU-CMA secure, we have that $\mathsf{Adv}\textbf{Game 5} - \mathsf{Adv}\textbf{Game 4} \leq \mathrm{negl}_5(\lambda)$.

Game 6. This game is identical to **Game 5** except that we simulate each zero knowledge proof issued in the Pay and Establish protocols. Since the proof system is zero knowledge, we have that $\mathsf{Adv}\textbf{Game 6} - \mathsf{Adv}\textbf{Game 5} \leq \mathrm{negl}_6(\lambda)$.

Game 7. This game is identical to **Game 6** except that whenever $\mathcal{A}$, acting as a merchant, presents signed evidence of a doubly-spent coin that is accepted by the Resolve algorithm, we abort. We argue that intuitively, such an adversary $\mathcal{A}$ can be used to break the EU-CMA property

of the signature scheme or the IND-CPA property of the symmetric encryption scheme as follows. On input a public key in the EU-CMA game, embed this key as pk_c. Now guess an index J at which the payment channel will be closed. We further replace each of the first $J-1$ ciphertexts created during the Establish protocol with the encryption of a random element. Now, if the adversary outputs a new proof, note that we can extract a witness to the (new) proof, which is distinct from any of the previous proofs and therefore embeds a valid secret key sk_c for the customer. This provides us with the signing key for the signature scheme and allows us to forge a signature on any message. This proof requires that $\mathcal{A}$ cannot distinguish the encryption of random messages from the encryption of valid proofs; this can be shown using the IND-CPA property of the signature scheme. Completing this proof requires a hybrid argument in which the above process is repeated for each customer. Thus, under the assumption that the scheme is IND-CPA secure and the signature scheme is EU-CMA secure, we have $\mathsf{Adv}\textbf{Game 7} - \mathsf{Adv}\textbf{Game 6} \leq \mathrm{negl}_7(\lambda)$.

By summation over the individual hybrids, we have that $\mathsf{Adv}\textbf{Game 7}$ is negligible in the security parameter. We note that the distribution of **Game 7** is computationally indistinguishable from the real experiment. Thus the simulation satisfies the property of Balance. □

C.4 Proof of Security for Bidirectional Scheme

Proof sketch. As in the previous proofs, the proof of Theorem 4.4.2 requires two separate arguments: (1) that the scheme satisfies the *anonymity* property and (2) that the scheme satisfies the *balance* property. We begin by addressing anonymity. Note that for this scheme we make the simplifying assumption that the legal adversary does not abort the Pay protocol.

C.4.1 Anonymity

To prove that the scheme satisfies the anonymity property, we must describe a simulator $\mathcal{S}^{X-Y(\cdot)}(\mathsf{pp}, auxparams, \cdot)$ such that for all $\mathsf{T}_{\mathcal{M}}$, no allowed adversary $\mathcal{A}$ can distinguish the Real experiment from the Ideal experiment with non-negligible advantage. Recall that in the Ideal experiment (as in the Real experiment), when the adversary $\mathcal{A}$ queries on channel initialization, establishment or closure, the customer answers these queries by honestly running the appropriate algorithms. When the adversary triggers a customer to initiate the Pay protocol, in the Real experiment the adversary runs the protocol honestly. In the Ideal experiment, the customer's side of the protocol is conducted by $\mathcal{S}$.

For all allowed adversaries $\mathcal{A}$, the simulator $\mathcal{S}$ operates as follows. First, if required by the zero-knowledge proof system, we generate a *simulation* CRS for the zero-knowledge proof system, and embed this in pp.[3] When $\mathcal{A}$ calls the simulator on a legal transaction, the simulator $\mathcal{S}$ emulates the customer's side of the Pay protocol, but with three differences: (1) the commitment wCom' is replaced with a commitment to a random message, (2) the simulator $\mathcal{S}$ generates a random public key *wpk* when it runs the protocol, and (3), the simulator employs the ZK simulation algorithm to simulate each of the zero-knowledge proofs. In all other ways it behaves as in the normal protocol, generating *wpk* and σ_{rev} as usual.

To prove that the Real and Ideal experiments are indistinguishable, we will begin with Real experiment, and modify elements via a series of games until we arrive at the Ideal experiment conducted using our simulator $\mathcal{S}$. Let $\mathrm{negl}_1, \mathrm{negl}_2$ be negligible functions. For notational convenience, let Adv**Game i** be $\mathcal{A}$'s advantage in distinguishing the output of **Game i** from the Real distribution.

Game 0. This is the real experiment.

Game 1. This game is identical to **Game 0** except that each NIZK generated by a customer at any stage of the Pay protocol interaction is replaced with a simulated proof. Note that we

[3]This is necessary for certain proof systems such as [99].

require all legal adversaries to refuse to proceed subsequent to the failure of any Pay protocol interaction, and we provide this information to $\mathcal{S}$. Thus, If the proof system is zero-knowledge, then $\mathsf{Adv}\textbf{Game 1} \leq \mathrm{negl}_1(\lambda)$.

Game 2. This game is identical to **Game 1** except that the commitment wCom' is replaced with a commitment to a random message. If the commitment scheme is (computationally) hiding, then $\mathsf{Adv}\textbf{Game 2} - \mathsf{Adv}\textbf{Game 1} \leq \mathrm{negl}_1(\lambda)$.

Game 3. This game is identical to **Game 2** except that the value *wpk* is replaced with a random key generated using the KeyGen algorithm. Note that the distribution of the replacement *wpk* value is identical to the distribution of the original value, hence $\mathsf{Adv}\textbf{Game 3} - \mathsf{Adv}\textbf{Game 2} = 0$.

By summation over the hybrids, we have that $\mathsf{Adv}\textbf{Game 3}$ is negligible in the security parameter. Since **Game 3** is identical to the Ideal experiment, the bidirectional scheme is anonymous.

C.4.2 Balance

To win the Balance game, a malicious adversary $\mathcal{A}$ must claim more money than actually available, as measured by her expenditures and channel openings. We proceed by describing a simulated experiment in which $\mathcal{A}$ wins the Balance game with probability 0, and proceed to show that the real protocol interaction is computationally indistinguishable from this simulation, under the assumptions that (1) the ZK proof system is simulation-extractable, (2) the signature scheme is EU-CMA secure, (3) the commitment scheme is secure. To complete this argument, let us first define the following hybrids.

Game 0. This is the real experiment.

Game 1. This game is identical to **Game 0** except that we extract on every proof π_1, π_2 in the Establish and Pay protocols and abort if the extractor fails. By the soundness of the proof system, $\mathsf{Adv}\textbf{Game 1} \leq \mathrm{negl}_1(\lambda)$.

Game 2. This game is identical to **Game 1** except that we abort if $\mathcal{A}$ ever presents a collision in wCom (*e.g.,* in the witness to any proof of knowledge). Assuming that the commitment scheme is binding, $\mathsf{Adv}\textbf{Game 2} - \mathsf{Adv}\textbf{Game 1} \leq \mathrm{negl}_2(\lambda)$.

Game 3. This game is identical to **Game 1** except that we abort if the extracted signature on wCom is not on a message signed by the merchant (as indicated by the witnesses extracted in the first game). Under the assumption that the signature scheme is EU-CMA, we have that $\mathsf{Adv}\textbf{Game 3} - \mathsf{Adv}\textbf{Game 2} \leq \mathrm{negl}_3(\lambda)$.

Game 4. This game is identical to **Game 2**, except we abort if σ_w in the refund transaction was not one produced by the merchant. Under the assumption that the signature scheme is EU-CMA, we have that $\mathsf{Adv}\textbf{Game 4} - \mathsf{Adv}\textbf{Game 3} \leq \mathrm{negl}_4(\lambda)$.

In the following we will argue that no alllowed adversary can succeed in the Balance game against **Game 4**. By summation over the hybrids we have that **Game 4** s indistinguishable from **Game 0**, and this implies that all allowed adversaries will succeed with at most negligible advantage against the real protocol.

Let $\mathcal{A}$ be a p.p.t. adversary that succeeds with non-negligible advantage in the Balance game. We argue that this implies one of the following events has occurred:

1. The adversarial customer has presented a signature σ_w (as a witness) that was not issued by the merchant. This cannot occur in **Game 4** as it would imply an abort due to a signature forgery.

2. The adversarial customer has forged a zero-knowledge proof. This cannot occur in **Game 4** as all proofs produce valid witnesses.

3. The adversarial customer has identified a collision in the commitment scheme. This cannot occur in **Game 4** as it would cause an abort.

4. The adversarial merchant has produced a refund token σ_{rev} that the honest customer did not produce. This cannot occur in **Game 4** as it would imply an abort due to a signature forgery.

Since these events do not occur in **Game 4**, the advantage of an adversary succeeding in this game is 0. This concludes the sketch $\square$

C.5 Additional assumptions for the PRF

In this section we briefly sketch a proof that the Dodis-Yampolskiy pseudorandom function [94] provides strong pre-image resistance if the q-DBDHI assumption holds in $\mathbb{G}$.

The Dodis-Yampolskiy PRF. Let p be a prime and let $\mathcal{I} \subset \mathbb{Z}_p \setminus \{0\}$ be a polynomially-sized input space. The public parameters for the Dodis-Yampolskiy PRF are a group $\mathbb{G}$ of prime order p with generator g. The seed is a random element $s \in \mathbb{Z}_p$ and the pseudorandom function is computed as $f_s(x) = g^{1/(s+x)}$. Security for the PRF over input space $\mathcal{I}$ with $|\mathcal{I}| = q$ is shown to hold under the q-DBDHI assumption in [94].

The Dodis-Yampolskiy PRF provides strong pre-image resistance. We now sketch a proof that the Dodis-Yampolskiy PRF provides strong pre-image resistance for a polynomially-sized domain under the q-DBDHI assumption.

Our proof proceeds as follows. Let $\mathcal{A}$ be a p.p.t. adversary that, given access to an oracle F_s^{DY} implementing the Dodis-Yampolskiy PRF with an honestly-generated seed s (with the restriction that $\mathcal{A}$ can query only on elements in $\mathcal{I}$) such that with non-negligible probability $\mathcal{A}$ outputs (x, s', x') with $x, x' \in \mathcal{I}$ and $F_s^{DY}(x) = F_{s'}^{DY}(x')$. We show that $\mathcal{A}$'s output can be used to recover the seed for any PRF instance, thus violating the pseudorandomness property of the PRF.

To show this last step, we construct a distinguisher $\mathcal{B}$ against the pseudorandomness of the Dodis-Yampolskiy scheme. $\mathcal{B}$ runs $\mathcal{A}$ internally and interacts with an oracle that implements either the PRF or a random function. Each time $\mathcal{A}$ queries on some value x_i, $\mathcal{B}$ queries its oracle on the

same value and returns the response to $\mathcal{A}$. When $\mathcal{A}$ outputs (x, s', x') such that $F_s^{DY}(x) = F_{s'}^{DY}(x')$, $\mathcal{B}$ computes a candidate guess for the PRF seed as $\bar{s} = s' + x' - x$, and tests to see whether two or more distinct outputs it receives from its oracle are consistent with $\bar{s}$. If so, $\mathcal{B}$ outputs 1.

If $\mathcal{B}$ is interacting with an instance of the PRF, then $\mathcal{A}$ will succeed with non-negligible probability. In this instance, the value $\bar{s}$ will be equal to the PRF seed, because if $F_s^{DY}(x) = F_{s'}^{DY}(x')$ then this implies the relation $g^{1/s+x} = g^{1/s'+x'}$ and thus $s + x = s' + x'$, yielding $s = s' + x' - x$. If $\mathcal{B}$ is interacting with a random function, then there is no seed to recover, and the probability that multiple oracle outputs are consistent with a recovered candidate seed is negligible. Thus $\mathcal{B}$ succeeds with non-negligible probability. Since the pseudorandomness of the Dodis-Yampolskiy PRF is shown to hold under the q-DBDHI assumption, this implies that the strong pre-image resistance must also hold if q-DBDHI holds in $\mathbb{G}$.

Other PRFs. While we recommend using the Dodis-Yampolskiy PRF for our constructions, the strong pre-image resistance property holds for other PRFs. For example, hash-based PRFs such as HMAC provide this property under the assumption that the underlying hash function is collision-resistant, since the equality of two distinct outputs implies a collision in the hash function.

Bibliography

[1] F. Reid and M. Harrigan, "An analysis of anonymity in the bitcoin system," in *PASSAT/SocialCom 2011, Privacy, Security, Risk and Trust (PASSAT), 2011 IEEE Third International Conference on and 2011 IEEE Third International Conference on Social Computing (SocialCom), Boston, MA, USA, 9-11 Oct., 2011*, 2011, pp. 1318–1326. [Online]. Available: https://doi.org/10.1109/PASSAT/SocialCom.2011.79

[2] S. Meiklejohn, M. Pomarole, G. Jordan, K. Levchenko, D. McCoy, G. M. Voelker, and S. Savage, "A fistful of bitcoins: characterizing payments among men with no names," in *Proceedings of the 2013 Internet Measurement Conference, IMC 2013, Barcelona, Spain, October 23-25, 2013*, 2013, pp. 127–140. [Online]. Available: http://doi.acm.org/10.1145/2504730.2504747

[3] Chainalysis, "Chainalysis inc," https://chainalysis.com/, 2015.

[4] Elliptic, "Elliptic enterprises limited," https://www.elliptic.co/, 2013.

[5] J. Buntinx, "Bitcoin Network Backlog Grows To Over 165,000 Unconfirmed Transactions," 2017.

[6] [Online]. Available: https://blockchain.info/charts/transactions-per-second?timespan=all

[7] C. Pacia, "Lightning Network skepticism," https://chrispacia.wordpress.com/2015/12/23/lightning-network-skepticism/#more-3249, December 2015.

[8] C. Decker and R. Wattenhofer, "A fast and scalable payment network with bitcoin duplex micropayment channels," in *Stabilization, Safety, and Security of Distributed Systems - 17th International Symposium, SSS 2015, Edmonton, AB, Canada, August 18-21, 2015, Proceedings*, 2015, pp. 3–18. [Online]. Available: https://doi.org/10.1007/978-3-319-21741-3_1

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

[10] I. Eyal and E. G. Sirer, "Majority is not enough: Bitcoin mining is vulnerable," in *Financial Cryptography and Data Security - 18th International Conference, FC 2014, Christ Church, Barbados, March 3-7, 2014, Revised Selected Papers*, 2014, pp. 436–454. [Online]. Available: https://doi.org/10.1007/978-3-662-45472-5_28

[11] D. Chaum, "Blind signatures for untraceable payments," in *Advances in Cryptology*, 1983.

[12] D. Chaum, A. Fiat, and M. Naor, "Untraceable electronic cash," in *Advances in Cryptology - CRYPTO '88, 8th Annual International Cryptology Conference, Santa Barbara, California, USA, August 21-25, 1988, Proceedings*, 1988, pp. 319–327. [Online]. Available: https://doi.org/10.1007/0-387-34799-2_25

[13] S. Brands, "Rapid demonstration of linear relations connected by boolean operators," in *Advances in Cryptology - EUROCRYPT '97, International Conference on the Theory and Application of Cryptographic Techniques, Konstanz, Germany, May 11-15, 1997, Proceeding*, 1997, pp. 318–333. [Online]. Available: https://doi.org/10.1007/3-540-69053-0_22

[14] J. Camenisch, S. Hohenberger, and A. Lysyanskaya, "Compact e-cash," in *Advances in Cryptology - EUROCRYPT 2005, 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Aarhus, Denmark, May 22-26, 2005, Proceedings*, 2005, pp. 302–321. [Online]. Available: https://doi.org/10.1007/11426639_18

[15] E. Ben-Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, and M. Virza,

"Zerocash: Decentralized anonymous payments from bitcoin," in *2014 IEEE Symposium on Security and Privacy, SP 2014, Berkeley, CA, USA, May 18-21, 2014*, 2014, pp. 459–474. [Online]. Available: https://doi.org/10.1109/SP.2014.36

[16] J. C. Benaloh and M. de Mare, "One-way accumulators: A decentralized alternative to digital sinatures (extended abstract)," in *Advances in Cryptology - EUROCRYPT '93, Workshop on the Theory and Application of of Cryptographic Techniques, Lofthus, Norway, May 23-27, 1993, Proceedings*, 1993, pp. 274–285. [Online]. Available: https://doi.org/10.1007/3-540-48285-7_24

[17] C. Schnorr, "Efficient signature generation by smart cards," *J. Cryptology*, vol. 4, no. 3, pp. 161–174, 1991. [Online]. Available: https://doi.org/10.1007/BF00196725

[18] E. Ben-Sasson, A. Chiesa, D. Genkin, E. Tromer, and M. Virza, "Snarks for C: verifying program executions succinctly and in zero knowledge," in *Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part II*, 2013, pp. 90–108. [Online]. Available: https://doi.org/10.1007/978-3-642-40084-1_6

[19] H. Pagnia and F. C. Gärtner, "On the impossibility of fair exchange without a trusted third party," Technical Report TUD-BS-1999-02, Darmstadt University of Technology, Department of Computer Science, Darmstadt, Germany, Tech. Rep., 1999.

[20] C. Garman, M. Green, and I. Miers, "Accountable privacy for decentralized anonymous payments," in *Financial Cryptography and Data Security - 20th International Conference, FC 2016, Christ Church, Barbados, February 22-26, 2016, Revised Selected Papers*, 2016, pp. 81–98. [Online]. Available: https://doi.org/10.1007/978-3-662-54970-4_5

[21] A. E. Kosba, A. Miller, E. Shi, Z. Wen, and C. Papamanthou, "Hawk: The blockchain model of cryptography and privacy-preserving smart contracts," in *IEEE Symposium on Security and Privacy, SP 2016, San Jose, CA, USA, May 22-26, 2016*, 2016, pp. 839–858. [Online]. Available: https://doi.org/10.1109/SP.2016.55

[22] A. Chiesa, M. Green, J. Liu, P. Miao, I. Miers, and P. Mishra, "Decentralized anonymous micropayments," in *Advances in Cryptology - EUROCRYPT 2017 - 36th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Paris, France, April 30 - May 4, 2017, Proceedings, Part II*, 2017, pp. 609–642. [Online]. Available: https://doi.org/10.1007/978-3-319-56614-6_21

[23] I. Miers, C. Garman, M. Green, and A. D. Rubin, "Zerocoin: Anonymous distributed e-cash from bitcoin," in *2013 IEEE Symposium on Security and Privacy, SP 2013, Berkeley, CA, USA, May 19-22, 2013*, 2013, pp. 397–411. [Online]. Available: https://doi.org/10.1109/SP.2013.34

[24] S. Barber, X. Boyen, E. Shi, and E. Uzun, "Bitter to better - how to make bitcoin a better currency," in *Financial Cryptography and Data Security - 16th International Conference, FC 2012, Kralendijk, Bonaire, Februray 27-March 2, 2012, Revised Selected Papers*, 2012, pp. 399–414. [Online]. Available: https://doi.org/10.1007/978-3-642-32946-3_29

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

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

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

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

[29] R. Cramer, I. Damgård, and B. Schoenmakers, "Proofs of partial knowledge and simplified design of witness hiding protocols," in *Advances in Cryptology - CRYPTO '94, 14th Annual International Cryptology Conference, Santa Barbara, California, USA, August 21-25, 1994, Proceedings*, 1994, pp. 174–187. [Online]. Available: https://doi.org/10.1007/3-540-48658-5_19

[30] J. Camenisch and M. Michels, "Proving in zero-knowledge that a number is the product of two safe primes," in *Advances in Cryptology - EUROCRYPT '99, International Conference on the Theory and Application of Cryptographic Techniques, Prague, Czech Republic, May 2-6, 1999, Proceeding*, 1999, pp. 107–122. [Online]. Available: https://doi.org/10.1007/3-540-48910-X_8

[31] J. Camenisch, "Group signature schemes and payment systems based on the discrete logarithm problem," Ph.D. dissertation, ETH Zurich, Zürich, Switzerland, 1998. [Online]. Available: http://d-nb.info/953066045

[32] A. Fiat and A. Shamir, "How to prove yourself: Practical solutions to identification and signature problems," in *Advances in Cryptology - CRYPTO '86, Santa Barbara, California, USA, 1986, Proceedings*, 1986, pp. 186–194. [Online]. Available: https://doi.org/10.1007/3-540-47721-7_12

[33] M. Chase and A. Lysyanskaya, "On signatures of knowledge," in *Advances in Cryptology - CRYPTO 2006, 26th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 2006, Proceedings*, 2006, pp. 78–96. [Online]. Available: https://doi.org/10.1007/11818175_5

[34] J. Camenisch and M. Stadler, "Efficient group signature schemes for large groups (extended abstract)," in *Advances in Cryptology - CRYPTO '97, 17th Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 1997, Proceedings*, 1997, pp. 410–424. [Online]. Available: https://doi.org/10.1007/BFb0052252

[35] N. Bari and B. Pfitzmann, "Collision-free accumulators and fail-stop signature schemes without trees," in *Advances in Cryptology - EUROCRYPT '97, International Conference on the Theory and Application of Cryptographic Techniques, Konstanz, Germany, May 11-15, 1997, Proceeding*, 1997, pp. 480–494. [Online]. Available: https://doi.org/10.1007/3-540-69053-0_33

[36] J. Camenisch and A. Lysyanskaya, "Dynamic accumulators and application to efficient revocation of anonymous credentials," in *Advances in Cryptology - CRYPTO 2002, 22nd Annual*

International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 2002, Proceedings, 2002, pp. 61–76. [Online]. Available: https://doi.org/10.1007/3-540-45708-9_5

[37] T. Sander, "Efficient accumulators without trapdoor extended abstracts," in *Information and Communication Security, Second International Conference, ICICS'99, Sydney, Australia, November 9-11, 1999, Proceedings*, 1999, pp. 252–262. [Online]. Available: https://doi.org/10.1007/978-3-540-47942-0_21

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

[39] 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," *J. Cryptographic Engineering*, vol. 3, no. 2, pp. 111–128, 2013. [Online]. Available: https://doi.org/10.1007/s13389-013-0057-3

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

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

[42] T. Okamoto and K. Ohta, "Universal electronic cash," ser. Lecture Notes in Computer Science, J. Feigenbaum, Ed. Springer Berlin / Heidelberg, 1992, vol. 576, pp. 324–337.

[43] T. Okamoto, "An efficient divisible electronic cash scheme," in *Advances in Cryptology - CRYPTO '95, 15th Annual International Cryptology Conference, Santa Barbara, California, USA, August 27-31, 1995, Proceedings*, 1995, pp. 438–451. [Online]. Available: https://doi.org/10.1007/3-540-44750-4_35

[44] T. Sander and A. Ta-Shma, "Auditable, anonymous electronic cash extended abstract," in *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*, 1999, pp. 555–572. [Online]. Available: https://doi.org/10.1007/3-540-48405-1_35

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

[46] A. Narayanan and V. Shmatikov, "De-anonymizing social networks," in *30th IEEE Symposium on Security and Privacy (S&P 2009), 17-20 May 2009, Oakland, California, USA*, 2009, pp. 173–187. [Online]. Available: https://doi.org/10.1109/SP.2009.22

[47] L. Backstrom, C. Dwork, and J. M. Kleinberg, "Wherefore art thou r3579x?: anonymized social networks, hidden patterns, and structural steganography," in *Proceedings of the 16th International Conference on World Wide Web, WWW 2007, Banff, Alberta, Canada, May 8-12, 2007*, 2007, pp. 181–190. [Online]. Available: http://doi.acm.org/10.1145/1242572.1242598

[48] A. Narayanan and V. Shmatikov, "Robust de-anonymization of large sparse datasets," in *2008 IEEE Symposium on Security and Privacy (S&P 2008), 18-21 May 2008, Oakland, California, USA*, 2008, pp. 111–125. [Online]. Available: https://doi.org/10.1109/SP.2008.33

[49] D. Ron and A. Shamir, "Quantitative analysis of the full bitcoin transaction graph," in *Financial Cryptography and Data Security - 17th International Conference, FC 2013, Okinawa, Japan, April 1-5, 2013, Revised Selected Papers*, 2013, pp. 6–24. [Online]. Available: https://doi.org/10.1007/978-3-642-39884-1_2

[50] L. Nguyen, "Accumulators from bilinear pairings and applications," in *Topics in Cryptology - CT-RSA 2005, The Cryptographers' Track at the RSA Conference 2005, San Francisco, CA, USA, February 14-18, 2005, Proceedings*, 2005, pp. 275–292. [Online]. Available: https://doi.org/10.1007/978-3-540-30574-3_19

[51] M. Chase, A. Healy, A. Lysyanskaya, T. Malkin, and L. Reyzin, "Mercurial commitments with applications to zero-knowledge sets," in *Advances in Cryptology - EUROCRYPT 2005, 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Aarhus, Denmark, May 22-26, 2005, Proceedings*, 2005, pp. 422–439. [Online]. Available: https://doi.org/10.1007/11426639_25

[52] J. Camenisch and A. Lysyanskaya, "An efficient system for non-transferable anonymous credentials with optional anonymity revocation," in *Advances in Cryptology - EUROCRYPT 2001, International Conference on the Theory and Application of Cryptographic Techniques, Innsbruck, Austria, May 6-10, 2001, Proceeding*, 2001, pp. 93–118. [Online]. Available: https://doi.org/10.1007/3-540-44987-6_7

[53] J. Groth, "Short pairing-based non-interactive zero-knowledge arguments," in *Advances in Cryptology - ASIACRYPT 2010 - 16th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 5-9, 2010. Proceedings*, 2010, pp. 321–340. [Online]. Available: https://doi.org/10.1007/978-3-642-17373-8_19

[54] H. Lipmaa, "Progression-free sets and sublinear pairing-based non-interactive zero-knowledge arguments," in *Theory of Cryptography - 9th Theory of Cryptography Conference, TCC 2012, Taormina, Sicily, Italy, March 19-21, 2012. Proceedings*, 2012, pp. 169–189. [Online]. Available: https://doi.org/10.1007/978-3-642-28914-9_10

[55] N. Bitansky, A. Chiesa, Y. Ishai, R. Ostrovsky, and O. Paneth, "Succinct non-interactive arguments via linear interactive proofs," in *Theory of Cryptography - 10th Theory of Cryptography Conference, TCC 2013, Tokyo, Japan, March 3-6, 2013. Proceedings*, 2013, pp. 315–333. [Online]. Available: https://doi.org/10.1007/978-3-642-36594-2_18

[56] R. Gennaro, C. Gentry, B. Parno, and M. Raykova, "Quadratic span programs and succinct nizks without pcps," in *Advances in Cryptology - EUROCRYPT 2013, 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 26-30, 2013. Proceedings*, 2013, pp. 626–645. [Online]. Available: https://doi.org/10.1007/978-3-642-38348-9_37

[57] B. Parno, J. Howell, C. Gentry, and M. Raykova, "Pinocchio: Nearly practical verifiable computation," in *2013 IEEE Symposium on Security and Privacy, SP 2013, Berkeley, CA, USA, May 19-22, 2013*, 2013, pp. 238–252. [Online]. Available: https://doi.org/10.1109/SP.2013.47

[58] H. Lipmaa, "Succinct non-interactive zero knowledge arguments from span programs and linear error-correcting codes," in *Advances in Cryptology - ASIACRYPT 2013 - 19th International Conference on the Theory and Application of Cryptology and Information Security, Bengaluru, India, December 1-5, 2013, Proceedings, Part I*, 2013, pp. 41–60. [Online]. Available: https://doi.org/10.1007/978-3-642-42033-7_3

[59] E. Ben-Sasson, A. Chiesa, E. Tromer, and M. Virza, "Succinct non-interactive zero knowledge for a von neumann architecture," in *Proceedings of the 23rd USENIX Security Symposium, San Diego, CA, USA, August 20-22, 2014.*, 2014, pp. 781–796. [Online]. Available: https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/ben-sasson

[60] C. Decker and R. Wattenhofer, "Information propagation in the bitcoin network," in *13th IEEE International Conference on Peer-to-Peer Computing, IEEE P2P 2013, Trento, Italy, September 9-11, 2013, Proceedings*, 2013, pp. 1–10. [Online]. Available: https://doi.org/10.1109/P2P.2013.6688704

[61] E. Ben-Sasson, "Universal and affordable computational integrity," May 2013, bitcoin 2013: The Future of Payments. [Online]. Available: http://www.youtube.com/watch?v=YRcPReUpkcU&feature=youtu.be&t=26m6s

[62] G. Danezis, C. Fournet, M. Kohlweiss, and B. Parno, "Pinocchio coin: building zerocoin from a succinct pairing-based proof system," in *PETShop'13, Proceedings of the 2013 ACM Workshop on Language Support for Privacy-Enhancing Technologies, Co-located with CCS 2013, November 4, 2013, Berlin, Germany*, 2013, pp. 27–30. [Online]. Available: http://doi.acm.org/10.1145/2517872.2517878

[63] S. Goldwasser, S. Micali, and C. Rackoff, "The knowledge complexity of interactive proof systems," *SIAM J. Comput.*, vol. 18, no. 1, pp. 186–208, 1989. [Online]. Available: https://doi.org/10.1137/0218012

[64] M. Bellare, A. Boldyreva, A. Desai, and D. Pointcheval, "Key-privacy in public-key encryption," in *Advances in Cryptology - ASIACRYPT 2001, 7th International Conference on the Theory and Application of Cryptology and Information Security, Gold Coast, Australia, December 9-13, 2001, Proceedings*, 2001, pp. 566–582. [Online]. Available: https://doi.org/10.1007/3-540-45682-1_33

[65] N. Bitansky, R. Canetti, A. Chiesa, and E. Tromer, "From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again," in *Innovations in Theoretical Computer Science 2012, Cambridge, MA, USA, January 8-10, 2012*, 2012, pp. 326–349. [Online]. Available: http://doi.acm.org/10.1145/2090236.2090263

[66] ——, "Recursive composition and bootstrapping for SNARKS and proof-carrying data," in *Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013*, 2013, pp. 111–120. [Online]. Available: http://doi.acm.org/10.1145/2488608.2488623

[67] J. Groth, R. Ostrovsky, and A. Sahai, "Perfect non-interactive zero knowledge for NP," vol. 2005, 2005, p. 290. [Online]. Available: http://eprint.iacr.org/2005/290

[68] ——, "Non-interactive zaps and new techniques for NIZK," in *Advances in Cryptology - CRYPTO 2006, 26th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 2006, Proceedings*, 2006, pp. 97–111. [Online]. Available: https://doi.org/10.1007/11818175_6

[69] D. Boneh and X. Boyen, "Secure identity based encryption without random oracles," in *Advances in Cryptology - CRYPTO 2004, 24th Annual International CryptologyConference, Santa Barbara, California, USA, August 15-19, 2004, Proceedings*, 2004, pp. 443–459. [Online]. Available: https://doi.org/10.1007/978-3-540-28628-8_27

[70] R. Gennaro, "Multi-trapdoor commitments and their applications to proofs of knowledge secure under concurrent man-in-the-middle attacks," in *Advances in Cryptology - CRYPTO 2004, 24th Annual International CryptologyConference, Santa Barbara,*

California, USA, August 15-19, 2004, Proceedings, 2004, pp. 220–236. [Online]. Available: https://doi.org/10.1007/978-3-540-28628-8_14

[71] C. Gentry and D. Wichs, "Separating succinct non-interactive arguments from all falsifiable assumptions," in *Proceedings of the 43rd ACM Symposium on Theory of Computing, STOC 2011, San Jose, CA, USA, 6-8 June 2011*, 2011, pp. 99–108. [Online]. Available: http://doi.acm.org/10.1145/1993636.1993651

[72] S. Micali, "Computationally sound proofs," *SIAM J. Comput.*, vol. 30, no. 4, pp. 1253–1298, 2000. [Online]. Available: https://doi.org/10.1137/S0097539795284959

[73] P. Valiant, "Incrementally verifiable computation or proofs of knowledge imply time/space efficiency," in *Theory of Cryptography, Fifth Theory of Cryptography Conference, TCC 2008, New York, USA, March 19-21, 2008.*, 2008, pp. 1–18. [Online]. Available: https://doi.org/10.1007/978-3-540-78524-8_1

[74] L. Babai, L. Fortnow, L. A. Levin, and M. Szegedy, "Checking computations in polylogarithmic time," in *Proceedings of the 23rd Annual ACM Symposium on Theory of Computing, May 5-8, 1991, New Orleans, Louisiana, USA*, 1991, pp. 21–31. [Online]. Available: http://doi.acm.org/10.1145/103418.103428

[75] E. Ben-Sasson, O. Goldreich, P. Harsha, M. Sudan, and S. P. Vadhan, "Short pcps verifiable in polylogarithmic time," in *20th Annual IEEE Conference on Computational Complexity (CCC 2005), 11-15 June 2005, San Jose, CA, USA*, 2005, pp. 120–134. [Online]. Available: https://doi.org/10.1109/CCC.2005.27

[76] E. Ben-Sasson, A. Chiesa, D. Genkin, and E. Tromer, "On the concrete efficiency of probabilistically-checkable proofs," in *Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013*, 2013, pp. 585–594. [Online]. Available: http://doi.acm.org/10.1145/2488608.2488681

[77] ——, "Fast reductions from rams to delegatable succinct constraint satisfaction problems: extended abstract," in *Innovations in Theoretical Computer Science, ITCS '13, Berkeley, CA, USA, January 9-12, 2013*, 2013, pp. 401–414. [Online]. Available: http://doi.acm.org/10.1145/2422436.2422481

[78] J. Katz and Y. Lindell, *Introduction to Modern Cryptography*. Chapman & Hall/CRC, 2007.

[79] M. Bellare, "New proofs for NMAC and HMAC: security without collision-resistance," in *Advances in Cryptology - CRYPTO 2006, 26th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 2006, Proceedings*, 2006, pp. 602–619. [Online]. Available: https://doi.org/10.1007/11818175_36

[80] P. Wuille, "Proposed BIP for dealing with malleability," Available at https://gist.github.com/sipa/8907691, 2014.

[81] Certicom Research, "SEC 1: Elliptic curve cryptography," 2000. [Online]. Available: http://www.secg.org/collateral/sec1_final.pdf

[82] National Institute of Standards and Technology, "FIPS PUB 180-4: Secure Hash Standard," http://csrc.nist.gov/publications/PubsFIPS.html, 2012.

[83] PolarSSL, "PolarSSL," http://polarssl.org, Oct 2013.

[84] R. Dingledine, N. Mathewson, and P. F. Syverson, "Tor: The second-generation onion router," in *Proceedings of the 13th USENIX Security Symposium, August 9-13, 2004, San Diego, CA, USA*, 2004, pp. 303–320. [Online]. Available: http://www.usenix.org/publications/library/proceedings/sec04/tech/dingledine.html

[85] G. Danezis, R. Dingledine, and N. Mathewson, "Mixminion: Design of a type III anonymous remailer protocol," in *2003 IEEE Symposium on Security and Privacy (S&P 2003), 11-14 May 2003, Berkeley, CA, USA*, 2003, pp. 2–15. [Online]. Available: https://doi.org/10.1109/SECPRI.2003.1199323

[86] T. B. Lee, "Bitcoin needs to scale by a factor of 1000 to compete with Visa. here's how to do it." The Washington Post (http://www.washingtonpost.com), November 2013.

[87] M. D. Green and I. Miers, "Bolt: Anonymous payment channels for decentralized currencies," p. 701, 2016. [Online]. Available: http://eprint.iacr.org/2016/701

[88] D. Chaum, A. Fiat, and M. Naor, "Untraceable electronic cash," in *Advances in Cryptology - CRYPTO '88, 8th Annual International Cryptology Conference, Santa Barbara, California, USA, August 21-25, 1988, Proceedings*, 1988, pp. 319–327. [Online]. Available: https://doi.org/10.1007/0-387-34799-2_25

[89] S. Brands, "Untraceable off-line cash in wallets with observers (extended abstract)," in *Advances in Cryptology - CRYPTO '93, 13th Annual International Cryptology Conference, Santa Barbara, California, USA, August 22-26, 1993, Proceedings*, 1993, pp. 302–318. [Online]. Available: https://doi.org/10.1007/3-540-48329-2_26

[90] J. Camenisch, S. Hohenberger, and A. Lysyanskaya, "Compact e-cash," in *Advances in Cryptology - EUROCRYPT 2005, 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Aarhus, Denmark, May 22-26, 2005, Proceedings*, 2005, pp. 302–321. [Online]. Available: https://doi.org/10.1007/11426639_18

[91] J. Poon and T. Dryja, "The Bitcoin Lightning Network: Scalable Off-Chain Instant Payments," https://lightning.network/lightning-network-paper.pdf, January 2016.

[92] E. Heilman, F. Baldimtsi, L. Alshenibr, A. Scafuro, and S. Goldberg, "Tumblebit: An untrusted tumbler for bitcoin-compatible anonymous payments," *IACR Cryptology ePrint Archive*, vol. 2016, p. 575, 2016. [Online]. Available: http://eprint.iacr.org/2016/575

[93] T. P. Pedersen, "Non-interactive and information-theoretic secure verifiable secret sharing," in *Advances in Cryptology - CRYPTO '91, 11th Annual International Cryptology Conference,*

Santa Barbara, California, USA, August 11-15, 1991, Proceedings, 1991, pp. 129–140. [Online]. Available: https://doi.org/10.1007/3-540-46766-1_9

[94] Y. Dodis and A. Yampolskiy, "A verifiable random function with short proofs and keys," in *Public Key Cryptography - PKC 2005, 8th International Workshop on Theory and Practice in Public Key Cryptography, Les Diablerets, Switzerland, January 23-26, 2005, Proceedings*, 2005, pp. 416–431. [Online]. Available: https://doi.org/10.1007/978-3-540-30580-4_28

[95] J. Camenisch and A. Lysyanskaya, "A signature scheme with efficient protocols," in *Security in Communication Networks, Third International Conference, SCN 2002, Amalfi, Italy, September 11-13, 2002. Revised Papers*, 2002, pp. 268–289. [Online]. Available: https://doi.org/10.1007/3-540-36413-7_20

[96] M. Belenkiy, M. Chase, M. Kohlweiss, and A. Lysyanskaya, "P-signatures and noninteractive anonymous credentials," in *Theory of Cryptography, Fifth Theory of Cryptography Conference, TCC 2008, New York, USA, March 19-21, 2008.*, 2008, pp. 356–374. [Online]. Available: https://doi.org/10.1007/978-3-540-78524-8_20

[97] J. Camenisch and A. Lysyanskaya, "Signature schemes and anonymous credentials from bilinear maps," in *Advances in Cryptology - CRYPTO 2004, 24th Annual International CryptologyConference, Santa Barbara, California, USA, August 15-19, 2004, Proceedings*, 2004, pp. 56–72. [Online]. Available: https://doi.org/10.1007/978-3-540-28628-8_4

[98] J. Camenisch, G. Neven, and A. Shelat, "Simulatable adaptive oblivious transfer," in *Advances in Cryptology - EUROCRYPT 2007, 26th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Barcelona, Spain, May 20-24, 2007, Proceedings*, 2007, pp. 573–590. [Online]. Available: https://doi.org/10.1007/978-3-540-72540-4_33

[99] J. Groth and A. Sahai, "Efficient non-interactive proof systems for bilinear groups," in *Advances in Cryptology - EUROCRYPT 2008, 27th Annual International Conference on the Theory and*

Applications of Cryptographic Techniques, Istanbul, Turkey, April 13-17, 2008. Proceedings, 2008, pp. 415–432. [Online]. Available: https://doi.org/10.1007/978-3-540-78967-3_24

[100] J. Camenisch, R. Chaabouni, and A. Shelat, "Efficient protocols for set membership and range proofs," in *Advances in Cryptology - ASIACRYPT 2008, 14th International Conference on the Theory and Application of Cryptology and Information Security, Melbourne, Australia, December 7-11, 2008. Proceedings*, 2008, pp. 234–252. [Online]. Available: https://doi.org/10.1007/978-3-540-89255-7_15

[101] F. Boudot, "Efficient proofs that a committed number lies in an interval," in *Advances in Cryptology - EUROCRYPT 2000, International Conference on the Theory and Application of Cryptographic Techniques, Bruges, Belgium, May 14-18, 2000, Proceeding*, 2000, pp. 431–444. [Online]. Available: https://doi.org/10.1007/3-540-45539-6_31

[102] J. Groth, *Simulation-Sound NIZK Proofs for a Practical Language and Constant Size Group Signatures*, 2006, pp. 444–459. [Online]. Available: https://doi.org/10.1007/11935230_29

[103] M. Belenkiy, M. Chase, M. Kohlweiss, and A. Lysyanskaya, "Compact e-cash and simulatable vrfs revisited," in *Pairing-Based Cryptography - Pairing 2009, Third International Conference, Palo Alto, CA, USA, August 12-14, 2009, Proceedings*, 2009, pp. 114–131. [Online]. Available: https://doi.org/10.1007/978-3-642-03298-1_9

[104] I. Bentov and R. Kumaresan, "How to use bitcoin to design fair protocols," in *Advances in Cryptology - CRYPTO 2014 - 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part II*, 2014, pp. 421–439. [Online]. Available: https://doi.org/10.1007/978-3-662-44381-1_24

[105] M. Andrychowicz, S. Dziembowski, D. Malinowski, and L. Mazurek, "Secure multiparty computations on bitcoin," in *2014 IEEE Symposium on Security and Privacy, SP*

2014, Berkeley, CA, USA, May 18-21, 2014, 2014, pp. 443–458. [Online]. Available: https://doi.org/10.1109/SP.2014.35

[106] M. Chase, S. Meiklejohn, and G. Zaverucha, "Algebraic macs and keyed-verification anonymous credentials," in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, Scottsdale, AZ, USA, November 3-7, 2014*, 2014, pp. 1205–1216. [Online]. Available: http://doi.acm.org/10.1145/2660267.2660328

[107] "The zcash currency," Available at https://z.cash/.

[108] G. Maxwell, "CoinJoin: Bitcoin privacy for the real world," August 2013, bitcoin Forum. [Online]. Available: https://bitcointalk.org/index.php?topic=279249.0

[109] R. Pass and A. Shelat, "Micropayments for decentralized currencies," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, October 12-6, 2015*, 2015, pp. 207–218. [Online]. Available: http://doi.acm.org/10.1145/2810103.2813713

[110] E. Heilman, F. Baldimtsi, and S. Goldberg, "Blindly signed contracts: Anonymous on-blockchain and off-blockchain bitcoin transactions," in *Financial Cryptography and Data Security - FC 2016 International Workshops, BITCOIN, VOTING, and WAHC, Christ Church, Barbados, February 26, 2016, Revised Selected Papers*, 2016, pp. 43–60. [Online]. Available: https://doi.org/10.1007/978-3-662-53357-4_4

[111] J. Ratcliff, "The Lightning Network is so great that it has all kinds of problems," http://codesuppository.blogspot.com/2016/02/the-lightning-network-is-so-great-that.html, February 2016.

[112] "The Ethereum Project," https://www.ethereum.org/.

[113] J. Camenisch and A. Lysyanskaya, "Dynamic accumulators and application to efficient revocation of anonymous credentials," in *Advances in Cryptology - CRYPTO 2002, 22nd*

Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 2002, Proceedings, 2002, pp. 61–76, extended Abstract. [Online]. Available: http://cs.brown.edu/~anna/papers/camlys02.pdf

[114] G. Danezis, "petlib: A python library that implements a number of Privacy Enhancing Technolgies," https://github.com/gdanezis/petlib.

[115] F. Baldimtsi and A. Lysyanskaya, "Anonymous credentials light," in *2013 ACM SIGSAC Conference on Computer and Communications Security, CCS'13, Berlin, Germany, November 4-8, 2013*, 2013, pp. 1087–1098. [Online]. Available: http://doi.acm.org/10.1145/2508859.2516687

[116] B. Lynn, "PBC:The Pairing-Based Cryptography Library," https://crypto.stanford.edu/pbc/.

[117] D. F. Aranha and C. P. L. Gouvêa, "RELIC is an Efficient LIbrary for Cryptography," https://github.com/relic-toolkit/relic.

[118] J. Camenisch and V. Shoup, "Practical verifiable encryption and decryption of discrete logarithms," in *Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003, Proceedings*, 2003, pp. 126–144. [Online]. Available: https://doi.org/10.1007/978-3-540-45146-4_8

[119] M. Bellare, A. Desai, E. Jokipii, and P. Rogaway, "A concrete security treatment of symmetric encryption," in *38th Annual Symposium on Foundations of Computer Science, FOCS '97, Miami Beach, Florida, USA, October 19-22, 1997*, 1997, pp. 394–403. [Online]. Available: https://doi.org/10.1109/SFCS.1997.646128

Vita

Ian Miers received a B. Sc. in Computer Science from The Johns Hopkins University in 2010. He then spent a year at Microsoft as a software engineer while living in Seattle and two years working as a research programmer in the Health and Medical Security Lab at Hopkins. He enrolled in the Computer Science Ph. D. program at Hopkins in 2013. His work focuses on applied cryptography and privacy enhancing technologies. He has worked on anonymous crypto-currencies, decentralized anonymous credentials, and secure messaging including attacks that decrypt Apple iMessage traffic and efficient searchable encryption for email. His work has been featured in the Washington Post, the New York Times, Wired, The Economist, and denounced in at least 2 op-eds. He is one of the founders of ZCash, a startup building a crypto-currency based on Zerocash.