Preserving privacy and proving reputation in decentralized token systems

Sarayu Namineni

CMU-CS-23-142
December 2023

Computer Science Department
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:
Seth Copen Goldstein (Chair)
Elaine Shi

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Computer Science.

Keywords: Privacy, blockchains, smart contracts, DApps, decentralized lending, blockchain reputation, zero-knowledge, incrementally verifiable computation

Abstract

In the decentralized token system ZUZ, users can define their own token-based currencies which accrue reputation from publicly observable actions recorded on a transparent, universally accessible ledger. We design, implement, and evalaute the cost of privacy-preserving versions of the ZUZ system for both the UTXO- and account-based models. In the UTXO-based model, we extend the Zerocash protocol to support anonymous transfers over origin, destination, and value for a multi-token system with negligible overhead over Zerocash transactions. In the account-based model, we design a privacy-preserving Ethereum smart contract which represents the ledger state more concisely, while still achieving anonymous transfers over destination and value. Our preliminary results show that transaction times scale linearly in the size of the sender's wallet and anonymity ring. Finally, we introduce a novel primitive which formalizes the notion of reputation with respect to a public ledger. This provides a starting point for how to address the conflict between the reputation of users and the anonymity of transfers in a privacy-preserving ledger.

Acknowledgments

First, I would like to thank Prof. Seth Goldstein for giving me my first opportunity to do research as an undergraduate and for continuing to guide and mentor me through the many different directions I have taken with my research projects over the past two and half years. I am grateful for his willingness to let me explore new ideas, which has contributed greatly to my ability to think independently as a researcher, and his consistent availability over the years to provide feedback on my progress, which has allowed me to improve my skills as a researcher, developer, and communicator.

I would also like to thank Prof. Elaine Shi for inspiring new directions in my research through her excellent teaching in her blockchains course, for having many conversations with me during office hours, and for supervising me as I pursued a new project over the past semester.

I would especially like to thank Abhiram Kothapalli for his guidance and mentorship over the past semester, without whom I would not have made nearly as much progress on my new project. I am grateful for him always taking the time to teach me and provide feedback on my work, which has helped me think about these new problems more deeply.

Finally, I would like to thank family and friends for all their support and encouragement, which have helped me finish my master's program.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

ZUZ [18] is a decentralized token system, which allows users to leverage their real-world reputation in order to generate capital. In particular, users can define their own token specifications, called ZUZ specifications, that are tied to their public identity. ZUZ specifications can represent any number of financial derivatives, such as interest rates, maturities, and buybacks.

Furthermore, users can mint tradeable tokens, or ZUZ instances, from the ZUZ specifications that they own. ZUZ instances represent forms of payment on the network. The owner of a ZUZ specification must always accept the ZUZ instances that they mint as a form of payment.

At first, the primary factor that contributes to whether a non-owner will accept tokens for a particular ZUZ specification is the real-world reputation of the ZUZ specification's owner. For instance, if a bakery owner creates `Bakery ZUZ`, patrons of this establishment will be incentivized to accept this token, since they can redeem it at the bakery.

Over time, however, users may begin to use the *token's* reputation, or the publicly observable transaction history over this ZUZ specification to make this decision. For instance, a user may still accept `Bakery ZUZ`, even if he or she does not have sweet tooth, because their trading partners accept payments over this ZUZ specification.

Under this system, the objectives of allowing users to generate capital and to protect the privacy of their financial transactions come into direct conflict with each other. This is because a user's ability to generate capital is determined by two factors: their real-world reputation and the publicly observable interactions over this token. The former requires them to tie their public identity to a new ZUZ specification, and the latter requires all transaction details to be publicly accessible to all network participants.

We briefly justify why these two conditions must be met in order for users to effectively generate capital. First, we claim that ZUZ specifications must be tied to public identities in order for users to leverage their existing real-world reputation. When a ZUZ specification is first created, there is no prior transaction history associated with them. Therefore, without a public identity attached to a ZUZ specification, there is not a single user to "vouch" for the trustworthiness of the ZUZ specification. As a result, other users have no reason to accept these funds as payment.

Next, we claim that transactions details must be publicly accessible to all network participants in order for the user to generate new capital. With just the public identity of a user attached to their ZUZ specification, a user's ability to generate capital is limited to users who already know

of their repuatation in the real world. In order for a user to generate new capital, they must be able to convince other users to accept their funds as payment through the publicly observable interactions on the ledger.

If some or all of a transaction's details are hidden, then the system becomes susceptible to *wash attacks*. Recall that the details of a transaction include the sender, recipient, and value of the payment. When either the sender or recipient of a transaction is hidden, the owner of ZUZ specification can falsely inflate its perceived value by creating trades between anonymous identities controlled by herself. When the value of a transaction is hidden, it is similarly possible for the owner to falsely inflate the perceived value of a ZUZ specification by fragmenting a single mint or transfer into many low-value transactions. Thus, a user's ability to effectively generate new capital using the ZUZ system relies on the transparency of the system's ledger.

In this thesis, we construct three novel protocols, which can be used to address the conflict user reputation and transactional privacy in the ZUZ system. In particular, we design, implement, and evaluate two novel privacy-preserving ledgers for the ZUZ application in the UTXO- and account-based models and we provide a construction for a new cryptographic primitive, *statements of blockchain reputation*, that allows users to efficiently prove and verify their reputation.

In Chapter 3, we implement a multi-token extension of the privacy-preserving UTXO protocol Zerocash [9] with minimal overhead. In Chapter 4, we implement a variation of the privacy-preserving smart contract Zether [10] which achieves greater privacy guarantees than Zether's confidential payment system with better scalability than Zether's fully-anonymous extension. In Chapter 5, we provide a construction for statements of blockchain reputation that uses *incrementally verifiable computation* to efficiently update and verify a user's proof of their reputation as new transactions are added to the blockchain.

Privacy-Preserving Ledger

With the rise of *privacy coins* such as Zerocash [9] and Monero [4], it is increasingly clear that users are opposed to having their transaction history be linked to their public identity [3]. Concerned with the privacy of their digital transactions, users are averse to having their interactions be publicly observable to all others on the network.

Our first attempt to conduct anonymous transactions in ZUZ is as follows. Simply put, a user takes a new pseudonym and defines a new ZUZ specification under it. Then, we claim that any instances that they mint from this scrip constitutes anonymous payment [15]. Since neither the scrip nor the pseudonym have any history associated with them, the transaction is completely anonymous. However, for the same reason, other users on the network have little to no incentive to accept these funds in a transaction.

Without a way for users to spend their existing funds privately, ZUZ is rendered unusable. As such, in Chapters 3 and 4, we design privacy-preserving ledgers for ZUZ in both the UTXO- and account-based models, and implement them on top of Bitcoin and Ethereum, respectively, and evaluate the privacy and scalability of these systems.

Efficient ZKP for Reputation

We formalize *statements of blockchain reputation* with respect to an application-specific scoring function. In particular, we *incrementally verifiable computation* [19] to prove statements of blockchain reputation with respect to a monotonic scoring function in zero knowledge. We claim that this construction is sufficient for token owners in the ZUZ system to use in order to prove the reputation of their tokens to other users.

As new transactions are added to the blockchain, a user's statement of reputation grows "outdated." A strawman attempt to update statements of blockchain reputation requires the token owner to re-compute their reputation scoring function over all transactions on the blockchain. Our work allows users to efficiently update the zero-knowledge proofs for their statements of reputation using a single-step function for each additional block that is added to the chain.

In Chapter 5, we provide a formal construction for statements of blockchain reputation with respect to a monotonic scoring function, extend our construction to the general case, and illustrate a variety of future applications for this primitive.

Chapter 2

Related Work

In permissionless networks such as Bitcoin, Ethereum, and Zerocash, new coins are created and awarded to miners when they process transactions and successfully add them to the ledger. On these networks, the creation of new money is not under the control of any individual role.

On the other hand, in permissioned networks, new coins are created by an issuer, who is usually the central bank. Sometimes, the role is distributed, such that a group of nodes must come to a consensus about issuing new coins. There is a separate role for validators, who are individuals that add transactions to the ledger. Validators may be appointed by another entity on the network, or may have incentives to work, such as transaction fees. Sometimes, these networks will have a single entity, which may or may not be distributed, perform both the roles of issuer and validator.

Out of these two settings, the ZUZ system most closely resembles a permissioned network, with a few distinctions. The first being that instead of having a single issuer authority, there are several authorities, respective to different ZUZ specification. The other difference is that these issuers are allowed to participate in transfers on the network. As such, on the ZUZ network, it is important that a ZUZ issuer does not act as a validator for the transactions over the ZUZ specifications that it owns.

Since prior work in permissioned systems only considers the case where there are designated issuers, there is no prior work that meets the first condition. As a result, we survey permissioned networks which meet the second condition, by having distinct roles for issuers and validators, in order to evaluate if they may be extensible as ZUZ networks. In particular, we consider related works in permissioned networks that provide user- or transaction-level privacy guarantees. Although no prior work addresses the notion of reputation as defined in this thesis, we consider privacy-preserving permissioned networks with the related objective of auditability.

In a decentralized anonymous payment system, there is no protection against money laundering or other financial crimes. In order to prevent abuse of these systems, there has been significant work into how to ensure regulatory compliance to a group of designated auditors. Some examples of these transaction policies are enforcing tax payments, deposit limits, and verifiable coin tracing, all while protecting the privacy of each transaction.

Androulaki et al. [5] proposes a privacy-preserving payment system where each user has a distinct auditor who can decrypt all of their transactions. The system has different roles for issuers and validators, or “certifiers," which means that may be extensible. The system’s ledger

is a decentralized data store from which any party can read and to which privileged parties, such as issuers and certifiers, can write. In particular, issuers and certifiers write all issue and transfer transactions, respectively, onto the ledger. When a user wants to transfer a token, they must request the certifier for a certificate that allows them to complete the transfer.

The role of the certifier is necessary due to the token-based nature of this system. The validity of the token is determined by its corresponding signature by the certifier. This gives the certifier the ability to link transfer transactions to a user based on certification requests. The centralization and leakage of privacy that come with the role of the certifier may be undesirable for a ZUZ system.

Barki-Gouget [7] propose a design for a central bank digital currency (CBDC) where a revocation authority outside of the network has the power to decrypt any transaction that is requested of it. Any user on the system can post transactions to the decentralized ledger, and all transactions are traceable. This means that the system achieves psuedonymity for its users, which can only be revoked by the revocation authority. In general, a pseudonymous system like this one is not ideal for a ZUZ system since users no longer have the option to choose whether an individual transaction will be public or private.

PRCash [27] allows users to spend anonymously up to a certain limit every epoch. The transaction design is based on the fully private currency MimbleWimble. There are distinct roles for issuers and validators on the network, which makes it extensible as a ZUZ system. In comparison to the previous two works, this system has the least centralization and allows users to make individual transactions public or private. However, transactions do require two steps, first from receiver to sender and then from sender to validator, and the expenditures of users are psuedonymous to the validators.

There are many other works in this field, which are not immediately extensible as ZUZ newtorks, which we briefly mention here. Garman et. al [17] was the first work to address auditability in a decentralized anonymous payment system is an extension of the privacy-preserving version of Bitcoin, Zerocash. In section 2 we demonstrate how to extend the Zerocash system as a ZUZ network and provide a preliminary implementation as well.

ZKLedger [20] is the first work which consider the general auditor case but suffers from scalability issues since it updates the balance of every account for each transaction on the network. MiniLedger [12] attempts to improve upon these scalability issues by pruning transactions. PGC [13] proposes the first decentralized confidential payment system with auditability, using a variant of ElGamal encryption to conceal transaction values and efficiently prove transaction policies over these encrypted values in zero-knowledge.

Chapter 3

UTXO-Based Ledger

3.1 Background and Motivation

The first digital currency to gain widespread adoption operates, Bitcoin, operates under the coin-based model. Under this model, each coin is attached to a unique Bitcoin address, or psuedonym. Although Bitcoins users may transact under different psuedonyms to increase their privacy, prior work has shown that it is possible to de-anonymize the transaction graph by just using information from these pseudonymous transactions, such as the network structure and the value and dates of transactions [6], [22], [23]. As a result, Bitcoins fails to offer any meaningful level of privacy to its users.

Zerocash [9] is a privacy-preserving version of Bitcoin. Zerocash allows users to use their Bitcoin to make transactions that conceal the origin, destination, and value of their payments. In particular, users mint “shielded" currency from the basecoin or “transparent" currency of the Bitcoin protocol. Then, users can spend their shielded currency anonymously, using the “pour" operation which allows shielded coins to be split, joined with other shielded coin, or optionally, converted back to basecoin. Each shielded transaction uses zk-SNARKs [8] to prove validity and correctness in zero-knowledge.

Shielded coins are represented as commitments and are added to the Merkle tree that represents the global state of all coins in the system. When a user wants to spend their shielded funds, they demonstrate their knowledge of the opening to their coin commitment using a zk-SNARK and compute a unique serial number from the opening that prevents double-spending of that coin in the future. Although zk-SNARKs can be used to efficiently prove and verify shielded transactions, one caveat is that these proofs rely on a trusted setup, or more specifically, an honestly-generated common reference string. Nevertheless, in practice, this assumption has not led to any known attacks on their system [26].

Our first attempt to design a privacy-preserving extension of the ZUZ protocol, then, is to model the system as an extension of the most widely-adopted fully anonymous payment system, Zerocash. In order to do this, we must first implement the ZUZ protocol in the coin-based model. Unlike Bitcoin, ZUZ does not have a basecoin, or native coin, by default. ZUZ specifications are user-defined and only the ZUZ specification’s owner decide to mint more instances of it. As a result, there is no one specification that can be used to substitute coinbase. In our proof of concept

implementation, we do not change the existing Bitcoin infrastructure for miner compensaton and do not implement any type of ZUZ-specific block rewards or transaction fees, leaving this decision up to future work.

3.2 Interface

We extend the ZCash RPC client to include the following functions.

1. Define : $(\mathsf{title}, [\mathsf{interest}], [\mathsf{maturity}], [\mathsf{buyback}], \mathsf{fee}) \rightarrow \mathsf{specOutput}$

   This function defines a new ZUZ specification using the given input and assigns ownership of this specification to the transaction fee payer. The owner of the ZUZ specification can spend the output from this transaction to mint instances of their specification.

   - Inputs:
     - title: Name of the ZUZ specification
     - [interest]: List of interest objects that specify an interest rate and a time period during which the rate is applied
     - [maturity]: List of maturities that specify an object (e.g. goods and services, fiat, other ZUZ specifications) that instances of this specification can be exchanged for after a certain date
     - [buyback]: List of buyback options, or objects that the owner can use to buy back all instances of their ZUZ specification
     - fee: Expenditure of an unspent transaction output as defined by the following values
       - hash: Hash of a previous transaction
       - index: Index of output in the array of the referenced transaction
       - amount: Desired amount to spend from the transaction output uniquely identified by txID and out
       - scriptPubKey: Spending script which encodes a hash of the transaction output owner's *public key*
   - Outputs:
     - specOutput: Spendable ouptut that contains the following values
       - specId: Unique ID for the ZUZ specification
       - specOwner: Hash of the fee payer's *public key*, as encoded in the fee's scriptPubKey
       - specAmount: Total number of instances created
       - specDetails: Includes details of ZUZ specification provided as input, such as title, [interest], [maturity], and [buyback]
2. Mint : $(\mathsf{specOutput}, \mathsf{amount}, \mathsf{fee}) \rightarrow (\mathsf{specOutput}, \mathsf{instance})$

   This function allows the owner of a ZUZ specification to mint new instances of it by spending the output of define ZUZ or a previous mint ZUZ transaction. By spending this output, the initiator of the transaction proves ownership over the specification. Each

mint transaction creates a new spendable output for the owner to spend on future mint transactions and transfers the specified amount of new ZUZ instances to the owner of the ZUZ specification.

- Inputs:
  - specOutput: Reference to an unspent transaction output (see Define)
  - amount: The amount of instances to mint
  - fee: Reference to an unspent transaction output (see Define)
- Outputs:
  - specOutput: Spendable output where the specAmount has been incremented by amount
  - instance: Spendable output that contains the following values
    - specId: Unique ID of the specOutput
    - owner: Public key of the owner of the specOutput
    - amount: Numerical value given by the input amount

3. Transfer : $([\text{instances}], [(\text{recipient}, \text{amount})], \text{fee}) \rightarrow [\text{instances}]$

   This function allows any user to transfer their ZUZ instances to other users by spending their existing transfer outputs and creating new transfer outputs of equal or lesser value for their recipients. Note that all transfer instances spent and created by this transaction correspond to the same ZUZ specification.

   - Inputs:
     - [instances]: List of unspent ZUZ instances (see Mint)
     - [(recipient, amount)]: List of the public keys of desired recipients and the amount of the ZUZ instance to transfer to each recipient
     - fee: Reference to an unspent transaction output (see Define)
   - Outputs:
     - [instances]: List of spendable ZUZ instances, where each instance corresponds to an entry (recipient, amount, specId) of the input array
       - specId: Unique ID given by specId
       - owner: Public key given by recipient
       - amount: Numerical value given by the input amount

4. Shield : $([\text{instances}], \text{fee}) \rightarrow [\text{shielded}]$

   This function allows a user to shield a list of their ZUZ instances, or convert them into private funds. Note that all instances spent and created by this transaction correspond to a single ZUZ specification, which is a publicly known value of this transaction.

   - Inputs:
     - [instances]: List of unspent ZUZ instances (see Transfer)
     - fee: Reference to an unspent transaction output (see Define)
   - Outputs:

- [shielded]: List of shielded ZUZ instances that contain the following values
  - specId: Unique ID given by specId
  - s: Randomly sampled commitment trapdoor
  - k: Commitment to the owner of the shielded coin

    $$k = \mathsf{COMM}_r(\mathsf{pubKey}||\rho)$$

    for some randomly sampled commitment trapdoor r and randomly sampled pseudorandom function seed ρ
  - v: Monetary value of the shielded coin
  - cm: Commitment to the value v of the shielded currency and owner pubKey of the coin

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

5. Pour : $([\text{shielded}], [(\text{recipient}, \text{amount})], \pi_{\mathsf{Pour}}) \rightarrow ([\text{shielded}], [\text{instances}])$

   This function allows a user to spend a list of private ZUZ instances corresponding to the same ZUZ specification and optionally convert their private funds back into public ZUZ instances. Since all of the inputs to the transaction have concealed values, the user must prove the correctness of their transaction in zero-knowledge. Note that the ZUZ specification is still a public parameter of all transaction outputs, public or private.

   - Inputs:
     - [shielded]: List of information needed to anonymously spend shielded coins
       - $\mathsf{rt}_{\mathsf{specId}}$: Merkle root of all shielded coin commitments pertaining to the ZUZ specification with the unique ID specId
       - $\mathsf{sn}_{\mathsf{i}}^{\mathsf{old}}$: Serial number of an unspent coin, or the output of the psuedorandom function with the key pubkey, corresponding to the owner of coin, and the randomly sampled seed $\rho_{\mathsf{i}}^{\mathsf{old}}$

         $$sn_i^{old} = \mathsf{PRF}_{\mathsf{pubKey}}(\rho_{\mathsf{i}}^{\mathsf{old}})$$

     - [(recipient, amount)]: List of transfer outputs (see Transfer)
     - π_{Pour}: Zero-knowledge proof of the user's ownership of the inputted funds and the preservation of balance between the inputted and outputted funds
   - Outputs:
     - [shielded]: List of shielded ZUZ instances (see Shield)
     - [instances]: List of public ZUZ instances (see Transfer)

3.3 Implementation

We implement the above interface by defining a new spending script that can be used to trade ZUZ instances, instead of Bitcoins [2]. We also consider how to interpret these custom spending

scripts as shielded coins, so that anonymous transfers can happen over any given ZUZ specification. In this proof of concept implementation, we interpret the specId encoded in the ZUZ spending script and encode its value into the publicly accessible memo field of shielded coins.

1. DefineTx : (title, fee) → specOutput

   We implement a simplified version of DefineTx, omitting the implementation details of financial derivatives such as interest rates, maturities, and buybacks, as the following Bitcoin transaction using the given inputs, where fee = (hash, index, amount, scriptPubKey).

   - Input: Corresponds to the transaction fee
     - hash: Hash of a previous transaction
     - index: Index of the output in the array of the referenced transaction
     - scriptLength: Length of the scriptPubKey in bytes
     - scriptSig: List of information needed to spend the output, which includes the fee payer's *public key* and the fee payer's *signature* over a modified version of the transaction
   - Output: Corresponds to the specOutput
     - value: The total number of instances minted of this ZUZ specification. Its value is 0 by default. This value will increment as a result of a MintTx.
     - scriptPubKey: A custom Bitcoin script which encodes the ownership of the output and the ZUZ spec it corresponds to: `OP_DUP OP_HASH160 <pubKeyHash> OP_EQUALVERIFY OP_CHECKSIGVERIFY <specId> OP_EQUAL`

2. MintTx : (specOutput, amount, fee) → (specOutput, instance)

   First, we validate that the specOutput provided as input to MintTx is indeed the output of DefineTx or a previous MintTx. In order to do this, we search the transaction mempool for the transaction corresponding provided hash value. Then, we inspect the structure of the transaction and the value of output at index in order to determine if this is a spendable output for a mint transaction.

   If the structure of the transaction corresponds to a DefineTx, by taking no input with a ZUZ script and returning one output with a ZUZ script, and the index corresponds to the one output with a ZUZ script, then we have a valid spendable input. If the structure of the transaction corresponds to a MintTx, by taking one input with a ZUZ script and returning one or two outputs with a ZUZ script that exceed the input value, and the index corresponds to the ZUZ output with greater value, then we also have a valid spendable input.

   We implement MintTx as the following Bitcoin transaction using the given inputs as follows.

   - Inputs:
     - Inputs corresponding to a ZUZ instance where each have the following values
       - hash: Hash of a MintTx transaction or previous TransferTx transaction
       - index: Index of the the ZUZ instance in the array of the referenced transaction
       - scriptLength: Length of the scriptPubKey in bytes

        - scriptSig: List of information needed to spend the output, which includes the fee payer's *public key*, the fee payer's *signature* over a modified version of the transaction, and the *specId*
    - Input corresponding to the transaction fee (see DefineTx)
- Outputs:
    - Output corresponding to the specOutput (see DefineTx) where the value is incremented by amount
    - Output corresponding to a new ZUZ instance
        - value: Numerical value given by the input amount
        - scriptPubKey: Assigns the ownership of the new ZUZ instances to the specification's owner using the custom ZUZ script introduced in DefineTx

3. TransferTx : $([\text{instances}], [(\text{recipient}, \text{amount})], \text{fee}) \rightarrow [\text{instances}]$

   First, we validate that the specOutput provided as input to Transfer is indeed the output of MintTx or a previous TransferTx. In order to do this, we search the transaction mempool for the transaction corresponding provided hash value. Then, we inspect the structure of the transaction and the value of output at index in order to determine if this is a spendable output for a transfer transaction.

   If the structure of the transaction corresponds to a MintTx, by taking one input with a ZUZ script and returning one output with a ZUZ script that exceeds the input value, and the index corresponds to the ZUZ output with lesser value, then we have a valid spendable input. If the structure of the transaction corresponds to a TransferTx, by having outputs with ZUZ scripts that are less than or equal in value to the inputs with ZUZ scripts, then we also have a valid spendable input.

   We implement TransferTx as the following Bitcoin transaction using the given inputs as follows.

   - Inputs:
       - Input corresponding to the specOutput
           - hash: Hash of a DefineTx transaction or previous MintTx transaction
           - index: Index of the specOutput in the array of the referenced transaction
           - scriptLength: Length of the scriptPubKey in bytes
           - scriptSig: List of information needed to spend the output, which includes the fee payer's *public key*, the fee payer's *signature* over a modified version of the transaction, and the *specId*
       - Input corresponding to the transaction fee (see DefineTx)
   - Outputs:
       - Outputs corresponding to new ZUZ instances corresponding to each tuple (recipient, amount)
           - value: Numerical value given by the input amount
           - scriptPubKey: Assigns the ownership of the instances to recipient using the custom ZUZ script introduced in DefineTx

4. $\mathsf{ShieldTx} : (\mathsf{[instances]}, \mathsf{fee}) \rightarrow \mathsf{[shielded]}$

   We implement ShieldTx as the following Zerocash transaction by encoding the specId shared by all inputted ZUZ instances in the memo field, which is publicly visible transaction data. For further details on Zerocash's implementation of shielding transparent funds, we direct readers to their whitepaper [9].

5. $\mathsf{PourTx} : (\mathsf{[shielded]}, \mathsf{[(recipient, amount)]}, \pi_{\mathsf{Pour}}) \rightarrow (\mathsf{[shielded]}, \mathsf{[instances]})$

   We implement PourTx as the following Zerocash transaction by encoding the specId shared by all outputted ZUZ instances in the memo field, which is publicly visible transaction data. For further details on Zerocash's implementation of the pour operation, we direct readers to their whitepaper [9].

3.4 Evaluation

Our preliminary implementation of the above privacy-privacy ZUZ protocol in the UTXO-based model is written in C++ as an extension of the ZCash repository. In our implementation, we define a custom `scriptPubKey`, or Bitcoin spending script, to represent the spending of distinct specifications.

In Table 3.1, we evaluate the performance of this custom spending script by measuring the transaction times and sizes of new ZUZ operations, such as defining a ZUZ specification and minting new ZUZ instances. Since a define transaction can contain an arbitrary number of user-defined financial derivatives, we only present the measurements for a baseline transaction, which only considers details like the unique ID and owner of the ZUZ specification. However, since a define transaction only happens once in the lifetime of a ZUZ specification, it is a one-time overhead that is incurred on generation. On the other hand, a mint transaction has a constant time and size complexity since it produces exactly two outputs, a new specOutput, which enables the owner to mint more ZUZ instances in the future, and the desired amount of ZUZ instances, also sent to the owner of the ZUZ specification.

In Table 3.2, we compare the transaction times and output spending script sizes for the transfer ZUZ operation to raw Bitcoin transactions, specifically the combined time of `createraw transaction`, `signrawtransaction`, and `sendrawtransaction`. Since the number of inputs and outputs of a transfer operation is variable, we have record the difference in the size of a single output, which differs in the number of operations in it spending script. In particular, since a ZUZ spending script encodes the unique ID of the specification it corresponds to, the size of script is dependent on the size of this field. We compare baseline transaction times for transferring BTC and ZUZ for a payment with a single spendable input with a single intended recipient.

Overall, transaction times for the define, mint, and transfer ZUZ operations are higher than baseline transaction times for BTC operations due to the definition of the registration and transfer outputs. In order to determine if an input to a mint transaction is the registration output produced by a previous define or mint transaction, the mempool has to be searched to find the transaction corresponding to each input. Similarly, in order to determine if an input to a transfer transaction is a transfer output produced by a previous mint or transfer transaction, the mempool has to be searched to find the transaction corresponding to each input.

ZUZ Txn	Time (ms)	Size (B)
Define	201	67
Mint	214	256

Table 3.1: Transaction times and sizes of baseline ZUZ operations

Coin	Transaction time (ms)	Size of script public key (B)
BTC	128	25
ZUZ	196	$27 + \lceil \log_{16}(\text{specId}) \rceil$

Table 3.2: Comparison of baseline transfer operations for BTC and ZUZ

Note that since we encode the spending of distinct specifications in the memo field of shielded outputs, the performance of our private transactions is identical to ZCash.

Chapter 4

Account-Based Ledger

We improve upon our privacy-preserving UTXO-based protocol with a more concise representation of public and private user funds in the account-based model. We design and implement single-step destination- and value-anonymous transfers for a privacy-preserving smart contract, which represents the ledger state more compactly. Additionally, we consider how multi-step protocols may enhance the privacy guarantees of our smart contract.

4.1 Background and Motivation

In Bitcoin, full nodes can form a concise representation of the blockchain by maintaining a set of all addresses with a nonnegative balance, or a *UTXO set* [16]. Indeed, full nodes use this representation to efficiently check for double spending, simply by checking if the given input to a transaction appears in this set or not. Note that when a transfer occurs, the size of this set stays the same; the address of the sender is removed from the UTXO set and the address of the recipient is added to the set.

However, under Zerocash, this concise representation breaks down, since addresses are only added to the UTXO set but are never removed. The size of the UTXO set is monotonically increasing, which has averse implications for the scalability of smart contract implementation of ZCash, ZETH [24]. The mixer contract maintains a deep Merkle tree, and where the entire set of leaves, along with all the corresponding serial numbers, needs to be kept in storage in perpetuity.

One notable decentralized confidential payment smart contract which maintain a concise representation of ledger state is Zether [10]. Each user's private account is represented by a single account balance, which is homomorphically encrypted, and a list of pending transfer amounts that is rolled over to user accounts at the start of each epoch.

The reason that Zether maintains a table of pending transfers is to prevent *front-running attacks*. This attack refers to the race condition where the ledger state changes between the time a client sends the transaction and a validator processes the transaction. This could happen because transactions within a block can be processed in any order.

For instance, suppose Alice wishes to privately send funds to Bob at the same time that Charlie wishes to privately send funds to Alice. If Charlie's transaction gets processed before Alice's, then Alice's private balance will be different than the state under which Alice formed

her transaction. For any system that guarantees anonymity over value, private balances should be computationally indistinguishable from each other, and it would not be desirable to leak partial ordering over the encrypted data.

In our design, we address this situation by allowing for flexibility between the account-based and the UTXO-based representation of wallets. We allow commitments from different private transfers to be appended to a user's wallet. For instance, in the previous example with Alice's transaction and Charlie's transaction, Alice's transaction can still be efficiently validated using the set of commitments she provided in her transaction, even if Charlie's commitment got added to her account before her transaction was processed, assuming that these are destination-and value-anonymous transfers.

In contrast to ZETH, we only allowing users to perform join operations over their sets of private wallet commitments, instead of both join and split operations. Whereas the flexibility with UTXO-like representation of private balances allows for handling concurrency issues, the join operation returns an account back to its concise representation, ensuring better scalability of the smart contract.

4.2 Interface

Let the smart contract ZUZ maintain a table Wallet that maps user account public keys to their ZUZ wallet and a table Spec that maps the unique ID of a ZUZ specification to the details of the specification, such as the public key of the owner, the title, interest rate, maturity dates, and buyback options.

Let a user's ZUZ wallet be represented by a table mapping ZUZ specification public keys to a tuple of balances $(\mathsf{b}_{\mathsf{pub}}, \mathsf{b}_{\mathsf{priv}})$ where $\mathsf{b}_{\mathsf{pub}}$ is a single numerical value representing the total amount of the public ZUZ instances and $\mathsf{b}_{\mathsf{priv}}$ is a list of commitments representing the private ZUZ instances.

We define the smart contract interface as follows.

1. $\mathsf{Define} : (\mathsf{title}, [\mathsf{interest}], [\mathsf{maturity}], [\mathsf{buyback}], \mathsf{fee}) \rightarrow \mathsf{specId}$

   This function defines a new ZUZ specification using the given input and assigns ownership of this specification to signer of the transaction. The owner of the ZUZ specification must use the same address to sign transactions to mint instances of their specification.

   - Inputs:
     - title: Name of the ZUZ specification
     - [interest]: List of interest objects that specify an interest rate and a time period during which the rate is applied
     - [maturity]: List of maturities that specify an object (e.g. goods and services, fiat, other ZUZ specifications) that instances of this specification can be exchanged for after a certain date
     - [buyback]: List of buyback options, or objects that the owner can use to buy back all instances of their ZUZ specification
     - fee: Gas cost of transaction, as defined by the following transaction values

- gasLimit: Upper limit on the computational resources that the transaction can consume
- maxFeePerGas: Maximum fee that the user is willing to pay per unit of gas consumed by the transaction
- maxPriorityFeePerGas: Maximum tip that the user is willing to pay to the validator per unit of gas consumed by the transaction

- Output: specId, or the public key that indexes into the account of this ZUZ specification in the table Spec

2. Mint : $(\mathsf{specId}, \mathsf{amount}, \mathsf{fee}) \to \{0,1\}$

   This function creates new ZUZ instances using the given input. The mint transaction is validated by ensuring that the transaction was signed using the ZUZ specification owner's address. If this is the case, then the sender's account balance for the given specId is increment by the requested amount.

   - Inputs:
     - specId: Public key for a ZUZ specification
     - amount: The amount of instances to mint
     - fee: Gas cost (see Define)
   - Output: Success or failure of the transaction

3. Fund : $(\mathsf{specId}, [(\mathsf{pk}, \mathsf{cm}^{\mathsf{recv}}, \mathsf{Enc}_{\mathsf{pk}}(\mathsf{b}, \omega))], \mathsf{v}_{\mathsf{pub}}, [\mathsf{cm}^{\mathsf{send}}], \mathsf{cm}^{\mathsf{send}}_{\mathsf{final}}, \pi_{\mathsf{Fund}}) \to \{0,1\}$

   This function allows users to convert public funds to private funds, or vice versa, or a make a private transfer. If the smart contract is able to validate the inputs against the state of the sender's wallet and verify the zero-knowledge proof, then it will update the sender's wallet, replacing the commitments $[\mathsf{cm}^{\mathsf{send}}]$ with the commitment $\mathsf{cm}^{\mathsf{send}}_{\mathsf{final}}$, and it will append the transfer commitments cm^{recv} to each of the recipient's respective accounts.

   - Inputs:
     - specId: Public key for a ZUZ specification
     - $[\mathsf{pk}, \mathsf{cm}^{\mathsf{recv}}, \mathsf{Enc}_{\mathsf{pk}}(\mathsf{b}, \omega)]$: A list of recipients and the respective values to transfer to each of them, including
       - pk: The public key of the recipient
       - $\mathsf{cm}^{\mathsf{recv}}$: A committed value of the amount to transfer the given recipient
       - $\mathsf{Enc}_{\mathsf{pk}}(\mathsf{b}, \omega)$: A ciphertext which encrypts the commitment openings under the public key of the recipient
     - $\mathsf{v}_{\mathsf{pub}}$: The numerical amount representing the change in the sender's public balance for specId
     - $[\mathsf{cm}^{\mathsf{send}}]$: A list of the sender's wallet commitments which correspond to the sender's private balance for specId
     - $\mathsf{cm}^{\mathsf{send}}_{\mathsf{final}}$: The sender's update private wallet commitment, appended to specId after the transaction
     - π_{Fund}: ZKP that demonstrates the correctness of the transaction
   - Output: Success or failure of the transaction

4.3 Implementation

We show how the above interface for the Fund transaction can be used in the following abstract implementation [1]. Specifically, we show how to use this interface to implement conversions between public and private ZUZ instances as well as make destination- and value-anonymous transfers.

1. $\mathsf{ShieldTx}(\mathsf{specId}, \mathsf{v_{pub}}, \mathsf{cm}^{\mathsf{send}}_{\mathsf{final}}, \pi_{\mathsf{SHIELD}}) \rightarrow \{0,1\}$

   We implement the transaction for converting public to private funds given the following parameters:

   - Inputs:
     - specId: Public key for a ZUZ specification
     - $\mathsf{v_{pub}}$: The numerical amount representing the change in the sender's public balance for specId
     - $\mathsf{cm}^{\mathsf{send}}_{\mathsf{final}}$: The sender's update private wallet commitment, appended to specId after the transaction
     - π_{Shield}: ZKP that demonstrates the correctness of the conversion
   - Output: Success or failure of the transaction

   In particular, the statement π_{Shield} asserts that the new commitment $\mathsf{cm}^{\mathsf{send}}_{\mathsf{final}}$ is opened by the tuple $(\mathsf{b_{final}}, \omega_{\mathsf{final}})$ such that the sender's balance is preserved, or $0 \leq \mathsf{b_{final}} \leq |\mathsf{v_{pub}}|$.

   The smart contract validates the inputs to ShieldTx by checking that the updated value of the sender's public balance on the smart contract $\mathsf{b}^{\mathsf{old}}_{\mathsf{pub}} + \mathsf{v_{pub}}$ is non-negative. The smart contract also verifies π_{Shield}. If both of these checks pass, then the value of the sender's public balance for spec is updated to $\mathsf{b}^{\mathsf{old}}_{\mathsf{pub}} + \mathsf{v_{pub}}$ and the commitment $\mathsf{cm}^{\mathsf{send}}_{\mathsf{final}}$ is added to the sender's private balance for specId.

2. $\mathsf{RevealTx}(\mathsf{specId}, \mathsf{v_{pub}}, [\mathsf{cm}^{\mathsf{send}}], \mathsf{cm}^{\mathsf{send}}_{\mathsf{final}}, \pi_{\mathsf{Shield}}) \rightarrow \{0,1\}$

   We implement the transaction for converting private to public funds given the following parameters:

   - Inputs:
     - specId: Public key for a ZUZ specification
     - $\mathsf{v_{pub}}$: The numerical amount representing the change in the sender's public balance for specId
     - $[\mathsf{cm}^{\mathsf{send}}]$: A list of the sender's wallet commitments which correspond to the sender's private balance for specId
     - $\mathsf{cm}^{\mathsf{send}}_{\mathsf{final}}$: The sender's update private wallet commitment, appended to specId after the transaction
     - π_{Reveal}: ZKP that demonstrates the correctness of the conversion
   - Output: Success or failure of the transaction

   In particular, the statement π_{REVEAL} asserts that the new commitment $\mathsf{cm}^{\mathsf{send}}_{\mathsf{final}}$ is opened by the tuple $(\mathsf{b_{final}}, \omega_{\mathsf{final}})$ such that the sender's balance is preserved, or $\mathsf{b_{final}} \geq 0$ and $\mathsf{b_{final}} + \mathsf{b}^{\mathsf{new}}_{\mathsf{pub}} = \mathsf{b}^{\mathsf{old}}_{\mathsf{pub}} + \Sigma_j \mathsf{b}^{\mathsf{send}}_{\mathsf{j}}$ where $\mathsf{b}^{\mathsf{send}}_{\mathsf{j}} \geq 0$ and $(\mathsf{b}^{\mathsf{send}}_{\mathsf{j}}, \omega^{\mathsf{send}}_{\mathsf{j}})$ open the sender's wallet

commitments $\mathsf{cm}_j^{\mathsf{send}}$, $\forall j$.

The smart contract validates the inputs to RevealTx by checking that

- the updated value of the sender's public balance on the smart contract $\mathsf{b}_{\mathsf{pub}}^{\mathsf{old}} + \mathsf{v}_{\mathsf{pub}}$ is non-negative
- the list of the sender's wallet commitments $\{\mathsf{cm}_j^{\mathsf{send}}\}_{\forall j}$ is a subset of the state of the sender's wallet.

Note that the last condition suffices since we enforce that each wallet commitment shields a non-negative value. Furthermore, this makes each operation resilient to front-running attacks, since concurrent transactions can only increase the user's balance.

The smart contract also verifies π_{REVEAL}. If both of these checks pass, then the value of the sender's public balance for spec is updated to $\mathsf{b}_{\mathsf{pub}}^{\mathsf{old}} + \mathsf{v}_{\mathsf{pub}}$ and the sender's wallet commitments $\{\mathsf{cm}_j^{\mathsf{send}}\}_{\forall j}$ are removed and the new commitment $\mathsf{cm}_{\mathsf{final}}^{\mathsf{send}}$ is added to the sender's private balance for spec.

3. $\mathsf{TransferTx}(\mathsf{specId}, [(\mathsf{pk}, \mathsf{cm}^{\mathsf{recv}}, \mathsf{Enc}_{\mathsf{pk}}(\mathsf{b}, \omega))], \mathsf{v}_{\mathsf{pub}}, [\mathsf{cm}^{\mathsf{send}}], \mathsf{cm}_{\mathsf{final}}^{\mathsf{send}}, \pi_{\mathsf{Transfer}}) \rightarrow \{0,1\}$

   We implement the transaction for transferring a confidential amount from a publicly known sender to an anonymous recipient in a ring, given the following parameters

   - Inputs:
     - specId: Public key for a ZUZ specification
     - $[\mathsf{pk}, \mathsf{cm}^{\mathsf{recv}}, \mathsf{Enc}_{\mathsf{pk}}(\mathsf{b}, \omega)]$: A list of recipients and the respective values to transfer to each of them, including
       - pk: The public key of the recipient
       - $\mathsf{cm}^{\mathsf{recv}}$: A committed value of the amount to transfer the given recipient
       - $\mathsf{Enc}_{\mathsf{pk}}(\mathsf{b}, \omega)$: A ciphertext which encrypts the commitment openings under the public key of the recipient
     - $\mathsf{v}_{\mathsf{pub}}$: The numerical amount representing the change in the sender's public balance for specId
     - $[\mathsf{cm}^{\mathsf{send}}]$: A list of the sender's wallet commitments which correspond to the sender's private balance for specId
     - $\mathsf{cm}_{\mathsf{final}}^{\mathsf{send}}$: The sender's update private wallet commitment, appended to specId after the transaction
     - π_{Fund}: ZKP that demonstrates the correctness of the transaction
   - Output: Success or failure of the transaction

   In particular, the statement π_{TRANSFER} asserts the sender has enough funds to complete this transfer, or that $\mathsf{b}_{\mathsf{pub}}^{\mathsf{old}} + \Sigma_j \mathsf{b}_j^{\mathsf{send}} = \mathsf{b}_{\mathsf{pub}}^{\mathsf{new}} + \mathsf{b}_{\mathsf{final}}^{\mathsf{send}} + \Sigma_i \mathsf{b}_i^{\mathsf{recv}}$ where

   - $\mathsf{b}_i^{\mathsf{recv}} \geq 0$, $(\mathsf{b}_i^{\mathsf{recv}}, \omega_i^{\mathsf{recv}})$ opens $\mathsf{cm}_i^{\mathsf{recv}}$, $\forall i$,
   - $\mathsf{b}_j^{\mathsf{send}} \geq 0$, $(\mathsf{b}_j^{\mathsf{send}}, \omega_j^{\mathsf{send}})$ opens $\mathsf{cm}_j^{\mathsf{send}}$, $\forall j$, and
   - $\mathsf{b}_{\mathsf{final}}^{\mathsf{send}} \geq 0$, $(\mathsf{b}_{\mathsf{final}}^{\mathsf{send}}, \omega_{\mathsf{final}}^{\mathsf{send}})$ opens $\mathsf{cm}_{\mathsf{final}}^{\mathsf{send}}$

   The smart contract validates the inputs to TransferTx by checking that

   - the updated value of the sender's public balance on the smart contract $\mathsf{b}_{\mathsf{pub}}^{\mathsf{old}} + \mathsf{v}_{\mathsf{pub}}$ is

ZKP circuit	Parameters	Time Complexity
Transfer (one-step)	w sender balances r recipient balances	$O(r+w)$

Table 4.1: Time complexity of destination- and value-anonymous transfer

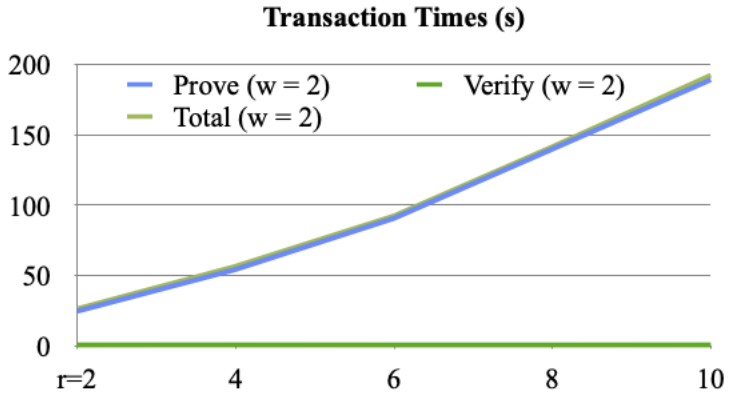

Figure 4.1: Destination- and value-anonymous transfer times

non-negative

- the list of the sender's wallet commitments $\{\mathsf{cm}_j^{\mathsf{send}}\}_{\forall j}$ is a subset of the state of the sender's wallet.

The smart contract also verifies π_{Transfer}. If both of these checks pass, then the value of the sender's public balance for spec is updated to $\mathsf{b}_{\mathsf{pub}}^{\mathsf{old}} + \mathsf{v}_{\mathsf{pub}}$ and the sender's wallet commitments $\{\mathsf{cm}_j^{\mathsf{send}}\}_{\forall j}$ are removed and the new commitment $\mathsf{cm}_{\mathsf{final}}^{\mathsf{send}}$ is added to the sender's private balance for spec. Furthermore, each commitment $\mathsf{cm}_i^{\mathsf{recv}}$ is added to the wallet of the recipient with the public key pk_i.

4.4 Preliminary Results

Our preliminary implementation of the above privacy-preserving ZUZ protocol in the account-based model is written in Solidity as a collection of Ethereum smart contracts. In our implementation, we use the ZoKrates library to model client-side offline proof generation and to generate smart contracts for proof verification. We encode the circuits $\pi_{\mathsf{SHIELD}}, \pi_{\mathsf{REVEAL}}$ and π_{TRANSFER} in ZoKrates' high-level domain-specific language and use their JavaScript bindings to write a client-side module for offline proof generation and transaction creation.

We note that this is a proof-of-concept implementation and that these results are only preliminary. As such, some details have been omitted from the implementation, such as the generation of random ω_i in the commitment schemes. Furthermore, when implementing the circuits using this library, we ran into limitations with the sizes of the circuits that could be generated. In future iterations of our design, we intend to use a lower-level zero-knowledge proof system and conduct more extensive testing.

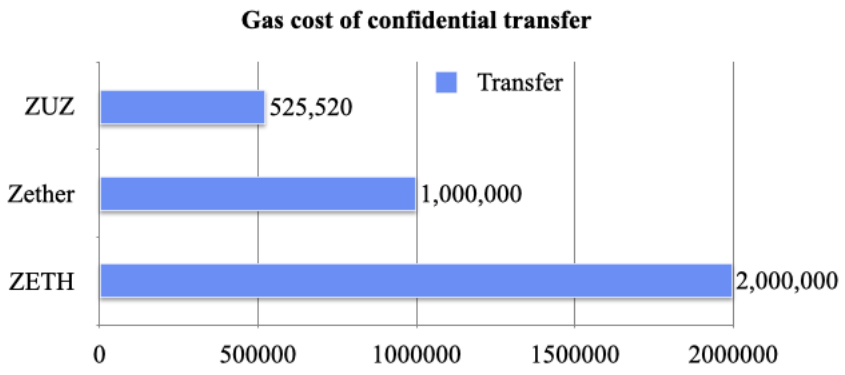

Figure 4.2: Comparison of gas costs for confidential transfer across various smart contracts

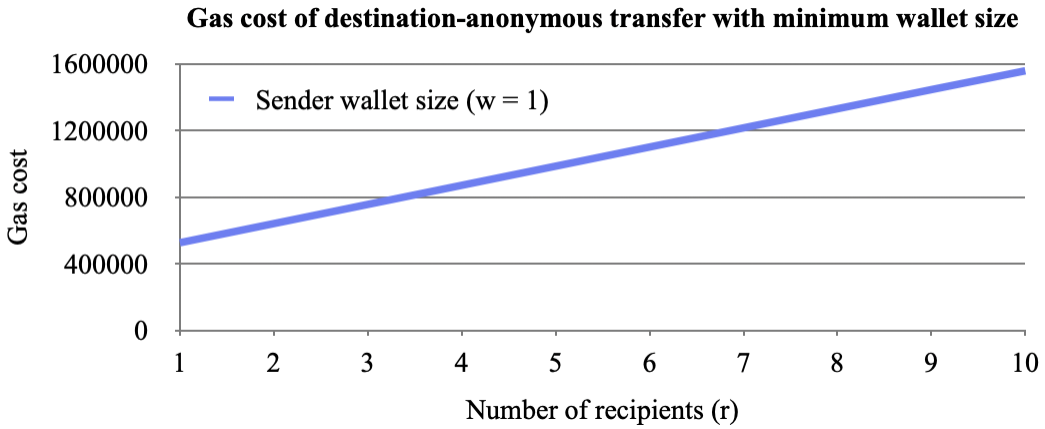

Figure 4.3: Gas costs for a fixed wallet size as the destination anonymity of a transfer increases

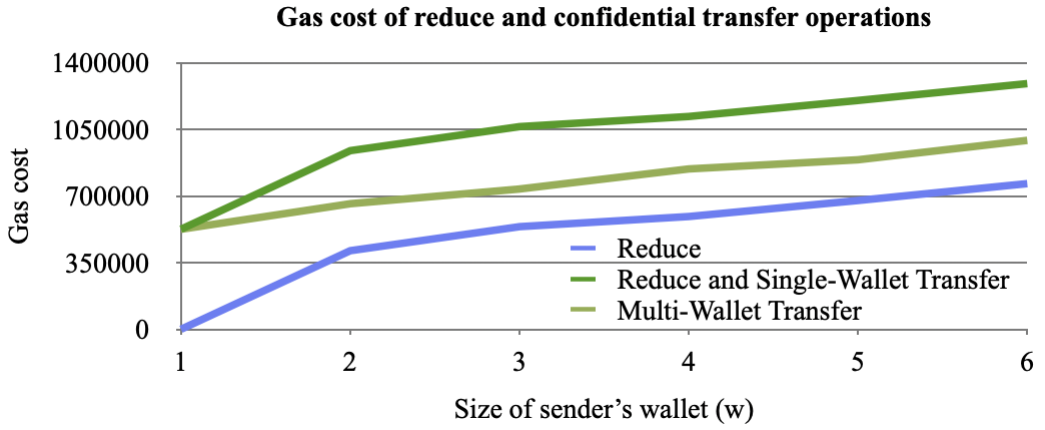

Figure 4.4: Gas costs for confidential transfer as the size of a sender's wallet increases

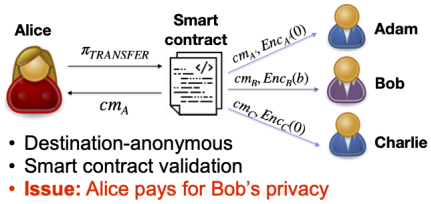

Figure 4.5: One-step transfer with destination-and value-anonymity

As shown in Figure 4.1, the transaction time for a single-step destination-and value-anonymous transfer scales linearly in the size of the anonymity set, or the number of recipients specified in the transfer. The graph measures the transfer time where the sender's wallet is fixed at a size of $w = 2$ commitments, and the number of recipients specified in the transfer scales from $r = 2$ to $r = 10$. Transaction times are dominated by the offline computation, specifically proof generation, where online computation, or proof verification by the smart contract, remain constant.

Furthermore, in Figure 4.2, we see that gas costs for confidential transfers are significantly lower than comparable privacy-preserving smart contract payment schemes [11], due to the fact that client-side computation outweighs smart contract computation in our scheme. In Figure 4.3, we measure how the gas cost of a transfer scales as the privacy of the transaction increases when the sender's wallet is the minimum size. We demonstrate that gas costs for the destination anonymity of a ZUZ transfer scales linearly in the size of the anonymity ring when the sender has a single wallet commitment. In fact, a ZUZ transfer can hide the recipient of a transfer among up to 5 users for a lower gas cost than a single confidential Zether transfer.

Our attempts to optimize for the gas cost of a transfer by considering the size of the sender's wallet yielded surprising results. As shown in Figure 4.4, the operations of transferring funds from a multi-sized wallet and reducing the size of a sender's wallet scale proportionally. This means that users that "refresh" their wallets, by performing the reduce operation independently, do not make gain any performance benefits.

4.5 Privacy Guarantees

4.5.1 Destination-and Value-Anonymous Transfer

First, we informally argue about the privacy guarantees of the existing design. Our destination-and value-anonymous transfer as shown in figure 4.5 achieves *plausible deniability* since the intended recipient of the transfer is unknown.

Intuitively, our design supports k-anonymity in its destination anonymity, where k is the number of recipients specified in any given transfer. We share a similar design to Anonymous Zether, where anonymity is limited to a ring of k specified recipients [14]. However, our designs make different trade-offs. Whereas Anonymous Zether supports a single k-anonymous transfer per epoch over origin, destination, and value, our design can support an arbitrary number of destination-and value-anonymous transfers per epoch.

Our design upholds the same property that forms the basis of the privacy in the smart con-

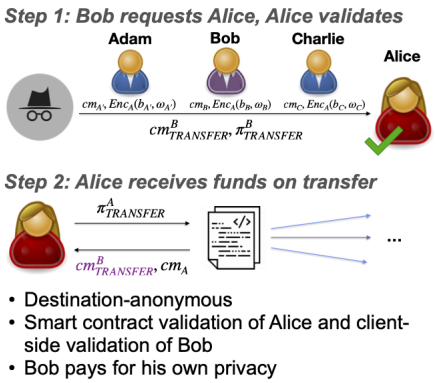

Figure 4.6: Two-step transfer with “pay for privacy"

tract ZETH, which guarantees destination-and value-anonymity transfers over arbitrary denominations using a mixer contract [25]. ZETH relies on the fact that the individual coins generated by each transaction cannot be traced. Similarly, in our design, users can combine the wallet commitments representing arbitrary denominations from different transfers, thereby obfuscating the flow of a specific commitment between sender and recipient.

The main difference in the privacy analysis between ZETH’s design and ours is that the sender of a ZETH transaction is hidden among the set of all previous users of the mixer contract, whereas in our scheme, the sender is hidden among the set of all previous users that have privately sent funds to the recipient.

4.5.2 “Pay for Privacy"

Under the model presented thus far, there arises a discrepancy in incentives when it comes to paying for privacy. In particular, the sender of a transaction must fund the gas cost for private transfer, where the cost increases proportional to the size of the anonymity ring. Thus, the sender pays for the level of privacy that the recipient receives.

We address the mismatch in incentives in a destination- and value-anonymous transfer by presenting a "pay for privacy" scheme as shown in figure 4.6, which extends the transfer protocol to allow a recipient to anonymously put in a request for a private transfer with some compensation to cover its cost.

Note that compensation must be in the form of a ZUZ instance, and not directly the ETH used to pay for gas, since that would link two Ethereum accounts interacting with this smart contract. Furthermore, we assume that each anonymous transaction is initiated from a distinct Ethereum account, which has no prior spending history with our smart contract.

Suppose that Bob wants to request Alice for funds. Then, he sends the transaction specifications to Alice, including the requested ZUZ specification, the list of recipient addresses (including Bob’s address), list of commitments corresponding to the amount to privately transfer to each of the recipients, and list of parameters that unlock each of the transfer commitments encrypted with Alice’s public key. Note that the sender of the transaction, Alice, is the only person who will be able to tell who requested funds since she can verify that Bob’s balance is nonzero in the

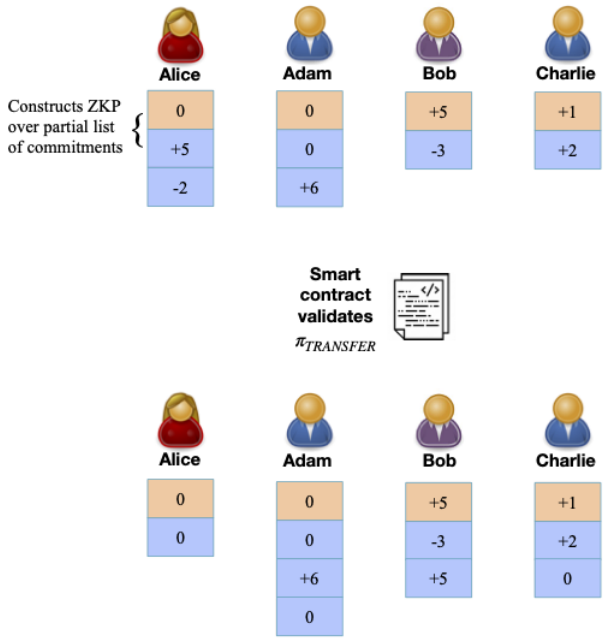

Figure 4.7: Wallet representations in a double-spending attack

above list of encrypted parameters.

Bob can also provide some compensation to Alice along with this specification by using the one-step transfer as a primitive. The only difference is that the zero-knowledge proof will have an additional constraint to prove knowledge of Bob's private key, as he is signing the transaction request from a different account. Note that Alice is the only entity that can efficiently validate this zero-knowledge proof, and thereby the transaction request, since she is the only party, including the smart contract, that can distinguish who the requester of the transaction is.

If Alice successfully validates the transaction, she can complete the request by providing Bob's transfer specification to the smart contract and following the one-step transfer protocol. Upon completing the transfer specification, the smart contract will transfer the associated compensation to Alice.

Ultimately, the key insight behind this design is to use client-side validation to efficiently hide the identity of the requester of a transaction, in order to mitigate the issues presented in the prior section with smart contract validation. Note, however, that Alice, or the sender of the transfer, is still a publicly known parameter.

4.5.3 Origin-Anonymous Transfer

Our first attempt to make an origin-anonymous transfer is to include the sender in the list of recipients of the transaction. Note that in order for the sender to remain anonymous in this ring of recipients, they must send this transaction from an account that is distinct from any of the recipients in the transaction. The sender transfers themselves a negative private balance such that the sum over all the transfer balances, positive and negative, equals zero.

The sender must also prove that they have sufficient funds to make this transfer. More specifically, the sender must prove that the sum over all their private balances, including the negative transfer balance, is nonnegative. However, in order to preserve anonymity of the origin under this model, the zero-knowledge proof doesn't just take in the sender's wallet commitments as input but rather all the recipients' wallet commitments.

As stated in the specification, the smart contract must validate that the inputs to the zero-knowledge proof are consistent with the ledger state in order to prevent double-spending attacks. As shown in Figure 4.7, if Alice has two private balances in her wallet, which correspond to the values $+5$ and -2, but she proves in zero-knowledge that she has sufficient funds only over the first commitment, she may be able to spend funds that she does not have. Thus, the smart contract must validate that all the sender's wallet commmitments used as input to the zero-knowledge proof are consistent with the ledger state.

In our origin-anonymous protocol, the zero-knowledge proof takes the wallet commitments of all the members of the ring as input. By definition of an origin-anonymous transfer, the smart contract should not be able to distinguish between the sender's wallet commitments and the recipients' wallet commitments. Thus, the smart contract needs to validate that *all* wallet commitments of both the sender and the recipient are consistent with the state of the ledger when it processes the transaction.

This presents a problem since recipients may be involved in other transfers at the same time, which means that the state of their wallet commitments could change between the time the sender constructs the zero-knowledge proof and the time that the transaction is processed.

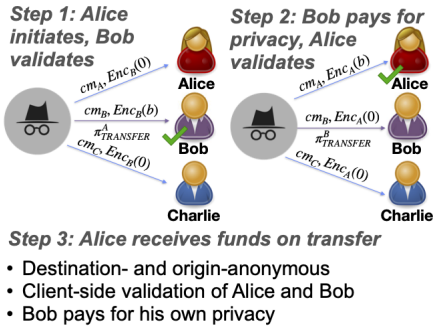

Figure 4.8: Three-step transfer with origin anonymity

Instead, building off of the prior scheme, we can use client-side validation by both parties in order to conceal the origin on the transfer, as shown in Figure 4.8.

First, Alice sends a transfer specification, similar to a one-step transfer, except that the list of parameters are all encrypted with Bob's public key. Thus, only Bob knows the identity of the sender of the transaction, which means that he is the only party who can efficiently validate Alice's transfer.

If Bob successfully validates the transaction, he can provide compensation for the transfer, similar to what he sends in the first round of the "pay for privacy" scheme. Once again, the list of parameters are encrypted with Alice's public key, which menas that she is the only party who can efficiently validate Bob's transfer.

Finally, if Alice successfully validates Bob's transfer, Alice can accept the transaction, at which point all of the transfer commitments are appended to all the involved members' wallets. If Alice or Bob reject the transfer at any step, all pending state in the smart contract is rolled back.

4.6 Summary

The protocols summarized in Figures 4.5, 4.6, and 4.8 offer increasing levels of privacy at the cost of extended rounds of computation. The one-step transfer primitive offers destination- and value-anonymity over arbitrary denominations with smart contract validation of the sender's funds. In order to have stronger privacy guarantees, the recipient can anonymously specify a ring of users in a transfer specification, along with compensation for the user who will complete the transfer request. The recipient will have their compensation validated by the requested sender, and the sender will have their completion of the transfer validated by the smart contract.

In this way, the recipient is able to pay for their privacy, providing the sender with an explicit transfer specification and some compensation to cover the cost of a private transfer. Nevertheless, the sender is still a public parameter throughout the two-step transfer process. By extending the protocol for a third round and using client-side validation for both the sender and recipient of the transfer, we can achieve origin-anonymity as well as a "pay for privacy" guarantee.

Note that a caveat of these protocols is that the gas cost of transfers are funded by Ethereum accounts, which can be linked to their private smart contract identities over repeated interaction with the smart contract. This means that the "pay for privacy" issue cannot be directly resolved by exchanging ETH to cover gas fees. Instead, compensation must be in the form of an anonymous ZUZ payment. In order to achieve the privacy guarantees we described in our design, we assumed that each anonymous transaction is initiated from a distinct Ethereum account, which has no prior spending history with our smart contract.

Chapter 5

Reputation

5.1 Background and Motivation

In the previous sections, we explored two different constructions for a privacy-preserving ZUZ ledger, in both the UTXO- and account-based models. Recall that in the decentralized token system ZUZ, individuals can define their own token specifications. Implicitly, when individuals create their own token specification on the ZUZ system, they are staking their own real-world reputation, since only the individuals who trust the owner of the token specification in real life will be willing to trade using the individual's ZUZ.

However, over time, users that do not know the owner's real-world reputation may be convinced to accept currency from a particular individual's token specification by reviewing the publicly observable interactions over this specification. For instance, a user may begin to accept a token specification without knowledge of the owner's real-world reputation simply because many of the user's peers accept trades made with this token specification.

Thus, the extent to which a user is able to generate capital on the ZUZ system depends on the ability of each user to convince others of their token's reputation. Users are implicitly able to evaluate the reputation of any token on the network when the system built upon a transparent and universally accessible ledger. As a result, the ability of the ZUZ system to build the reputation of token specifications comes into direct conflict with its ability to preserve the privacy of user transactions.

In this section, we begin to address this conflict beteween privacy and reputation by formalizing *statements of blockchain reputation* for application-specific reputation scoring functions with respect to a public ledger. We introduce this notion with the goal of eventually constructing statements which can be used to measure the reputation of token specifications in the privacy-preserving ledgers described earlier.

The remaining sections demonstrate how we can efficiently prove and verify a user's reputation with respect to their past transactions on a public ledger. A naive protocol to compute a "reputation score" over a user's transaction history requires an individual to compute over the entire blockchain. In practice, it is infeasible for users to compute this value before interacting with other parties.

Our protocol uses *incrementally verifiable computation* to allow users to efficiently prove

and verify statements about their reputation. In particular, incrementally verifiable computation allows users to efficiently update their proof of reputation as they perform new transactions. Regardless of the number of updates that the user performs, the resulting proof using this technique remains constant size, resulting in efficient verification times.

Without the ability to efficiently evaluate the individual transacting with them, smart contracts and network users are inherently limited in their capability for peer-to-peer interaction. As such, we also explore a variety of applications of this novel primitive in this section.

5.2 Preliminaries

Definition 1 (Merkle Trees)**.** *A merkle tree* MT *is defined by the following interface:*

- $\mathsf{MT.insert}(\mathsf{rt}, \mathsf{e}) \rightarrow \mathsf{rt}'$*: Inserts element* e *as a leaf in the Merkle tree rooted at* rt *and returns the updated root* rt'
- $\mathsf{MT.verify}(\mathsf{rt}, \pi, \mathsf{e}) \rightarrow \{0,1\}$*: Verifies that the element* e *is a leaf in the Merkle tree rooted at* rt *using the sibling hashes along the path in the tree from the element* e *to* rt

Definition 2 (Blockchain)**.** *A blockchain* BL *is defined as a list of n blocks* $\mathsf{B}_1, ..., \mathsf{B}_n$ *such that* $\forall i \in [n]$

$$\mathsf{B}_i = (\mathsf{h}_{\mathsf{parent}}, \mathsf{h}_{\mathsf{tx}}, e)$$

A block contains the hash of its parent block $\mathsf{h}_{\mathsf{parent}}$*, the Merkle tree root of all transactions included in that block* h_{tx}*, and an epoch number e.*

Definition 3 (Incrementally Verifiable Computation)**.** *An incrementally verifiable computation (IVC) scheme is defined by PPT algorithms* $(\mathscr{G}, \mathscr{P}, \mathscr{V})$ *and deterministic* $\mathscr{K}$*, denoting the generator, the prover, the verifier, and the encoder respectively, with the following interface*

- $\mathscr{G}(1^\lambda) \rightarrow \mathsf{pp}$*: On input security parameter* λ*, samples public parameters* pp*.*
- $\mathscr{K}(\mathsf{pp}, F) \rightarrow (\mathsf{pk}, \mathsf{vk})$*: On input public parameters* pp*, and polynomial-time function F, deterministically produces a prover key* pk *and a verifier key* vk*.*
- $\mathscr{P}(\mathsf{pk}, (i, z_0, z_i), \omega_i, \Pi_i) \rightarrow \Pi_{i+1}$*: On input a prover key* pk*, a counter i, an initial input* z_0*, a claimed output after i iterations* z_i*, a non-deterministic advice* ω_i*, and an IVC proof* Π_i *attesting to* z_i*, produces a new proof* Π_{i+1} *attesting to* $z_{i+1} = F(z_i, \omega_i)$*.*
- $\mathscr{V}(\mathsf{vk}, (i, z_0, z_i), \Pi_i) \rightarrow \{0,1\}$*: On input a verifier* vk*, a counter i, an initial input* z_0*, a claimed output after i iterations* z_i*, and an IVC proof* Π_i *attesting to* z_i*, outputs 1 if* Π_i *is accepting, and 0 otherwise.*

An IVC scheme $(\mathscr{G}, \mathscr{K}, \mathscr{P}, \mathscr{V})$ *satisfies the following requirements.*

1. *Perfect Completeness: For any PPT adversary* $\mathscr{A}$

$$\Pr\left[\mathscr{V}(\mathsf{vk}, i, z_0, z_{i+1}, \Pi_{i+1}) = 1 \;\middle|\; \begin{array}{l} \mathsf{pp} \leftarrow \mathscr{G}(1^\lambda), \\ F, (i, z_0, z_i, \Pi_i), \leftarrow \mathscr{A}(\mathsf{pp}), \\ (\mathsf{pk}, \mathsf{vk}) \leftarrow \mathscr{K}(\mathsf{pp}, F), \\ z_{i+1} = F(z_i, \omega_i), \\ \mathscr{V}(\mathsf{vk}, i, z_0, z_i, \Pi_i) = 1, \\ \Pi_{i+1} \leftarrow \mathscr{P}(\mathsf{pk}, (i, z_0, z_i), \omega_i, \Pi_i) \end{array}\right] = 1$$

where F is a polynomial time computable function.

2. *Knowledge Soundness: Consider constant* $n \in \mathbb{N}$. *For all expected polynomial-time adversaries* $\mathscr{P}^*$ *there exists an expected polynomial-time extractor* $\mathscr{E}$ *such that over all randomness* r

$$\Pr\left[\begin{array}{l|l} & \mathsf{pp} \leftarrow \mathscr{G}(1^\lambda), \\ z_n = z \text{ where} & F, (z_0, z_i, \Pi) \leftarrow \mathscr{P}^*(\mathsf{pp}, \mathsf{r}), \\ z_{i+1} \leftarrow F(z_i, \omega_i) & (\mathsf{pk}, \mathsf{vk}) \leftarrow \mathscr{K}(\mathsf{pp}, F), \\ \forall i \in \{0, \ldots, n-1\} & \mathscr{V}(\mathsf{vk}, (n, z_0, z), \Pi) = 1, \\ & (\omega_0, \ldots, \omega_{n-1}) \leftarrow \mathscr{E}(\mathsf{pp}, \mathsf{r}) \end{array}\right] \approx 1.$$

3. *Succinctness: The size of an IVC proof* Π *is independent of the number of iterations* n.

Definition 4 (Relation for blockchain reputation)**.** *Let the relation* $\mathscr{R}_{BR}$ *consist of tuples of structure, instance, and witness defined as follows*

- *Input structure is of the form* $\mathsf{s} = f$*, where* f *is an application-specific reputation scoring function*
- *Instances are of the form* $x = \{\mathsf{h}_{\mathsf{B}_0}, \mathsf{h}_{\mathsf{B}_n}, \mathsf{addr}, r\}$. *An instance* x *specifies a reputation score* r *for the public key* addr *with respect to the blockchain starting from block* B_0 *and ending with block* B_n*, denoted by the hashes* $\mathsf{h}_{\mathsf{B}_0}$ *and* $\mathsf{h}_{\mathsf{B}_n}$*, respectively.*
- *Witnesses are of the form* $w = \{(\mathsf{B}_i)_{i=1}^n, (\mathsf{tx}_j, \pi_j, k_j)_{j=1}^m, \mathsf{aux}_F\}$ *where the transactions in each block are concisely represented by the Merkle tree root hash, or* $\mathsf{B}_i = (\mathsf{h}_{\mathsf{parent}}, \mathsf{h}_{\mathsf{tx}}, e)$. *The witness* w *includes a list of up to a large constant* m *transactions* $\mathsf{tx}_1, \ldots, \mathsf{tx}_m$ *such that* $\forall j \in [m]$

$$\mathsf{tx}_j = (\mathsf{pk}_{\mathsf{send}}, \mathsf{pk}_{\mathsf{recv}}, \mathsf{amt})$$

The prover also provides a list of the corresponding proofs of inclusion π_j *for transaction* tx_j. *Specifically,* π_j *is the path to the leaf* tx_j *in the Merkle tree rooted at* h_{tx} *for the block with index* $k_j \in [n]$*, or* B_{k_j}. *We denote any application-specific witnesses for the reputation scoring function* F *with* aux_F.

Suppose the default reputation value r_0 *is hard-coded. Given an instance* x*, a witness* w *is valid for* x *under* $\mathscr{R}_{BR}$ *if Algorithm 1 holds.*

5.3 Construction

Construction 1 (IVC scheme for monotone blockchain reputation)**.** *We instantiate an incrementally verifiable monotone blockchain reputation scheme by constructing the function* F *described in Algorithm 2 with respect to a monotone application-specific reputation scoring function* f_{rep}.

We additionally require that the verifier has access to the blockchain and can verify that h_{B_0} *and* h_{B_n} *are indeed valid block hashes that denote an interval of* n *blocks.*

We claim that a statement of monotone blockchain reputation proved under the IVC scheme under the function F for a monotone application-specific reputation function f_{rep} is functionally equivalent to one that holds under the relation for blockchain reputation presented in Algorithm 1 by the correctness of the IVC scheme.

This construction allows users to begin computing their reputation with respect to some user-defined starting block, B_0, and some application-defined value for reputation, r_0. If desired, the

Algorithm 1: Relation for blockchain reputation

Data: $f_{rep}, \{\mathsf{h}_{\mathsf{B}_0}, \mathsf{h}_{\mathsf{B}_n}, \mathsf{addr}, r\}, \{(\mathsf{B}_i)_{i=1}^{n}, (\mathsf{tx}_j, \pi_j, k_j)_{j=1}^{m}, \mathsf{aux}_F\}$
Result: validBlocks and validRep
begin
  for $i \in \{1,...,n\}$ **do**
    validBlocks $\leftarrow \mathsf{B}_{i+1}.\mathsf{h}_{\mathsf{parent}} = \mathsf{h}_{\mathsf{B}_i}$ and validBlocks
  end
  for $j \in \{1,...,m\}$ **do**
    validTxn $\leftarrow \mathsf{MT.verify}(\mathsf{tx}_j, \pi_j, \mathsf{B}_{k_j}.\mathsf{h}_{\mathsf{tx}})$
    if *validTxn* **then**
      $r_{j+1} \leftarrow f_{rep}(r_j, \mathsf{tx}_j, \mathsf{aux}_F)$
    end
  end
  validRep $\leftarrow r_m = r$;
end

user can obtain the signature of a trusted authority for the particular application that describes the user's reputation between the genesis block at B_0.

The IVC step function F allows a user to update their existing proof for their reputation from block B_i to block B_{i+1}. The user has the ability to include up m transactions from the block as private witnesses, which will be used to update their reputation. This parameter poses an upper limit on the amount a user's reputation can increase per block.

This protocol ensures the validity of the provided transactions by requiring the user to update their proof contiguously for each block starting from B_0. Furthermore, we requires that the hash of the starting block $\mathsf{h}_{\mathsf{B}_0}$ and end block $\mathsf{h}_{\mathsf{B}_n}$, as included in the statement of reputation, correspond to $\mathsf{h}(\mathsf{B}_0.\mathsf{e}))$ and $\mathsf{h}(\mathsf{B}_n.\mathsf{e}))$, respectively, in the state of the blockchain.

Algorithm 2: Application-specific IVC function

Data: $\{\mathsf{h}_{\mathsf{B}_0}, \mathsf{h}_{\mathsf{B}_i}, \mathsf{addr}, r\}, \{\mathsf{B}_{i+1}, (\mathsf{tx}_j, \pi_j)_{j=1}^{m}, \mathsf{aux}_f\}$
Result: $(\mathsf{h}_{\mathsf{B}_0}, H(\mathsf{B}_{i+1}), \mathsf{addr}, r')$
begin
  validBlock $\leftarrow \mathsf{B}_{i+1}.\mathsf{h}_{\mathsf{parent}} = \mathsf{h}_{\mathsf{B}_i}$ **for** $j \in \{1,...,m\}$ **do**
    validTxn $\leftarrow$ *MT.verify*$(\mathsf{tx}_j, \pi_j, \mathsf{B}_{i+1}.\mathsf{h}_{\mathsf{tx}})$
    if validBlock *and* validTxn **then**
      $r' \leftarrow f_{rep}(r, \mathsf{tx}, \mathsf{aux}_f)$
    end
  end
  return $(\mathsf{h}_{\mathsf{B}_0}, H(\mathsf{B}_{i+1}), \mathsf{addr}, r')$
end

5.4 Applications

5.4.1 Monotonic Functions

Monotonic reputation functions are useful for capturing the quantity, rather than the quality, of user interactions. They can be used to capture user activity, or the popularity of an item, for instance, by tracking the frequency of usage. This property makes them useful for measuring the worth of different tokens on ZUZ and Ethereum.

ZUZ Specifications

We list a few functions that can be used to measure the reputation of a ZUZ specification, out of many possible metrics, below. The key insight is that by including additional transactions made using a given ZUZ specification, the reputation of the specification according to each of these metrics cannot decrease, since they measure the frequency of token usage.

Metric 1 (Number of transactions traded over this spec): Let $r_0 = 0$ and $\mathsf{aux}_f = \mathsf{specId}$. We define the reputation function f_{rep} in Algorithm 3. A transaction contributes to the total number of trades if it was made using coins of the given specId.

Algorithm 3: f_{rep} for Metric 1

```
Data: r, tx, specId
Result: r'
begin
    r' ← r + (tx.specId = specId)
    return r'
end
```

Metric 2 (Amount of money traded over this spec): Let $r_0 = 0$ and $\mathsf{aux}_f = \mathsf{specId}$. We define the reputation function f_{rep} in Algorithm 4. If a transaction was made using coins of the given specId, its contribution to the total reputation is scaled by the amount transferred.

Algorithm 4: f_{rep} for Metric 2

```
Data: r, tx, specId
Result: r'
begin
    r' ← r + tx.amt * (tx.specId = specId)
    return r'
end
```

Metric 3 (Number of users who own instances of this spec): Let $r_0 = \mathsf{rt}$, or the root of a Merkle tree where each leaf representing the balance of a different user is initialized to 0, and $\mathsf{aux}_f = \mathsf{specId}$. We define the reputation function f_{rep} in Algorithm 5. If a transaction was made using coins of the given specId, the account balances corresponding to the sender and receiver of the

transaction are updated according to the transaction amount in the Merkle tree. The number of leaves with positive balances represents the number of users who own instances of the given ZUZ specification.

Algorithm 5: f_{rep} for Metric 3

Data: $r, \mathsf{tx}, \mathsf{specId}$
Result: $\mathsf{rt}_{\mathsf{recv}}$
begin
  $\mathsf{amt}_{\mathsf{send}} \leftarrow \mathsf{MT.get}(\mathsf{tx.pk}_{\mathsf{send}}) - \mathsf{tx.amt} * (\mathsf{tx.specId} = \mathsf{specId}))$
  $\mathsf{amt}_{\mathsf{recv}} \leftarrow \mathsf{MT.get}(\mathsf{tx.pk}_{\mathsf{send}}) + \mathsf{tx.amt} * (\mathsf{tx.specId} = \mathsf{specId}))$
  $\mathsf{rt}_{\mathsf{send}} \leftarrow \mathsf{MT.insert}(\mathsf{r}, \mathsf{tx.pk}_{\mathsf{send}}, \mathsf{amt}_{\mathsf{send}})$
  $\mathsf{rt}_{\mathsf{recv}} \leftarrow \mathsf{MT.insert}(\mathsf{r}, \mathsf{tx.pk}_{\mathsf{recv}}, \mathsf{amt}_{\mathsf{recv}})$
  return $\mathsf{rt}_{\mathsf{recv}}$
end

Metric 4.1 (Number of active users who own instances of this spec): Suppose a user is considered "active" if he or she has made at least one transaction within an interval of recent history $[e_1, e_2]$. Let $r_0 = \mathsf{rt}$, or the root of a Merkle tree where each leaf representing the balance and activity of a different user is initialized to (0, false), and $\mathsf{aux}_f = (\mathsf{e}_{\mathsf{tx}}, \mathsf{specId})$ where e_{tx} is the epoch of the block to which this transaction belongs. We define the reputation function f_{rep} in Algorithm 6. This algorithm is similar to Metric 3, except that each leaf contains a tuple of values that must be updated, instead of just a single value. The number of leaves with positive balances and set boolean activity flags represents the number of users who own instances of the given ZUZ specification.

Algorithm 6: f_{rep} for Metric 4.1

Data: $r, \mathsf{tx}, \mathsf{specId}$
Result: $\mathsf{rt}_{\mathsf{recv}}$
begin
  $(\mathsf{amt}_{\mathsf{send}}, \mathsf{act}_{\mathsf{send}}) \leftarrow \mathsf{MT.get}(\mathsf{tx.pk}_{\mathsf{send}})$
  $\mathsf{amt}'_{\mathsf{send}} \leftarrow \mathsf{amt}_{\mathsf{send}} - \mathsf{tx.amt} * (\mathsf{tx.specId} = \mathsf{specId}))$
  $\mathsf{act}'_{\mathsf{send}} \leftarrow \mathsf{act}_{\mathsf{send}} \vee (e_1 \leq \mathsf{e}_{\mathsf{tx}} \leq e_2)$
  $(\mathsf{amt}_{\mathsf{recv}}, \mathsf{act}_{\mathsf{recv}}) \leftarrow \mathsf{MT.get}(\mathsf{tx.pk}_{\mathsf{recv}})$
  $\mathsf{amt}'_{\mathsf{recv}} \leftarrow \mathsf{amt}_{\mathsf{recv}} + \mathsf{tx.amt} * (\mathsf{tx.specId} = \mathsf{specId}))$
  $\mathsf{act}'_{\mathsf{recv}} \leftarrow \mathsf{act}_{\mathsf{recv}} \vee (e_1 \leq \mathsf{e}_{\mathsf{tx}} \leq e_2)$
  $\mathsf{rt}_{\mathsf{send}} \leftarrow \mathsf{MT.insert}(\mathsf{r}, \mathsf{tx.pk}_{\mathsf{send}}, (\mathsf{amt}'_{\mathsf{send}}, \mathsf{act}'_{\mathsf{send}}))$
  $\mathsf{rt}_{\mathsf{recv}} \leftarrow \mathsf{MT.insert}(\mathsf{rt}_{\mathsf{send}}, \mathsf{tx.pk}_{\mathsf{recv}}, (\mathsf{amt}'_{\mathsf{recv}}, \mathsf{act}'_{\mathsf{recv}}))$
  return $\mathsf{rt}_{\mathsf{recv}}$
end

Metric 4.2 (Number of active users who own instances of this spec): Suppose a user is considered "active" if he or she trades consistently, or has made at least c trades over any ZUZ specification within an interval of recent history $[e_1, e_2]$. Let $r_0 = \mathsf{rt}$, or the root of a Merkle tree where

each leaf representing the balance and activity of a different user is initialized to (0, false), and $\mathsf{aux}_f = (\mathsf{e_{tx}}, \mathsf{specId})$ where $\mathsf{e_{tx}}$ is the epoch of the block to which this transaction belongs. We define the reputation function f_{rep} in Algorithm 7. This algorithm is similar to Metric 3, except that each leaf contains a tuple of values that must be updated, instead of just a single value. The number of leaves with positive balances and at least c frequency in trades represents the number of users who own instances of the given ZUZ specification.

Algorithm 7: f_{rep} for Metric 4.2

Data: $r, \mathsf{tx}, \mathsf{specId}$
Result: $\mathsf{rt_{recv}}$
begin
  $(\mathsf{amt_{send}}, \mathsf{act_{send}}) \leftarrow \mathsf{MT.get}(\mathsf{tx.pk_{send}})$
  $\mathsf{amt'_{send}} \leftarrow \mathsf{amt_{send}} - \mathsf{tx.amt} * (\mathsf{tx.specId} = \mathsf{specId}))$
  $\mathsf{act'_{send}} \leftarrow \mathsf{act_{send}} + (e_1 \leq \mathsf{e_{tx}} \leq e_2)$
  $(\mathsf{amt_{recv}}, \mathsf{act_{recv}}) \leftarrow \mathsf{MT.get}(\mathsf{tx.pk_{recv}})$
  $\mathsf{amt'_{recv}} \leftarrow \mathsf{amt_{recv}} + \mathsf{tx.amt} * (\mathsf{tx.specId} = \mathsf{specId}))$
  $\mathsf{act'_{recv}} \leftarrow \mathsf{act_{recv}} + (e_1 \leq \mathsf{e_{tx}} \leq e_2)$
  $\mathsf{rt_{send}} \leftarrow \mathsf{MT.insert}(\mathsf{r}, \mathsf{tx.pk_{send}}, (\mathsf{amt'_{send}}, \mathsf{act'_{send}}))$
  $\mathsf{rt_{recv}} \leftarrow \mathsf{MT.insert}(\mathsf{rt_{send}}, \mathsf{tx.pk_{recv}}, (\mathsf{amt'_{recv}}, \mathsf{act'_{recv}}))$
  return $\mathsf{rt_{recv}}$
end

Decentralized marketplaces

We claim that metrics presented above to analyze the reputation of tokens on the ZUZ network can be applied to measure the worth of various Ethereum assets in decentralized marketplaces. Ethereum has a diverse token system, consisting of different assets whose worth can be hard to efficiently compare to each other. Since Ethereum is a permissionless network, any user can write a smart contract that adheres to any of the Ethereum token standards and begin trading their digital assets. For instance, the above metrics can be used to compare two fungible tokens under the ERC-20 standard based on their smart contract transaction histories.

5.4.2 General Functions

Our existing construction for monotonic reputation scoring functions can be extended to be a general construction for arbitrary reputation scoring functions by enforcing that all transactions from each block are included as private witness to each step of IVC. In order to reduce the burden of computation on the user, we claim that the user can outsource the overhead of computing from the genesis block up to a more recent block to a trusted server.

The user obtains the signature of the trusted server to describes the user's reputation r_0 for a particular application between the genesis block up to some recent block B_0, before which the block's transactions are generally less relevant to the user's reputation. The user can continue

updating their proof of reputation for new blocks that are added, which are presumably more relevant to their transaction history.

With an efficient scheme to compute statements of blockchain reputation with respect to a general reputation scoring function, this proof system becomes usable for a much wider scope of applications. In this section, we consider how to improve the financial resilience of the most popular Ethereum DApps, namely decentralized exchanges and loan pools.

Decentralized Exchanges

One of the most popular applications on Ethereum today is UniSwap, which is a decentralized exchange that is implemented using automated market makers (AMMs). Liquidity providers can contribute their assets towards the liquidity pools in exchange for tokens that represent their stake. They can redeem these tokens for a percentage of the trading fees over these assets. For instance, if Alice contributes $10,000$ ETH to a liquidity pool that held $100,000$ ETH in total, she would receive a token for 10% of that pool that she could redeem from the reserve which holds all trading fees relative to ETH.

Rewards for liquidity providers are parameterized solely on asset contribution, agnostic to the lending behavior of the users. This leaves exchanges vulnerable to a wide range of front-running and sandwich attacks, such as the just-in-time (JIT) liquidity attack. In this attack, the attacker searches the mempool for large pending exchange swaps and provides liquidity to the associated pool before the swap is confirmed. This allows the attacker to gain a large cut of the trading fees only to subsequently pull that liquidity out in the same block.

We can provide a simple defense to JIT liqudity attacks by measuring the reputation of liquidity providers, in proportion to the the longevity of their contributions to a given asset pool. By issuing rewards to liqudity providers based on their reputation, liquidity providers are incentivized to make long-term investments, which increases the financial stability of this applications.

Decentralized Loan Pools

In 2021, there was a total of 39.88B USD value locked in decentralized loan pools. Compound, Aave, and MakerDAO are some of the most widely-used loan pools, representing 85% of the lending market on Ethereum [21]. In a loan pool, users can choose to either contribute or borrow assets. Borrowers have to put down collateral up front, usually worth an amount that is much greater than the asset that they are taking out a loan for.

While there is no specific time limit by which the borrower has to pay back their loan, the “health" of the loan is continually assessed, by comparing the worth of the borrower’s collateral to the asset that was loaned out. If at any point the health of the loan drops too low, the smart contract will automatically liquidate some percentage of their asset. The specific percentage that they can liquidate is called the *close factor*.

The loans issued by these various applications are usually parameterized by fixed values. For instance, both Compound and Aave have the same close factor of 50%. However, by taking the user’s history in borrowing into account, a loan can be parameterized dynamically. For instance, the close factor of a user’s loan could be determined from their reputation. Note that since the

value of collateral put up by the borrower does not change, investors do not take on any external risk.

By measuring borrower's reputation, or more specifically their ability to repay their debts, we can make smarter loans and give users an incentive to cleanup their bad debts. Bad debt is a situation in which it is not financially rational for the borrower to close his lending position. This occurs, for instance, when the collateral value falls below the value of the debt, or the cost of a transaction fee exceeds the collateral that a borrower wants to reclaim by closing his position.

Chapter 6

Conclusion

6.1 Summary

We explore the conflict between privacy and reputation in the decentralized token system, ZUZ, in several ways. First, we identify how to improve privacy guarantees for ZUZ in both the UTXO- and account-based models. We design ZUZ as an extension of Zerocash, which is a fully anonymous version of Bitcoin, and we also design ZUZ as a privacy-preserving smart contract atop Ethereum.

Whereas we achieve anonymity over origin, destination, and value under Zerocash in the UTXO-based model, we only achieve anonymity over destination and value in the account-based model. However, our UTXO-based protocol also inherits a few downsides from Zerocash, including its trusted setup and a monotonically increasing UTXO set. On the other hand, our privacy-preserving smart contract provides a concise representation of public and private user funds which can eventually be deployed on a mobile or low-resource client. Furthermore, we propose multi-round extensions of our account-based protocol to enhance our privacy guarantees.

In Figure 6.1, we compare the various implementations of ZUZ presented in this thesis in terms of privacy level and transfer costs. In particular, we label the UTXO-based implementation as ZUZ-UTXO and the smart contract-based implementation as ZUZ-C and ZUZ-KA, to differentiate between confidential and destination-anonymous transfers. Since the UTXO-based implementation is an extension of Zerocash, a fully-anonymous privacy-preserving payment system, ZUZ-UTXO similarly achieves full privacy over origin, destination, and value of each transfer. We estimate the "gas cost" of such a transfer based on ZETH, the smart contract representation of Zercoash.

We use two different labels, ZUZ-C and ZUZ-KA, for our smart contract-based implementation in order to compare a confidential transfer under the ZUZ smart contract to a confidential transfer under other privacy-preserving smart contracts, such as Zether. In particular, we denote the confidential payment smart contract as Zether-C. As shown, ZUZ-C achieves confidentiality for nearly half the gas cost as Zether-C. In fact, ZUZ-KA actually hides the recipient of a transfer among up to 5 users for the same cost as a single Zether-C transaction.

We note that our work addresses a unique middle ground in decentralized privacy-preserving

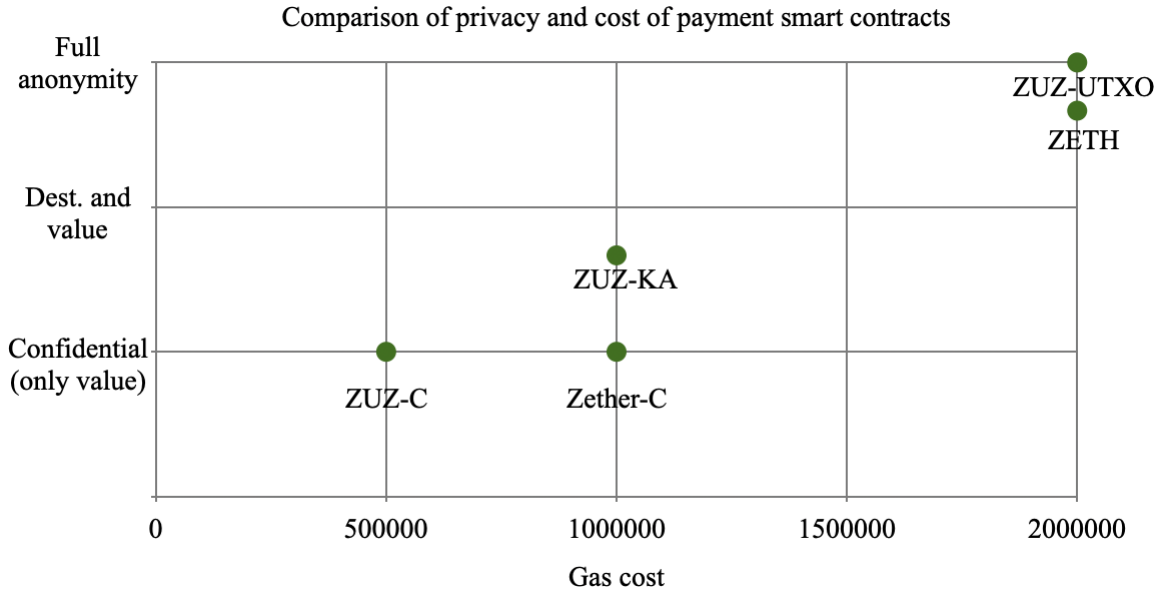

Figure 6.1: Comparison of ZUZ-C (confidential), ZUZ-KA (destination anonymous), and ZUZ-UTXO (fully anonymous) in terms of costs and privacy

payment systems, which lies between confidentiality and full anonymity. Recall that a payment includes a sender or origin, recipient or destination, and value or amount. A confidential payment system aims to conceal just the value or amount of each transfer whereas a full anonymous payment systems aims to conceal all three values. Through our destination- and value-anonymous transfers, we achieve hiding in two out of the three values.

Furthemore, note that in the smart contract setting, we cannot achieve “full" anonymity of sender or recipient in the sense that it is infeasible to update *all* accounts for each transfer. As a result, we observe the behavior of a k-anonymous transfer, or how the cost of a transfer scales as k accounts are updated, or as the privacy of the transfer increases. As a result, both the smart contracts and ZUZ-KA and ZETH are placed slightly below the line for destination- and value-anonymity and full anonymity, respectively, since they approach these levels of privacy.

Finally, we also introduce a novel primitive, statements of blockchain reputation, which uses incrementally verifiable computation to efficiently prove and verify computation over a public ledger. This primitive formalizes the notion of reputation with respect to a token specification, which we can eventually extend to a privacy-preserving ledger. We present a wide variety of decentralized applications, apart from ZUZ, in which this primitive can be used in the public setting.

6.2 Future Work

We leave further optimizations of the privacy-preserving smart contract to future work. In particular, we highlight specific problems, such as designing a fewer-round origin-anonymous transfer for the privacy-preserving smart contract or considering how malicious agents may impact the

use of client-side validation in a multi-round protocol.

We also leave further optimizations of the IVC scheme for blockchain reputation to future work. In particular, we are interested in finding an efficient solution for an arbitrary reputation scoring function and finding an efficient solution in the private ledger setting, where reputation is being computed for an artifact that multiple users interact with.

The latter case corresponds exactly to the problem of calculating the reputation of a token specification on a privacy-preserving ZUZ ledger. We leave optimizations for integrating these statements of reputation into a privacy-preserving ledger to future work as well.

Bibliography

[1] https://github.com/monatized/zuz-account. 4.3

[2] https://github.com/sarayu-namineni/zuz-utxo. 3.3

[3] Ghada Almashaqbeh and Ravital Solomon. Sok: Privacy-preserving computing in the blockchain era. Cryptology ePrint Archive, Paper 2021/727, 2021. URL https://eprint.iacr.org/2021/727. 1

[4] Kurt M. Alonso and Jordi Herrera Joancomartí. Monero - privacy in the blockchain. Cryptology ePrint Archive, Paper 2018/535, 2018. URL https://eprint.iacr.org/2018/535. 1

[5] Elli Androulaki, Jan Camenisch, Angelo De Caro, Maria Dubovitskaya, Kaoutar Elkhiyaoui, and Björn Tackmann. Privacy-preserving auditable token payments in a permissioned blockchain system. In *Proceedings of the 2nd ACM Conference on Advances in Financial Technologies*, pages 255–267, 2020. 2

[6] Simon Barber, Xavier Boyen, Elaine Shi, and Ersin Uzun. Bitter to better - how to make bitcoin a better currency. In A D Keromytis, editor, *Financial Cryptography and Data Security: 16th International Conference, FC 2012 Revised Selected Papers [Lecture Notes in Computer Science, Volume 7397]*, pages 399–414. Springer, Germany, 2012. doi: 10.1007/978-3-642-32946-3_29. URL https://eprints.qut.edu.au/69169/. 3.1

[7] Amira Barki and Aline Gouget. Achieving privacy and accountability in traceable digital currency. *Cryptology ePrint Archive*, 2020. URL https://eprint.iacr.org/2020/1565. 2

[8] Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza. Succinct non-interactive zero knowledge for a von neumann architecture. Cryptology ePrint Archive, Paper 2013/879, 2013. URL https://eprint.iacr.org/2013/879. 3.1

[9] Eli Ben-Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized anonymous payments from bitcoin. Cryptology ePrint Archive, Paper 2014/349, 2014. URL https://eprint.iacr.org/2014/349. 1, 3.1, 4, 5

[10] Benedikt Bünz, Shashank Agrawal, Mahdi Zamani, and Dan Boneh. Zether: Towards privacy in a smart contract world. Cryptology ePrint Archive, Paper 2019/191, 2019. URL https://eprint.iacr.org/2019/191. 1, 4.1

[11] Antonio Salazar Cardozo and Zachary Williamson. EIP 1108: reduce alt_bn128 precom-

pile gas costs, 2018. URL https://github.com/ethereum/EIPs/blob/master/EIPS/eip-1108.md. 4.4

[12] Panagiotis Chatzigiannis and Foteini Baldimtsi. Miniledger: Compact-sized anonymous and auditable distributed payments. Cryptology ePrint Archive, Paper 2021/869, 2021. URL https://eprint.iacr.org/2021/869. 2

[13] Yu Chen, Xuecheng Ma, Cong Tang, and Man Ho Au. Pgc: Pretty good decentralized confidential payment system with auditability. Cryptology ePrint Archive, Paper 2019/319, 2019. URL https://eprint.iacr.org/2019/319. https://eprint.iacr.org/2019/319. 2

[14] Benjamin E. Diamond. Many-out-of-many proofs and applications to anonymous zether. Cryptology ePrint Archive, Paper 2020/293, 2020. URL https://eprint.iacr.org/2020/293. 4.5.1

[15] Tassos Dimitriou. Decentralized reputation. In *ACM Conference on Data and Application Security and Privacy*, 2021. https://doi.org/10.1145/3422337.3447839. 1

[16] Prastudy Fauzi, Sarah Meiklejohn, Rebekah Mercer, and Claudio Orlandi. Quisquis: A new design for anonymous cryptocurrencies. Cryptology ePrint Archive, Paper 2018/990, 2018. URL https://eprint.iacr.org/2018/990. 4.1

[17] Christina Garman, Matthew Green, and Ian Miers. Accountable privacy for decentralized anonymous payments. Cryptology ePrint Archive, Paper 2016/061, 2016. URL https://eprint.iacr.org/2016/061. 2

[18] Seth Copen Goldstein, Denizalp Goktas, Miles Conn, Shanmukha Phani Teja Pitchuka, Mohammed Sameer, Maya Shah, Colin Swett, Hefei Tu, Shrinath Viswanathan, and Jessica Xiao. Bolt: Building on local trust to solve lending market failure, 2020. URL https://www.cs.cmu.edu/~seth/bolt/bolt-draft.pdf. 1

[19] Abhiram Kothapalli, Srinath Setty, and Ioanna Tzialla. Nova: Recursive zero-knowledge arguments from folding schemes. Cryptology ePrint Archive, Paper 2021/370, 2021. URL https://eprint.iacr.org/2021/370. 1

[20] Neha Narula, Willy Vasquez, and Madars Virza. zkLedger: Privacy-Preserving auditing for distributed ledgers. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, pages 65–80, Renton, WA, April 2018. USENIX Association. ISBN 978-1-939133-01-4. URL https://www.usenix.org/conference/nsdi18/presentation/narula. 2

[21] Kaihua Qin, Liyi Zhou, Pablo Gamito, Philipp Jovanovic, and Arthur Gervais. An empirical study of defi liquidations: Incentives, risks, and instabilities. In *Proceedings of the 21st ACM Internet Measurement Conference*, pages 336–350, 2021. 5.4.2

[22] Fergal Reid and Martin Harrigan. An analysis of anonymity in the bitcoin system. In *Security and Privacy in Social Networks*, pages 197–223. Springer, 2013. 3.1

[23] Dorit Ron and Adi Shamir. Quantitative analysis of the full bitcoin transaction graph. Cryptology ePrint Archive, Paper 2012/584, 2012. URL https://eprint.iacr.org/2012/584. 3.1

[24] Antoine Rondelet. Thinking around integrating zerocash on ethereum, 2018. URL https://github.com/AntoineRondelet/zerocash-ethereum. 4.1

[25] Antoine Rondelet and Michal Zajac. ZETH: on integrating zerocash on ethereum. *CoRR*, abs/1904.00905, 2019. URL http://arxiv.org/abs/1904.00905. 4.5.1

[26] Zooko Wilcox. The design of the ceremony, 2016. URL https://electriccoin.co/blog/the-design-of-the-ceremony/. 3.1

[27] Karl Wüst, Kari Kostiainen, Vedran Čapkun, and Srdjan Čapkun. Prcash: Fast, private and regulated transactions for digital currencies. In Ian Goldberg and Tyler Moore, editors, *Financial Cryptography and Data Security*, pages 158–178, Cham, 2019. Springer International Publishing. ISBN 978-3-030-32101-7. 2