Privacy-Preserving Reputation-Based Lending Smart Contract

Sarayu Namineni
Carnegie Mellon University

Seth Copen Goldstein
Carnegie Mellon University

Abstract

In a reputation-based lending system, the acceptability of a scrip is measured through its transaction history, creating an inherent trade-off between contributing either to the reputation of a scrip or the privacy of a transfer. Our proposal, *ZUZ*, maintains a concise but flexible state, where the private balance alternates between an account- and a UTXO-like wallet representation. We implement destination- and value-anonymous transfers over rings of users, which obfuscates the link between a sender and a recipient through *plausible deniability*, while still allowing the underlying scrip to gain reputation. We also design transfer protocols under this model with stronger privacy guarantees, including a "pay for privacy" scheme and origin-anonymous transfers. Our approach results in practical online computation speeds and constant gas costs, whereas total transaction times scales linearly in the anonymity set.

1 Introduction

In a reputation-based lending system, individuals and businesses can issue their own *scrips* to be redeemed in exchange for goods and services in the future. In turn, these scrips gain value when they are traded by individuals or at businesses outside of the ones that issued them.

Reputation-based lending systems mitigate issues present in the current lending market, such as *asymmetric information* and *imperfect competition*, by recording all transactions on a *distributed public ledger*. Transactions made under a transparent and universally accessible system contribute to the reputation of an individual or business, as measured by the acceptability of their scrips.

However, with the rise of *privacy coins* [1] such as Zerocash and Monero, it is increasingly clear that users are opposed to having their transaction history be linked to their public identity. Concerned with the privacy of their digital transactions, users are averse to leaving the very digital trace necessitated by reputation-based lending systems.

Our first attempt to conduct anonymous transactions in a reputation-based system is as follows. Simply put, a user takes a new pseudonym and defines a new scrip under it. Then, we claim that any instances that they mint from this scrip constitutes anonymous payment [6].

Since neither the scrip nor the pseudonym have any history associated with them, the transaction is completely anonymous; however, for the very 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, such a system is rendered unusable.

Thus, our goal is to design a privacy-preserving payment scheme over the scrips in a reputation-based lending smart contract. This payment mechanism will allow users to privately transfer instances of user-defined scrips, by concealing the amount of a given scrip that is transferred and unlinking the sender from the recipient of a transfer, all while allowing the underlying scrip to accrue reputation.

2 Background and Motivation

Bitcoin and Ethereum are the two cryptocurrencies with the highest market capitalization, but there is an important difference in the way that they store information. Bitcoin stores a list of all unspent transaction outputs, or *UTXOs*, whereas Ethereum maintains a balance for every *account*. Neither cryptocurrency natively supports private transactions.

For UTXO-based blockchains, there are well-known solutions that allow for private payments, such as Zerocash [3], Monero [2], and Dash. On the other hand, for account-based blockchains, there has been no major trend towards adoption of any single privacy-preserving technology.

Importantly, due to their intuitive representation of state, account-based blockchains facilitate the development of various applications atop them, or *smart contracts*. Designing a practical privacy-preserving smart contract presents the opportunity to add a meaningful level of privacy to the numerous decentralized applications, or *dApps*, on account-based blockchains.

2.1 Account-Based Approaches

For a given user u and ZUZ specification s, a public ZUZ instance is represented by a single numerical value b_{pub}, which is the user's balance on this ZUZ specification. A private ZUZ instance is represented as a commitment cm of some numerical value b_{priv} representing the amount of private funds that the user owns on this ZUZ specification. This is the most concise representation of the ledger state that we can achieve under this model, given that there must be a distinction between public and private funds.

However, enforcing an account-based representation of private wallet funds at all times makes the design vulnerable to *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.

Zether [4], which homomorphically encrypts users' account balances, faces the same issue. Their solution is to maintain a table of pending transfers, instead of updating the users' actual account balances. However, this leads to the issue of when to rollover the funds into the user's accounts without inadvertently launching a front-running attack again. Their solution is to rollover a user's funds the next time they make a transaction so that they are available to spend during the transaction.

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.

Similar to how Zether rolls over pending transfers at the start of the next transaction, we also enforce that users combine their list of wallet commitments during an operation (see section 3.2.1). The difference is that this computation is offloaded to the user. By shifting the burden of computation, we can achieve more practical smart contract compute speeds and gas costs.

2.2 UTXO-based approaches

Under the ZUZ protocol, there is a one-way flexibility when it comes to representing their private account balance as a list of wallet commitments. We enforce that the commitments are joined after every operation, but we do not allow users to split their existing commitments any further. Rather, the multiplicity of wallet representation can only be triggered by external events. We briefly consider why it would be impractical to take a completely UTXO-based approach, with both join and split functionality.

ZETH [9] is a smart contract integration of Zerocash, a UTXO-based privacy-preserving payment scheme which allows users to commit their funds into a pool and anonymously spend from that pool by proving ownership in zero-knowledge over some funds in set of all commitments. In ZETH, the pour operation, which allows users to transfer funds, is generalized to convert N private inputs into M private outputs. This generality allows users to join and split their funds arbitrarily.

Even though Zerocash takes a Bitcoin-like approach in their design, there is an important difference between the two systems. 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* [7]. 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 the smart contract implementation, ZETH [8]. 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.

In our design, we address this situation by 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.

3 ZUZ Smart Contract

3.1 Definitions

First, we define the objects in a reputation-based lending system. There is a user, who conducts transactions under various *pseudonyms*, each represented by a different account keypair. We refer to the denomination of all user-defined currency as *ZUZ*. The user has the ability to issue new scrips, which we will call *ZUZ specifications* under any one of their pseudonyms. When the user wants to put their specification into circulation, they mint *ZUZ instances*.

3.2 Implementation

The implementation currently consists of creating ZUZ specifications, minting, and transferring public ZUZ instances, as well as converting private instances to and from public instances and performing destination-and value- anonymous transfers over private instances.

The specification for transferring and converting to and from private instances consists of a single operation:

Fund $(s, [pk_1, ..., pk_r], [cm_1^{recv}, ..., cm_r^{recv}], [cm_1^{old}, ..., cm_w^{old}], cm^{new}, b_{pub}^{old}, b_{pub}^{new}, [Enc_{pk_1}(b_1, \omega_1)...Enc_{pk_r}(b_r, \omega_r)], \pi_{FUND})$, which takes in a ZUZ specification s, a list of recipients for the transaction $[pk_1, ..., pk_r]$, a list of the private transfer commitments respective to the list of recipients $[cm_1^{recv}, ..., cm_r^{recv}]$, a list of the sender's current private wallet commitments $[cm_1^{old}, ..., cm_w^{old}]$, the sender's new private wallet commitment cm^{new}, the sender's current public balance b_{pub}^{old}, the sender's new public balance b_{pub}^{new}, a list of the parameters (i.e. the transfer amount b_i and randomeness ω_i) needed to unlock each of the transfer commitments, encrypted with respect to the list of recipients $[Enc_{pk_1}(b_1, \omega_1)...Enc_{pk_r}(b_r, \omega_r)]$, and a zero-knowledge proof π_{FUND} that shows that the transaction is well-formed and the sender has sufficient funds for transaction.

The smart contract must validate that list of the sender's wallet commitments matches the state of the sender's wallet on the ledger to prevent double-spending attacks. If the smart contract is able to validate the inputs to the zero-knowledge proof and verify the zero-knowledge proof, then it will append the update the sender's wallet with the new commitment and append transfer commitments to their respective accounts.

3.2.1 Conversions

First, we demonstrate how the fund operation can be used to convert public ZUZ instances into private ones, and vice versa. A private balance is represented as a commitment of a numerical value b and randomness ω over a spec s. The fund operation adds a commitment to the sender's wallet, as well as all the listed recipient's wallets, given that the sender has sufficient funds to make the transfer. Thus, in order to convert public ZUZ instances into private ones, or vice versa, the user can invoke **Fund** with empty lists of recipients, recipient transfer commitments, and recipient transfer parameters.

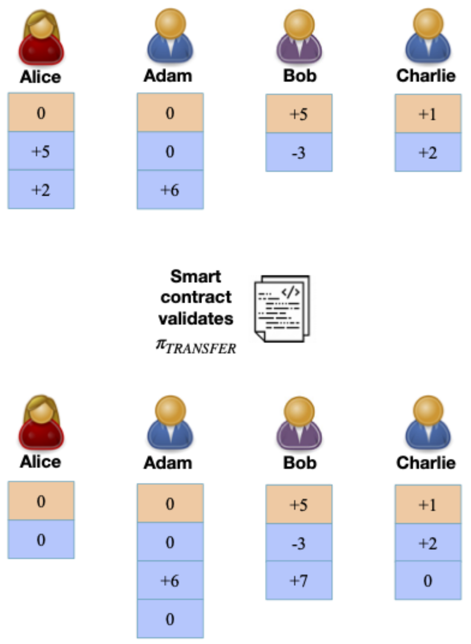


Figure 1: Wallet representations in a destination- and value-anonymous transfer

3.2.2 Destination- and Value-Anonymous Transfer

Next, we illustrate how the fund operation can be used to perform a destination- and-value anonymous transfer (Figure 1). The sender specifies a ring of recipient users to send a transfer commitment to. Only one of these users needs to have a non-zero transfer balance. The sender will broadcast the parameters needed to unlock the transfer commitments, which are the transfer balance b_i and randomness ω_i. (For the proof of concept implementation, ω_i is omitted). These parameters will be encrypted with the public keys of the respective recipient, so only the sender and recipient will know the value in the commitment.

Note that by the nature of smart contract transactions, only the owner of an account can authorize expenditure of the private wallet commitments stored in their account. This is because transactions are signed with the private key of the sender, which means that in order to initiate a transaction to spend a wallet commitment stored in an account, one must know the private key corresponding to that account. Even though the sender and the recipient both know the value in the transfer commitment, the sender cannot spend it since constructing such a transaction would require knowledge of the recipient's private key.

This specification guarantees that the sender's account balance returns to a concise, dual-balance representation after every operation. The sender provides a list of their current wallet commitments and a new commitment, which represents the aggregate state over all their private balances, including the private balance that is implicit in the transfer.

The zero-knowledge proof ensures that the transfer is well-formed and that the sender has sufficient funds for to complete the transfer. More specifically, it checks that the sum over all private transfer balances $b_1^{recv} + \ldots + b_r^{recv}$ that unlock the transfer commitments $cm_1^{recv}, \ldots, cm_r^{recv}$ is less than or equal to the sum over sender's public and private balances before the transfer, or $b_1^{recv} + \ldots + b_r^{recv} \leq b_{pub}^{old} + b_1^{old} + \ldots + b_w^{old}$. It also ensures that the sender's new public and private balance splits the difference between these two values, or $b_{pub}^{old} + b_1^{old} + \ldots + b_w^{old} - (b_1^{recv} + \ldots + b_r^{recv}) = b_{pub}^{new} + b_{priv}^{new}$.

Incentive to pay for privacy With a destination- and value-anonymous transfer, every recipient has *plausible deniability* since it is unknown who the intended recipient of the transfer was. Furthermore, since our scheme allows for the arbitrary denomination of private balances and every transfer joins all the wallet commitments of a given user, it is impossible to explicitly trace the flow of a commitment between sender and recipient. This follows from the security analysis in ZETH, which only guarantees destination-and value-anonymity transfers over arbitrary denominations using a mixer contract [9].

The only difference in the security analysis 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 (which may include the recipient themselves). Even if no one has privately sent funds to a particular user before, the user still has some privacy, or plausible deniability, over the size of the ring specified by the sender to transfer funds.

However, there arises a discrepancy in incentives when it comes to paying for privacy. Under the model presented thus far, the sender of a transaction funds the gas cost for private transfer. Although the privacy for the recipient increases as the size of the ring increases, the cost of a transfer does too. Thus, the sender must pay for the recipients' privacy, which does not always align with the sender's incentives.

3.2.3 Attempt at Origin-Anonymous Transfer

In order to make a transaction on an account-based blockchain like Ethereum, the sender of the transaction must pay some gas fees. A prerequisite to making an origin-anonymous transfer, then, is to be able to acquire and spend ETH on an account that is distinct from the origin of the transfer of the user-defined scrip.

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,

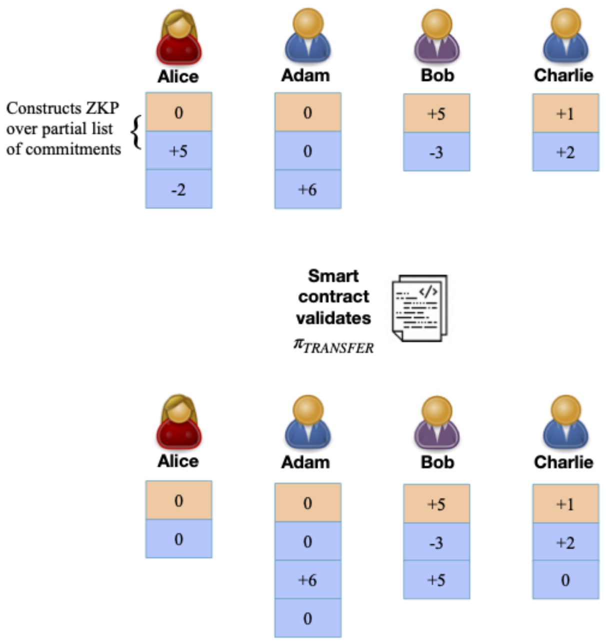


Figure 2: Wallet representations in a double-spending attack

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 2, 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 the sender's wallet commmitments used as input to the zero-knowledge proof are consistent with the ledger state.

However, in the 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 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.

4 Transfer Protocols

4.1 "Pay for Privacy" Scheme

First, we address the mismatch in incentives in a destination- and value-anonymous transfer by presenting a "pay for privacy" scheme, 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, not ETH. Also note that the person who is being requested to send money, or the sender of the transfer, is still a publicly known parameter.

Suppose that Bob wants to request Alice for funds. Then, from a distinct account, 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 Alice is the only person who will be able to tell who requested funds from her since Bob's balance will be nonzero in the 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.

4.2 Origin-Anonymous Transfer

Next, we extend the above protocol in order to conceal the origin of a transfer. From a distinct account, 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

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

Table 1: Time complexity of destination- and value-anonymous transfer

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.

Building off of the prior scheme, client-side validation is now used by both parties in the transfer.

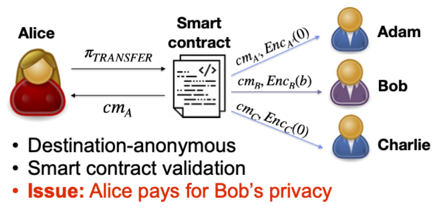
4.3 Summary

The protocols summarized in Figure 3 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.

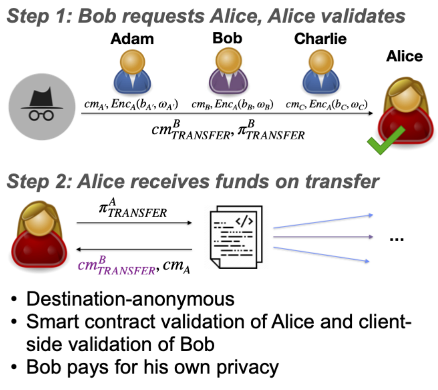
In this way, the recipient is able to pay for their privacy, providing the sender with an explicit transfer specification and some compensation to cover the cost of a private transfer. Nevertheless, the sender is still a public parameter throughout the two-step transfer process. By extending the protocol 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.

5 Results

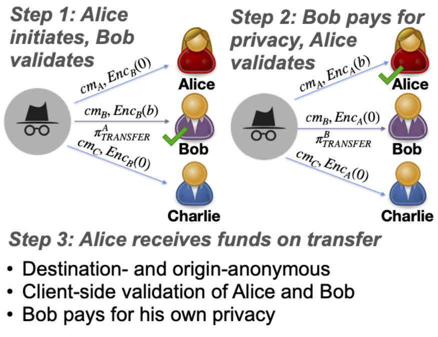
In our implementation, we wrote a smart contract in Solidity to model the reputation-based lending system and used the Zokrates library to perform offline proof generation and generate smart contracts for proof verification. One limitation of this library is that its Javascript bindings cannot withstand larger circuits, so in future iterations of this implementation



(a) One-step transfer with destination-and value-anonymity


Step 1: Bob requests Alice, Alice validates
Adam
Bob
Charlie
Alice
$cm_{A'}, Enc_A(b_{A'}, \omega_{A'})$
$cm_B, Enc_A(b_B, \omega_B)$
$cm_C, Enc_A(b_C, \omega_C)$
$cm^B_{TRANSFER}, \pi^B_{TRANSFER}$
Step 2: Alice receives funds on transfer
$\pi^A_{TRANSFER}$
$cm^B_{TRANSFER}, cm_A$
...
• Destination-anonymous
• Smart contract validation of Alice and client-side validation of Bob
• Bob pays for his own privacy

(b) Two-step transfer with "pay for privacy"


Step 1: Alice initiates, Bob validates
$cm_{A'}, Enc_B(0)$
$cm_B, Enc_B(b)$
$\pi^A_{TRANSFER}$
$cm_C, Enc_B(0)$
Alice
Bob
Charlie
Step 2: Bob pays for privacy, Alice validates
$cm_{A'}, Enc_A(b)$
$cm_B, Enc_A(0)$
$\pi^B_{TRANSFER}$
$cm_C, Enc_A(0)$
Alice
Bob
Charlie
Step 3: Alice receives funds on transfer
• Destination- and origin-anonymous
• Client-side validation of Alice and Bob
• Bob pays for his own privacy

(c) Three-step transfer with origin anonymity

Figure 3: Summary of private transfer protocols

we will switch to using a lower-level zero-knowledge proof library.

Note that in the proof of concept implementation provided, the specification is implemented as three separate operations, which correspond to each type of conversion as well as a one-step transfer. Some minor details are omitted, such as ω_i when generating the commitments or public balances evaluated as part of the list of parameters.

As shown in Figure 4, the times for a destination-and value-anonymous scale 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

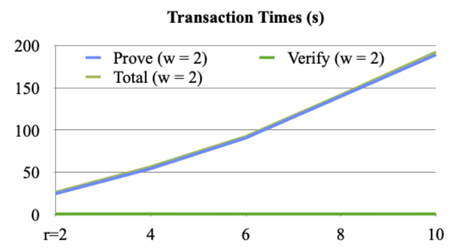


Figure 4: Destination- and value-anonymous transfer times

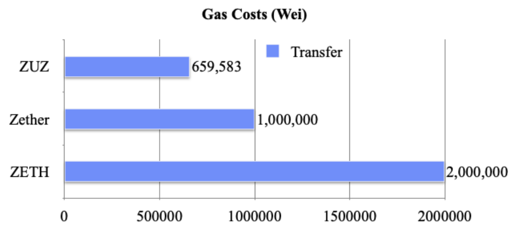


Figure 5: Comparison of gas costs for privacy-preserving payment smart contracts

proof verification by the smart contract, remain constant.

Furthermore, in Figure 5, we see that gas costs for transfers are significantly lower than comparable privacy-preserving smart contract payment schemes [5], due to the fact that client-side computation outweighs smart contract computation in our scheme.

6 Conclusion

Moving computation offline results in practical smart contract transaction speeds and gas costs for destination- and value-anonymous transfers. Furthermore, by introducing client-side validation, our protocols allow senders to conceal their identity and recipients to pay for their privacy.

One remaining challenge is mitigating *wash attacks* on the reputation of ZUZ specifications under private transfers. We also hope to analyze the system against various attack vectors, such as malicious clients or statistical inference attacks.

Future work aims to reduce the time complexity of client-side proof generation, by considering alternative wallet representations, such as Merkle trees, or pre-compiling zero-knowledge proofs.

References

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

[2] Kurt M. Alonso and Jordi Herrera Joancomartí. Monero - privacy in the blockchain. Cryptology ePrint Archive,

Paper 2018/535, 2018. https://eprint.iacr.org/2018/535.

[3] 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. https://eprint.iacr.org/2014/349.

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

[5] Antonio Salazar Cardozo and Zachary Williamson. EIP 1108: reduce alt_bn128 precompile gas costs. https://github.com/ethereum/EIPs/blob/master/EIPS/eip-1108.md.

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

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

[8] Antoine Rondelet. Thinking around integrating zerocash on ethereum. https://github.com/AntoineRondelet/zerocash-ethereum.

[9] Antoine Rondelet and Michal Zajac. ZETH: on integrating zerocash on ethereum. *CoRR*, abs/1904.00905, 2019. https://arxiv.org/pdf/1904.00905.pdf.