Private and Secure Mixing in Credit Networks

Lalitha Muthu Subramanian
Department of Computer Science,
New Mexico State University, USA
lalitha@nmsu.edu

Guruprasad Eswaraiah
Department of Computer Science,
New Mexico State University, USA
guru@nmsu.edu

Roopa Vishwanathan
Department of Computer Science,
New Mexico State University, USA
roopav@nmsu.edu

ABSTRACT

In this paper, we propose a system for mixing transactions in payment networks such as credit networks. Credit networks like Ripple and Stellar are increasingly popular, and can facilitate cross-currency transactions in a fraction of the time it would take for banks or other financial institutions to process the same transaction, and at a fraction of the cost. Unlike for cryptocurrencies, there has been little work in the area of designing secure and private mixers for credit networks. Mixers for cryptocurrencies such as Bitcoin cannot be directly applied to the credit network domain because credit networks have an inherently different structure and purpose than cryptocurrencies. We design a system that uses cryptographic constructs such as *ring signatures*, *commitments*, and *zero knowledge proofs* to provide security/integrity of all transactions, ensures privacy of the users involved in a transaction, as well as privacy of the amount transacted. We also provide preliminary experimental results.

CCS CONCEPTS

• **Security and privacy → Privacy-preserving protocols**.

KEYWORDS

payment networks; blockchain; anonymity; mixing networks

1 INTRODUCTION

Blockchain technology has enabled cryptocurrencies such as Bitcoin [22], and platforms providing smart contract functionalities such as Ethereum [7]. Credit networks such as Ripple [23] and Stellar [29] have been developed as complementary financial service systems to cryptocurrencies. Credit networks are peer-to-peer systems in which a user extends trust to other users in the network in the form of I Owe You (IOU) credits. The transaction between the users is enabled by "flowing" the IOU credits along a trusted path created among them, and writing the transaction information to a public blockchain. Credit networks enable faster transactions than banks, have the capacity to perform same and cross-currency transactions at a significantly lower cost than traditional banking solutions, and consequently are gaining popularity among users for cross-currency, global transactions. One of the major privacy/anonymity issues in this area is that credit networks like Ripple post their entire transaction logs to the blockchain for providing accountability, which make it easy to link/track the users who are involved in a transaction, and glean other transaction details.

Mixer networks are a popular solution for providing a degree of privacy to users in blockchain-enabled cryptocurrencies. TumbleBit [11] enables anonymity in credit networks where payments are sent off-blockchain, via an entity called as *tumbler* who can efficiently mix several transactions in the order of seconds. Although, TumbleBit is efficient and provides unlinkability among the users, it does not provide for the ability for users to perform *micropayments* transactions, since the tumbler allows users to make transactions worth 1 Bitcoin only. In some applications, being able to transact only in unit currency might be an inconvenience to users; additionally micropayments transactions allow users to transact in small denominations (e.g., a $3 coffee costs 0.00073 Bitcoin). Other trusted third party-based mixing protocols such as Mixcoin [4] were introduced where the mixing servers were susceptible to theft from the users, and a third party can violate anonymity. Other mixers include CoinJoin [17], which can be used opportunistically by a set of users, and can be used to achieve stronger anonymity in smaller groups, and CoinSwap [18] which can achieve anonymity in larger sets at the cost of additional transaction fees. All of the above mixers are proposed for cryptocurrencies, not credit networks.

PathShuffle [20] is a recently presented, path mixing protocol which is fully compliant with the Ripple credit network. PathShuffle offers a fully decentralized, yet fast solution for complete anonymity of Ripple transactions. PathShuffle leverages the Ripple network's *gateway wallet*, which is trusted by other users' wallets in the network to create and maintain consistent and correct credit links, and also enables micropayments. In PathShuffle, there need to be several users available for effectively mixing transactions, and each user involved in a mixing transaction knows the transaction amounts of the other users, which could lead of linkability of transactions, and loss of value (amount) privacy for a single transaction.

Our Contributions. In this paper, we design a system where a set of users can perform simultaneous transactions in a credit network while maintaining their anonymity and privacy from all other users in the network. We have multiple intermediaries in our system that mix the transactions, and privacy of the users is protected from the intermediaries too. Additionally the amount transacted by every user is not known to any other user in the network, nor does a user need to find several other users in order for the intermediary to conduct the transaction. Our system is scalable, supports micropayments and provides fast end-to-end transactions.

IECC ' 19, July 7–9, 2019, Okinawa, Japan

ACM ISBN 978-1-4503-7177-3/19/07…$15.00
DOI:https://doi.org/10.1145/3343147.3343171

2 RELATED WORK

In this section we will review literature in payment networks, cryptocurrency mixers and credit networks, and compare our network design with existing systems.

2.1 Payment channel networks

Fulgor [16] is the first payment protocol for peer-to-peer payment channel networks that provides provable privacy guarantees, and Rayo [16] is the first protocol for peer-to-peer payment channel networks that enforces non-blocking progress. In Fulgor and Rayo: 1) The users among the path between a sender and receiver would act as intermediaries and need to establish a payment path, and determine the path capacity prior to starting the transaction. 2) Every user in the path is assumed to be honest, whereas our design with multiple intermediaries do not assume every user in the path to be honest providing security against malicious intermediaries and malicious users. 3) During concurrent payments in Fulgor, deadlocks are possible. Unlike Fulgor, in our model, there will not be any deadlocks occurring over the transaction path, and users do not have to make a hard choice between anonymity and concurrency. 4) In Fulgor and Rayo, users have at least partial knowledge about network topology. In our design, none of the users will have knowledge about any part of the network topology, except for knowing the intermediaries. Non-Custodial Second Layer Financial Intermediary (NOCUST) [13] is a payment network with a single intermediary. The trusted intermediary computes the balance in a public ledger for every user, not providing anonymity, value and balance privacy. We provide value privacy by a commitment scheme where a sender provides commitments to the intermediaries and the commitments are placed on the blockchain through intermediaries for the receiver.

Teechain [14] uses trusted execution environments (TEEs) that write entries to a public blockchain to enforce correct operation against mutually distrusting entities. Although value privacy is trivially provided in a Teechain network, linkability is a problem, since any adversary can link the signed entry in the blockchain to the TEE that made these updates into the blockchain. This system also places the entirety of trust on the TEEs, thereby allowing the system to be vulnerable to attacks on the TEEs. One example of such a case would be the foreshadow attack [5]. We aim to provide unlinkability providing a commitment scheme that would not reveal any value transferred from any of the users in the network.

Khalil et al. proposed REVIVE [12], a payment channel network that allows users to re-balance their share of coins in a channel without having to communicate with the blockchain. However, the channel re-balancing process is not transparent and requires an out of band coordination. Moreover, REVIVE only works in a restricted class of network topologies that has cyclic structures and it is not clear that the re-balancing mechanism is feasible in a general topology.

Perun [8] proposed *virtual* payment networks over Blockchain. Perun allows two parties to establish a virtual payment network without interacting with the intermediaries. Although Perun was the first to introduce the concept of a virtual payment network, they propose an explicit re-balancing scheme and the sender/receiver can be linked to the value of the payment done over the payment network. In our design, we focus primarily on the sender/receiver and value privacy.

2.2 Cryptocurrencies

TumbleBit [11] provides a backward compatible, centralized mixing service where several users transfers their funds to an entity called *tumbler*, and the tumbler in turn returns them to the users at a fresh address. In our network, we allow the users to do micro-transactions, unlike [11], and preserve their privacy using ring signatures and zero knowledge proofs. Bitcoin tumblers such as Blindcoin [11] and Mixcoin [4] use a trusted third party to mix Bitcoin addresses. CoinShuffle [26] and CoinShuffle++ [27] allow only 538 users per mix, CoinJoin [17] allows just 50 users per mix. In our network, we support over 800 users.

2.3 Credit Networks

SilentWhispers [15] presents a decentralized credit network (DCN) architecture which consists of subsets of paths between the sender and receiver calculated via several trusted entities called *landmarks*. At regular time intervals, each landmark starts two instances of breadth-first-search (BFS) rooted at itself. First one is between the sender and itself and the second one between the receiver and itself. These two paths are stitched forming a complete path between the sender and receiver. [15] provides transaction integrity, accountability as well as sender receiver and transaction value privacy. It does not provide detailed mechanisms for concurrent transactions (which is essential for scalability). It is also vulnerable to deadlocks, and requires a user to join the network only at fixed time intervals. Prior to going offline, the user needs to handover the signing keys and the transaction-related data to the landmarks which will impersonate the user during her absence. In our paper, we have n intermediaries out of which k intermediaries form a honest majority and are online all the time, thereby reducing the trust placed on a single intermediary. Also we do not require path-finding between a sender and receiver.

The DCN presented by Roos et al [25] used graph embedding for efficient routing, with support for concurrent transactions overcoming the inefficiencies in [15]. The embedding algorithm constructs a rooted spanning tree of the network graph. In [25]: a) Senders choose random amounts to transmit along a path which might lead to a high rate of transaction failure. b) The user has to go though a waiting time before joining the network. Unlike [25], in our case, there is no waiting time imposed on users, and users do not transmit randomly-picked amounts, hence a transaction is guaranteed to be successful.

Panwar *et al.* [21] very recently proposed a DCN system where users can perform path-based transactions in a way that preserves sender, receiver, and value privacy. In their system, users need to perform a rather complicated and inefficient path-finding phase. Their system also has a significantly high number of messages being written to the blockchain in the course of a normal, successful transaction (more in the case of rollbacks, and other edge cases), thus resulting in very high blockchain fees being incurred per transaction. In our system, we require only a single blockchain-write per transaction.

3 SYSTEM DESIGN

In this section, we describe the structure of our privacy protected network model, introduce our notations, and describe the cryptographic primitives we employ in our system.

3.1 Parties

In our system, senders and receivers form a set of users organized into a *ring*. We have intermediaries $I_1, \ldots, I_n$ who act as facilitators for transactions between a sender and receiver. The intermediaries' identities are publicly known to all users in the system, and all users can contact the intermediaries. In Figure 1 we illustrate our system model with n intermediaries and m users. All users in the ring will sign messages using a ring signature scheme [24], to authenticate each of their transactions. All the signatures produced by senders and receivers in the ring will be *anonymous* from the intermediaries' point of view. The only information an intermediary gleans by inspecting a signature is that the signature was produced by a valid user within the ring. We assume that a subset of k intermediaries such that $k < n$, are honest and online all the time, hence we allow for $n - k$ intermediaries to be dishonest. All the intermediaries are connected to each other and they communicate with each other over secure and authenticated channels. We assume that the $I_1, \ldots, I_n$ intermediaries are setup with their traditional digital signature signing and verification keypairs (not ring signatures).
We do not make any trust assumptions on the users in the ring. Any user can be malicious, will try to lie about their transaction amounts, and will try to cheat other honest users.

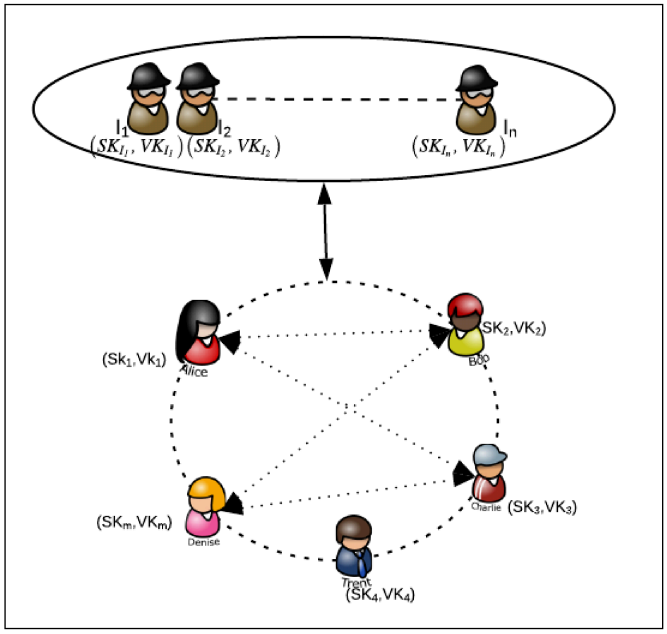

Figure 1. Figure showing system design representing a ring of m users and n intermediaries, $I_1, \ldots, I_n$. The dark arrows represent direct communication, dotted arrows indicate the communication takes place via intermediaries.

3.2 Overview of a transaction

In our system, a sender, Alice picks a value v_i that she wants to transfer to a receiver, Bob, and creates a commitment to v_i: Com_i. She picks a value $t_i \leftarrow \{0,1\}^\lambda$ which will be shared with Bob out-of-band (λ is security parameter). She encrypts t_i and obtains $C_i = E_K(t_i)$, where K is a symmetric key shared with Bob. Next, she signs (Com_i, C_i) using her ring signature signing key. The ring signature scheme guarantees that her identity will be anonymous to all users in the system, including the intermediaries. She then sends the tuple (Com_i, C_i) and the signature over the tuple to k intermediaries. We use the commitment scheme to provide value privacy to Alice. Additionally, since we are modeling a credit network, we have a party called a *gateway*, denoted by *gw* in the system; real-world credit networks such as Ripple have gateway wallets. A gateway is a well-known reputed wallet in the Ripple network that several users can trust to create and maintain a credit link in a correct and consistent manner.
The k-of-n intermediaries will then verify the signature, sign the commitment and post their signed commitment and the original commitment to the blockchain. The receiver Bob, will then do a zero-knowledge proof (ZKP) with the k intermediaries to prove that he knows the token t_i, thus proving he is the right receiver. If the ZKP verifies properly, Bob will be able to claim the v_i coins in the commitment. We give our notations in Table 1.

3.3 Cryptographic Primitives

In this section we give a brief description of the cryptographic primitives that we employ in our system.
Ring signatures: A ring signature scheme [24] provides anonymity to the users within a set called as a *ring*. It consists of two functions RingSign and RingVerify that allows the users to sign and verify using ring signature keys, respectively. All a verifier can tell looking at a ring signature scheme is that some user within the ring has produced the signature. Using ring signatures, we hide the identity of the sender and receiver involved in the transaction, thus providing anonymity. Abe, Ohkubo, and Suzuki, in 2002, developed AOS ring signatures [1] based on the DL assumption, which enabled gaining significant savings in size and verification time for ring signatures. Similarly Greg and Maxwell defined an elliptic curve method as a new way of creating the ring signature which is another efficient solution to save signature size and reduce verification times [19]. Constant Size Ring Signature Without Random Oracle by Bose et al [10], presents a generic constant size ring signatures which is independent of the cardinality of the ring, but uses non-standard assumptions.
Zero knowledge proofs: A zero knowledge proof [9] is a technique by which a prover can prove to a verifier that the prover knows some secret, but without revealing the secret itself. In our system, a receiver, Bob, proves to an intermediary, that he knows the token t_i contained in a given commitment tuple, (Com_i, C_i), without conveying any information apart from the fact that he knows the value of t_i. Hence using ZKPs, the receivers can validate themselves as the right receiver.
Pedersen commitment scheme: A Pedersen commitment scheme [30] allows one to commit to a chosen value, such that the commitments are *perfectly hiding*. The commitment will not leak any information about value (in an information-theoretic sense), while being binding on the sender, i.e., the sender cannot open the commitment to any other value. We use commitments to provide privacy of transaction values from intermediaries.

4 ADVERSARY MODEL

In our system, the adversary can adaptively corrupt a single user or a set of users in the ring. The corrupted user can be either the sender, receiver or the intermediary. The adversary can be either from the

Table 1. List of notations

Variable	Definition
λ	Security parameter
(SK_i, VK_i)	Signing/verification keypair for user i
$I_1, I_2, \ldots, I_n$	Set of n intermediaries
Com_i	Commitment i
t_i	Token i
C_i	Encryption of t_i
K	Symmetric key
v	Value committed to by sender
(PK_i, DK_i)	Encryption/decryption keypair for i^{th} user
σ	Signature
(SK_{gw}, VK_{gw})	Signing/verification keypair for gateway gw
(SK_{I_j}, VK_{I_j})	Signing/verification keypair for intermediary I_j

ring, or any outside user influencing honest users in the ring. Each user i has her own signing and verification key pair (SK_i, VK_i). By corrupting any sender, the adversary takes full control over the user's actions and the user's signing key is compromised by the adversary. Any corrupted sender Alice can promise a specific amount say, v_i to Bob, but provide a commitment with a lesser value in the commitment. In case the adversary corrupts any of the receivers in the ring, the corrupted receiver may have claimed the right token from the k out of n intermediaries, but claim that he did not get the correct value in the commitment. The adversary can also corrupt a set of (possibly colluding) users not involved in the transaction who meddle with the commitments.

We assume a honest majority k among the n intermediaries. The adversary can corrupt up to a maximum of $n - k$ intermediaries. If an intermediary gets corrupted, it can collude with a sender or receiver and forge commitments or acknowledgments respectively. The corrupted intermediary can also modify the commitments made by the sender. We assume that the gw and the set of intermediaries do not collude with each other.

4.1 Privacy and Security properties

The following are the set of security and privacy properties offered by our system.

Sender/receiver privacy: The adversary will not know the identity of the sender or receiver since they cloak their identity among a ring of users and validate their identity using ring signatures.

Unlinkability: Since the identities of the sender and receiver are cloaked among a ring of users, any intermediary or adversary will not be able to link commitments to a sender or receiver in the payment ring. Additionally, no two commitments can be traced back to the same user.

Value Privacy: Any adversary in the ring will not know the value of a given transaction, since the value is secured by a commitment scheme along with a encrypted token, unless the adversary holds the token to claim the value.

Lastly, our system offers one other desirable property:

Scalability: Our system allows users to do micro-payments. So users could transact in fractions of regular cryptocurrency payments (e.g., \$3 = 0.00073 Bitcoin).

5 OUR CONSTRUCTION

Our construction comprises of five phases: setup, ring signatures, commitment, verify, and pay phases. In this section, we describe each of those phases.

5.1 Setup Phase

In this phase, the users in a network and the intermediaries are setup with their respective secret and verification keys. There are m users and n intermediaries in our network. We assume that m users form a *ring*. The ring member who produces the signature is the signer. We assume that each user i in the ring has their own verification key VK_i and secret key SK_i. The general notion of ring signatures does not require any special properties, but assumes that the signer uses trapdoor one-way permutations to generate the ring signatures.

Algorithm 1 shows the steps involved in setting up keys for all the users in ring and signing/verification keys for the intermediaries. In this algorithm, we create ring signature key-pairs $(VK_1, SK_1), (VK_2, SK_2), \ldots., (VK_m, SK_m)$ for all users (Line 2). The signer does not need the knowledge, consent, or assistance of the other ring members to put them in the ring; all she needs is knowledge of their public verification keys. The next part in the setup phase involves creating key pairs for the n intermediaries in the system. The verification and signing keys $(VK_{I_1}, SK_{I_1}), (VK_{I_2}, SK_{I_2}), \ldots, (VK_{I_n}, SK_{I_n})$ are setup for each of the intermediaries (Line 4). The intermediaries' signing/verification key-pairs are those of a traditional digital signature scheme (not ring signatures). Similarly, the gateway gw has a signing/verification key-pair denoted by (VK_{gw}, SK_{gw}) (Line 5).

Algorithm 1: Setup phase

Parties : m users and n intermediaries

1 **for** $i \in [1..m]$ **do**

  /* Create users in ring and setup signing and verifying keys */

2 Each user creates her key-pair (VK_i, SK_i).

3 **end**

4 The intermediaries $I_1 \ldots I_n$ create their key-pairs $(SK_{I_1}, VK_{I_1}), (SK_{I_2}, VK_{I_2}), \ldots., (SK_{I_n}, VK_{I_n})$.

5 gw creates signing and verification keys (SK_{gw}, VK_{gw})

5.2 Ring Signatures

We now describe Algorithm 2. In our construction we use a ring signature scheme [24] to provide sender/receiver anonymity. RingSign denotes the algorithm that a user in the ring invokes to create a ring signature on a message, *msg*. A sender, Alice, picks a random initialization value a or "glue" from $\{0,1\}^\lambda$, and a message, *msg* to be signed. Alice picks a random sk_i for all the other users in the

ring. This is not the original secret key of the other users, rather fake-secret keys created by Alice to cloak her identity among the m users. Alice, then uses her knowledge of trapdoor permutations to invert $g_i(sk_i)$ (Line 4) defined in following way. For any λ-bit input sk_i which is the message, define non-negative integers q_i and r_i, so that $sk_i = q_i n_i + r_i$ and $0 < r_i < n_i$. Then

$$g_i(sk_i) = \begin{cases} q_i n_i + f(r_i), & \text{if } (q_i + 1)\, n_i \leq 2^{\lambda} \\ sk_i, & \text{otherwise.} \end{cases} \quad (1)$$

For λ-bit sk_i define non-negative integers q_i and r_i so that $msg = q_i n_i + r_i$ and $0 \leq r_i < m_i$. She creates a signature $\text{RingSign}(VK_1, VK_2, ..VK_m; a; sk_1, sk_2..., sk_k) \longrightarrow \sigma_i$ in Line 7 of this algorithm.

RingVerify describes the verification process by a verifier i. He hashes the msg which is (Com_i, C_i) and computes the encryption key, $K = H(msg)$. Intermediaries then verify the signature by checking the equation $C_{K,a}(y_1, y_2, .., y_a) \stackrel{?}{=} a$ which is a combining function that takes input K and initialization value a which is used to verify the signature σ_i produced by Alice using the RingSign function. In our system, the intermediaries and the gateway wallet, gw also run the RingVerify function in Algorithm 6. In a transaction where Alice is the sender, the intermediary I_i uses the RingVerify function to verify the ring signature obtained on a msg from Alice in Line 14 of RingVerify.

5.3 Commitment Phase

We now describe the commitment phase outlined in Algorithm 3. The commitment phase consists of the sender and the n intermediaries. k out of n intermediaries are assumed to be available and honest. The sender from the ring of m users creates commitments for values to be sent to k receivers. We set the number of receivers to be the same as the number of honest intermediaries for clarity and ease of presentation, but in practice, they could be different. Let us consider a scenario where a sender Alice chooses values (amounts) $v_1, v_2, \ldots, v_k$ (Line 2) to send to k receivers.

The commitments are created using the Pedersen commitment scheme [30] where Alice chooses randomness $x_1 \ldots x_k$ for all the k receivers (Line 2). Creating commitments helps to preserve value privacy in our system. Every transaction value is protected such that the intermediaries do not get to know any values transacted between any pair of users. Alice then creates tokens $t_1, t_2, ..., t_k$ which are sent out-of-band to every receiver, and each t_i is unique to a single receiver (Line 4). Alice then creates tuples containing (Com_i, C_i) where C_i is the encrypted t_i, and PK_i is the public encryption key of the receiver. The main idea behind creating these tokens is to ensure each valid receiver among the k receivers gets the right commitment created for them. These tuples are shared to the k intermediaries using a secret sharing scheme, e.g., Shamir's secret sharing scheme [28] (Line 7). In a secret sharing scheme, a secret is divided among a group of n members, such that any k out of them can collaboratively re-construct the secret, as given in Algorithm 4.

Alice generates a signature σ_A using $\text{RingSign}(msg_{com}, Vk_1, Vk_2, ..Vk_m; a; sk_1, sk_2..., sk_m)$ function defined in Algorithm 2, where $msg_{com} = (S_1 \ldots S_n)$ (Lines 9, 10). Alice, also creates σ_v to be sent to gw, where σ_v is the signature created on msg using

Algorithm 2: Ring Signatures

```
/* Ring Signatures                                          */
1  Function
     RingSign(msg, VK_1, VK_2, ..VK_m; a; sk_1, sk_2..., sk_k)
       Data: msg, the verification keys, fake-secret keys are
             the input.
       Result: σ_i
2      for i ∈ [1..m] do
3        User i picks picks a ← {0,1}^λ and computes msg to
           be signed;
4        She picks sk_i for all the other ring members
           1 ≤ i ≤ m, i ≠ p (p denotes sender herself) and
           computes y_i = g_i(sk_i)
5        She solves y_i : C_{K,a}(y_1, y_2, .., y_a) = a
6        She computes sk_p = g_p^{-1}(y_p)
7        She creates signature on msg as:
           Sign_{SK_i}(VK_1, VK_2, ..VK_m; a; sk_1, sk_2..., sk_k)
                                                    ⟶ σ_i
8      end
9  end
   /* Verify function                                          */
10 Function RingVerify(msg, σ_i)
       Data: msg, signature is the input to this function
       Result: true or false
11     for i ∈ [1..m] do
12       Verifier computes y_i = g_i(sk_i)
13       He computes K = H(msg)
14       He verifies C_{K,a}(y_1, y_2, .., y_a) ≟ a and does
           Verify_{VK_i(σ_i, msg)} ≟ "true"
15     end
16 end
```

$\text{RingSign}(msg, Vk_1, Vk_2, ..Vk_m; a; sk_1, sk_2..., sk_m)$,
where $msg = (v_i, r_i, t_i)$ (Line 10). Intermediaries $I_1 \ldots I_k$ use the SecRecover() function to retrieve the commitment-tuple pair (Line 14). $I_1 \ldots I_k$ verifies the (Com_i, C_i) and the signature, following which the intermediaries post $\left(\sigma_{Com_j}, Com_j\right)$ to blockchain, where σ_{Com_j} is the signature intermediary j creates after verifying Com_j (Lines 15, 16, 17).

5.4 Zero Knowledge Proof Claim for Bob

In our system, we use zero knowledge proofs (ZKPs) to ensure that a valid receiver authenticates himself to k out of n intermediaries. Specifically, a receiver proves in zero knowledge that he knows the value of the token t_i contained in the sender's tuple: $(Com_i, C_i = E_{PK_i}(t_i))$. We illustrate the ZKP in Algorithm 5. A sender Alice and receiver Bob share a secret t_i through out-of-band communication. The intermediaries then verify if Bob is the valid receiver so that he can claim his amount (payment) from the intermediaries. The ZKP depicted in Algorithm 5 is straightforward, and we do not reiterate the steps here. After confirming that Bob is a valid receiver the intermediaries then sign a transcript of the proof, τ, and send the proof along with the signature to the gateway wallet gw.

Algorithm 3: Commitment phase

Input : Set of senders, Set of receivers, $\mathbb{G}$, $g, h \in \mathbb{G}$, $q = |\mathbb{G}|$
Output : σ_A, σ_v
Parties : Alice and set of n intermediaries

/* Alice creates commitments to amounts and tokens, sends to n intermediaries */

1 **for** $i \in [1..k]$ **do**
2 Alice chooses values $v_i \in \mathbb{Z}_q$ and randomness $x_i \in \mathbb{Z}_q$
3 She creates $Com_i = g^{v_i} h^{x_i}$
4 She creates $C_i = E_{PK_i}(t_i)$ where $t_i \in \{0,1\}^\lambda$
5 She constructs commitment, token tuple (Com_i, C_i)
6 **end**
7 Alice runs
SecShare $((Com_1, C_1), ..., (Com_k, C_k)) \longrightarrow (S_1, S_2, ..S_n)$

/* Alice signs the commitments */

for $i \in [1..k]$ **do**
Alice computes msg_{com} as $K = (S_1, S_2, ..S_n)$
Alice calls function
RingSign $(msg_{com}, VK_1, VK_2, ..VK_m; a; sk_1, .., sk_k) \longrightarrow \sigma_A$
, and sends σ_A to intermediaries.
She computes $msg = (v_i, r_i, t_i)$, calls
RingSign $(msg, VK_1, VK_2, ..VK_m; a; sk_1, sk_2 ..., sk_k) \longrightarrow \sigma_v$
Alice sends σ_v to gw.
end

/* Intermediaries verify the commitments */

8 **for** $i \in [1..k]$ **do**
9 **for** $j \in [1..k]$ **do**
10 I_i runs $(Com_j, C_j) \leftarrow$ SecRecover $(S_1, S_2, ...S_n)$
11 I_i calls RingVerify $(\sigma_A, msg) \stackrel{?}{=}$ "accept"
12 If accept, I_i do $Sign\left(Com_j, C_j\right) \rightarrow \sigma_{Com_j}$ using SK_{I_i}
13 $\left(\sigma_{Com_j}, Com_j\right)$ is posted on blockchain
14 **end**
15 **end**

Algorithm 4: Secret sharing scheme

1 **for** $i \in [1..k]$ **do**
2 SecShare$((Com_1, C_1) \ldots (Com_k, C_k)) \rightarrow (S_1, S_2, \ldots, S_n)$: Splits a secret $(Com_1, C_1) \ldots (Com_k, C_k)$ into n shares, with each $S_i \in \mathbb{Z}_q$.
3 SecRecover$(S_1, S_2, \ldots, S_n)) \rightarrow (Com_1, C_1) \ldots (Com_k, C_k)$: If $S_1, S_2, \ldots, S_n$ are k different shares produced by the SecShare operation, then the value $(Com_1, C_1) \ldots (Com_k, C_k)$ that this produces is the original secret value.
4 **end**

Algorithm 5: Zero Knowledge proof for Bob

Parties : Alice,Bob and n intermediaries

1 **for** $i \in [1..m]$ **do**
2 **for** $j \in [1..n]$ **do**
/* Prove that Bob is a valid receiver of token t */
3 **begin**
4 Bob sends $A \leftarrow g^t \mod q$ to I_j
5 I_j picks $s \leftarrow \mathbb{Z}_q$, sends $Com(s)$ to Bob
6 Bob computes $r_1 \leftarrow \mathbb{Z}_q, y_1 = g^{r_1} \mod q$
7 Bob sends y_1 to I_j
8 I_j sends s to Bob
9 Bob verifies Com, does $z = (ts + r_1) \bmod q$, sends z to I_j
10 I_j verify $g^z \stackrel{?}{=} (A^s y_1) \bmod q$
11 If true, then I_j constructs $Sign_{SK_{I_i}}\left(s, y, A, z, g, q, \mathbb{G}, \sigma_{Com_j}\right) \rightarrow \tau$, sends to gw
12 **end**
13 **end**
14 **end**

5.5 Pay Phase

Algorithm 6 depicts the pay phase in which the gateway wallet, gw releases the amount to the receiver, after the receiver has authenticated himself to the k intermediaries. The gateway gw has its own signing and verification keys $\left(SK_{gw}, VK_{gw}\right)$, that were setup in Algorithm 1. Figure 2 shows the steps involved in gw releasing an amount v_i to a receiver, Bob's wallet. The numbers in the figure depict the order in which verification is carried out by gw.
In Algorithm 6, Bob signs $(msg_B, VK_1, VK_2, VK_m; a; sk_1, \ldots, sk_k) \rightarrow \sigma_B$ sends σ_B to gw, where msg_B is as defined in Line 2. Then, the gw verifies if (σ_v, msg) provided by Alice to gw in Line 10 of Algorithm 3 is valid. If yes, the gw verifies the ZKP, τ, and the signature of each intermediary, $\sigma_{Com_j}; j \in [1..k]$ on τ (Lines 6, 7, 8). If both are valid, then the gw verifies if the Com_i provided by Bob, matches with the commitment that is present in the proof in Line 11 of Algorithm 5. If all the verification pass, gw releases the value v_i to Bob's wallet (Line 10). In a scenario where any of the verification fail, gw cancels the transaction. It would be a very interesting idea to explore implementing this transaction using smart contracts, which we envision as a part of our future work.

6 IMPLEMENTATION AND EVALUATION

We have implemented our system using Solidity [7], and used the Charm cryptographic library [2] for implementing the Pedersen commitments and ZKPs[1]. Our experiments were run on a desktop class computer with Intel(R) Core(TM) i3-7100 CPU @ 3.90GHzx4 and 8GB RAM on ubuntu-16.04 platform. The participants in our system are instantiated by an Ethereum instance with 100 Ethers credited by default to each participant. The most efficient mixer

[1] https://github.com/sigcrypto/privacy

Algorithm 6: Pay Phase

Input : $G, g \in G, q = |\mathbb{G}|, Com_i, C_i, v_i$
Output : v_i
Parties : Alice, Bob, k intermediaries, gw

for $i \in [1..k]$ **do**
  /* Pay v_i amount to Bob */
  Bob computes $msg_B = ((s, y, A, z, g, q, \mathbb{G}, \sigma_{Com_j}), t_i)$, calls RingSign$(msg_B, VK_1, VK_2, VK_m; a; sk_1, .., sk_k) \rightarrow \sigma_B$
  Bob sends σ_B to gw
  gw calls RingVerify $(\sigma_v, msg) \stackrel{?}{=} \{true, false\}$
  if *true* **then**
    gw does $Verify_{VK_{I_i}}\left(\tau, \left(s, y, A, z, g, q, \mathbb{G}, \sigma_{Com_j}\right)\right) \stackrel{?}{=} \{true, false\}$
    if *true* **then**
      gw calls RingVerify $(\sigma_B, msg_{Com}) \stackrel{?}{=} \{true, false\}$
      if *true* **then**
        gw verifies if $Com_i = (g^{v_i}, h^{r_i})$, then releases v_i to Bob's wallet
      end
    end
  end
  else
    gw cancels transaction.
  end
end

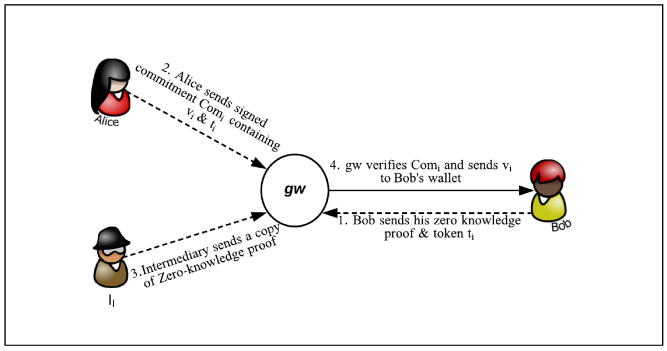

Figure 2. Illustration of final pay phase. The dotted arrows represent the input to gw. Solid arrow represent flow of credit from gw to Bob's wallet. The numbers represent the sequence in which the execution of transaction takes place.

network for Bitcoin supports up to a max. of 800 users [11]. CoinJoin and Coin shuffle can support upto 50 and 538 users per mix respectively, whereas our system can support more users than any of them. The size of a ring signature is proportional to the number of users in the ring, as shown in Table 2. A ring signature ensures a signer can sign any message anonymously on behalf of a group. Unfortunately, the size of the signature grows linearly with the size of the ring, and becomes inefficient when the ring size is large. To mitigate this, it would be worthwhile to explore ring signature schemes which offer constant-sized signatures such as [10]. Although such signature schemes exist, they are proven secure in the random oracle model and use non-standard assumptions, whereas the schemes we use are based on the well-known RSA and discrete logarithm assumptions. Table 3 shows the expenses in terms of

Table 2. Ring signature timings

Number of users in ring	Time taken to sign (sec)	Time taken to verify (sec)	Size of signature (bytes)
100	4.222	8.940	2980
200	8.464	17.754	5945
400	18.324	34.393	11886
800	33.427	68.505	23757

Gas , ether, and equivalent USD for deploying the cryptographic primitives. We used altbn128 elliptic curves in the implementation of the ZKP, and implemented AOS ring signatures [19].

We also measured the time taken for creating and verifying a

Table 3. Deployment costs

Functionalities	Gas	ETH	USD
Contract Migration	277398	0.00554796	0.832
altbn128 elliptic curves	74748	0.00149496	0.224
ZKP	74684	0.00149368	0.224
AOS ring signatures	438206	0.00876412	1.315
Total	865036	0.01730072	2.595

ZKP and Pedersen commitment; the timings are given in Table 4. The first column shows the average time taken to create a ZKP proof and to create a Pedersen commitment. The second column shows the average time taken to verify a ZKP or a commitment. We recollect that in our system, we only need a honest majority k, of n intermediaries. In our experiments, we set $k = 7$, and $n = 10$, similar to [15], although one could vary k, with some tradeoffs. Choosing a smaller k would require users to place more trust on the intermediaries, whereas a higher k value would mae the system incur more computational costs.

Table 4. Time for commitments and ZKPs

	Time for creation (sec)	Time for Verification (sec)
Zero Knowledge Proof	0.00189	0.05948
Pedersen Commitment	0.0267	0.0267

We calculate the time taken for a transaction to go through in our system using Equation 2. This equation specifies the total time taken for cryptographic operations involved. The time taken for

Table 5. Max. users in ring & corresponding end-to-end transaction time

Time taken for a transaction	Number of users in the Ring
4 Seconds [Ripple]	10 users
120 seconds [Ethereum]	400 users
1200 seconds [Bitcoin]	3200 users

ring signatures, ZKP and commitments are specified in Table 2 and Table 4. δ denotes the overhead time, which includes a standard sign and verify operation by the intermediaries and other network latencies in the system, which we assume would be lesser than the cost of the major cryptographic operations.

$$\text{Transaction time} = k(\text{Time taken for commitment}) + 3(\text{Ring Sign}) + 3(\text{Ring Verify}) + k(\text{Time taken for ZKP}) + \delta \quad (2)$$

Using the timing values provided in Table 3 and Table 4 in Equation 2, we calculate the number of users in the ring with transaction confirmation time for Ripple (4 Sec), Ethereum (2+ min) and Bitcoin (20 min) [23], as reference, and the values are given in Table 5. It is important to note that the time mentioned for these cryptocurrencies are just *transaction confirmation times*, and the actual *end-to-end transaction time* is a lot more, since it depends on block creation for each of these transactions [3]. Whereas in our system, the corresponding number of users in the ring as mentioned in Table 5 could do an *end-to-end transaction* in the times given in first column of Table 5, with total privacy and anonymity.

7 CONCLUSION AND FUTURE WORK

In this paper, we have proposed techniques for providing user and value privacy, as well as transaction integrity and security, while mixing transactions in a credit network. Our experiments show that our system can be deployed on a reasonably large networks, and is comparable to mixer networks proposed for cryptocurrencies [11]. Although we envision our system to be of use in credit networks, one can apply it to other payment networks too. In future work, we plan to analyze the security and privacy properties of our system in a formal model such as the Universal Composability framework of Canetti [6], and prove its security. We also plan to study the interesting problem of constructing *virtual* payment channels [8] for credit networks.

ACKNOWLEDGMENTS

Research supported by NSF award #1800088. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

REFERENCES

[1] Abe, M., Ohkubo, M., and Suzuki, K. 1-out-of-n signatures from a variety of keys. *IEICE Transactions 87-A*, 1 (2004), 131–140.
[2] Akinyele, J. A., Garman, C., Miers, I., Pagano, M. W., Rushanan, M., Green, M., and Rubin, A. D. Charm: a framework for rapidly prototyping cryptosystems. *J. Cryptographic Engineering 3*, 2 (2013), 111–128.
[3] Coinbase. https://www.coinbase.com/.
[4] Bonneau, J., Narayanan, A., Miller, A., Clark, J., Kroll, J. A., and Felten, E. W. Mixcoin: Anonymity for bitcoin with accountable mixes. *IACR Cryptology ePrint Archive 2014* (2014), 77.
[5] Bulck, J. V., Minkin, M., Weisse, O., Genkin, D., Kasikci, B., Piessens, F., Silberstein, M., Wenisch, T. F., Yarom, Y., and Strackx, R. Foreshadow: Extracting the keys to the intel SGX kingdom with transient out-of-order execution. In *USENIX Security Symposium* (2018), USENIX Association, pp. 991–1008.
[6] Canetti, R. Universally composable security: A new paradigm for cryptographic protocols. *IACR Cryptology ePrint Archive 2000* (2000), 67.
[7] Dannen, C. Bridging the Blockchain Knowledge Gap. In *Introducing Ethereum and Solidity: Foundations of Cryptocurrency and Blockchain Programming for Beginners*, C. Dannen, Ed. Apress, Berkeley, CA, 2017, pp. 1–20.
[8] Dziembowski, S., Eckey, L., Faust, S., and Malinowski, D. Perun: Virtual payment channels over cryptographic currencies. Tech. rep., IACR Cryptology ePrint Archive, 2017: 635, 2017.
[9] Feigenbaum, J. Overview of interactive proof systems and zero-knowledge. *Contemporary Cryptology: The Science of Information Integrity* (1992), 423–439.
[10] Gu, K., and Wu, N. Constant size traceable ring signature scheme without random oracles. *IACR Cryptology ePrint Archive 2018* (2018), 288.
[11] Heilman, E., Baldimtsi, F., Alshenibr, L., Scafuro, A., and Goldberg, S. Tumblebit: An untrusted tumbler for bitcoin-compatible anonymous payments. *IACR Cryptology ePrint Archive 2016* (2016), 575.
[12] Khalil, R., and Gervais, A. Revive: Rebalancing off-blockchain payment networks. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security* (2017), ACM, pp. 439–453.
[13] Khalil, R., and Gervais, A. Nocust–a non-custodial 2 nd-layer financial intermediary. Tech. rep., Cryptology ePrint Archive, Report 2018/642. https://eprint.iacr. org/2018/642, 2018.
[14] Lind, J., Eyal, I., Kelbert, F., Naor, O., Pietzuch, P., and Sirer, E. G. Teechain: Scalable blockchain payments using trusted execution environments. *arXiv preprint arXiv:1707.05454* (2017).
[15] Malavolta, G., Moreno-Sanchez, P., Kate, A., and Maffei, M. Silentwhispers: Enforcing security and privacy in decentralized credit networks. *IACR Cryptology ePrint Archive 2016* (2016), 1054.
[16] Malavolta, G., Moreno-Sanchez, P., Kate, A., Maffei, M., and Ravi, S. Concurrency and privacy with payment-channel networks. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security* (2017), ACM, pp. 455–471.
[17] Maxwell, G. Coinjoin: Bitcoin privacy for the real world. In *Post on Bitcoin forum* (2013).
[18] Maxwell, G. Coinswap: Transaction graph disjoint trustless trading. *CoinSwap: Transactiongraphdisjointtrustlesstrading (October 2013)* (2013).
[19] Maxwell, G., and Poelstra, A. Borromean ring signatures, 2015.
[20] Moreno-Sanchez, P., Ruffing, T., and Kate, A. Pathshuffle: Credit mixing and anonymous payments for ripple. *Proceedings on Privacy Enhancing Technologies 2017*, 3 (2017), 110–129.
[21] Panwar, G., Misra, S., and Vishwanathan, R. Blanc: Blockchain-based anonymous and decentralized credit networks. *IACR Cryptology ePrint Archive 2019* (2019), 14.
[22] Poon, J., and Dryja, T. The bitcoin lightning network: Scalable off-chain instant payments. *See https://lightning. network/lightning-network-paper. pdf* (2016).
[23] Ripple website. www.ripple.com, 2015.
[24] Rivest, R. L., Shamir, A., and Tauman, Y. How to leak a secret. In *ASIACRYPT* (2001), vol. 2248 of *Lecture Notes in Computer Science*, Springer, pp. 552–565.
[25] Roos, S., Moreno-Sanchez, P., Kate, A., and Goldberg, I. Settling payments fast and private: Efficient decentralized routing for path-based transactions. *CoRR* (2017).
[26] Ruffing, T., Moreno-Sanchez, P., and Kate, A. Coinshuffle: Practical decentralized coin mixing for bitcoin. In *ESORICS (2)* (2014), vol. 8713 of *Lecture Notes in Computer Science*, Springer, pp. 345–364.
[27] Ruffing, T., Moreno-Sanchez, P., and Kate, A. P2P mixing and unlinkable bitcoin transactions. In *24th Annual Network and Distributed System Security Symposium, NDSS 2017, San Diego, California, USA, February 26 - March 1, 2017* (2017), The Internet Society.
[28] Shamir, A. How to share a secret. *Commun. ACM 22*, 11 (1979), 612–613.
[29] Stellar website. https://www.stellar.org/.
[30] Tang, C., Pei, D., Liu, Z., and He, Y. Non-interactive and information-theoretic secure publicly verifiable secret sharing. *IACR Cryptology ePrint Archive 2004* (2004), 201.