Pedro Moreno-Sanchez*, Tim Ruffing, and Aniket Kate

PathShuffle: Credit Mixing and Anonymous Payments for Ripple

Abstract: The *I owe you* (IOU) credit network Ripple is one of the most prominent alternatives in the burgeoning field of decentralized payment systems. Ripple's path-based transactions set it apart from cryptocurrencies such as Bitcoin. Its pseudonymous nature, while still maintaining some regulatory capabilities, has motivated several financial institutions across the world to use Ripple for processing their daily transactions. Nevertheless, with its public ledger, a credit network such as Ripple is no different from a cryptocurrency in terms of weak privacy; recent demonstrative deanonymization attacks raise important concerns regarding the privacy of the Ripple users and their transactions. However, unlike for cryptocurrencies, there is no known privacy solution compatible with the existing credit networks such as Ripple.

In this paper, we present PathShuffle, the first path mixing protocol for credit networks. PathShuffle is fully compatible with the current credit networks. As its essential building block, we propose PathJoin, a novel protocol to perform atomic transactions in credit networks. Using PathJoin and the P2P mixing protocol DiceMix, PathShuffle is a decentralized solution for anonymizing path-based transactions. We demonstrate the practicality of PathShuffle by performing path mixing in Ripple.

Keywords: Ripple, Stellar, *IOweYou* Settlements, Credit Networks, Privacy, P2P Mixing

DOI 10.1515/popets-2017-0031
Received 2016-11-30; revised 2017-03-15; accepted 2017-03-16.

1 Introduction

Decentralized cryptocurrencies such as Bitcoin [46] and Ethereum [28] as well as credit networks such as Ripple [11] and Stellar [12] provide financial settlement solutions that are revolutionizing the finance industry globally. Their use of pseudonymous identities and transactions, their ability to settle transactions worldwide at a small consistent fee, and their potential to monetize everything regardless of jurisdiction [47] have been pivotal to their success.

Credit Networks. In a credit network [27, 29, 34] such as Ripple [11] and Stellar [12], users extend trust to others in terms of *I Owe You* (IOU) credit. This enables transactions between two connected users by appropriately settling IOU credit across the trust path connecting them. From a practical perspective, this endows credit networks with a unique capability of performing same and cross-currency settlement transactions between fiat currencies, cryptocurrencies and even user-defined currencies at a very low cost in few seconds [24].

Ripple has gained unprecedented traction over the last years and several financial institutions worldwide are adopting Ripple in their transaction backbone [5, 10, 26, 32, 37, 50, 56]. Ripple, however, is not limited to banks or fiat currencies. Ripple supports cross-currency transactions where the payer and payee specify the amount of IOU to be transacted in their own preferred currency, including cryptocurrencies such as Bitcoin. Therefore, all merchants accepting Bitcoin payments can now accept payments through Ripple [7]. Moreover, Ripple supports user-defined currencies, such as Goodwill [47], a currency traded in exchange for socially valuable services (e.g., an interesting forum post). Currently, the Ripple network caters to over 200 thousand user wallets and serves a daily transaction volume over $1 million [8].

Besides Ripple, Stellar is a second example of a credit network that has been deployed in practice. However, Stellar is still at an early stage and has not been widely adopted yet [13]. For the sake of concreteness, we focus on the Ripple network in this work, but we note that all our techniques apply to credit networks in general, including Stellar.

Privacy Challenges and Related Work. While the Ripple network offers a multitude of benefits and capabilities to the financial industry, the public nature of its transaction ledger exposes its individual users, groups, organizations, and companies to the same severe privacy

***Corresponding Author: Pedro Moreno-Sanchez:** Purdue University, E-mail: pmorenos@purdue.edu
Tim Ruffing: Saarland University, E-mail: tim.ruffing@mmci.uni-saarland.de
Aniket Kate: Purdue University, E-mail: aniket@purdue.edu

BY-NC-ND
Unauthenticated
Download Date | 2/21/18 6:24 PM

attacks as already observed in Bitcoin [15, 17, 36, 40, 41, 48, 57]. A recent study [45] makes this privacy concern justifiable by showing that a significant portion of Ripple transactions today can be easily deanonymized such that everybody can determine who paid what to whom.

Moreno-Sanchez et al. [44] show the first solution that prevents deanonymization in credit networks. Their solution leverages trusted hardware to enforce strong privacy guarantees by accessing the credit network by means of a data-oblivious algorithm hiding the access pattern. Although this solution provides strong privacy guarantees, it is not compatible with the current trust philosophy of Ripple.

Recently, Malavolta et al. [38] propose an alternative solution to the same problem by completely avoiding the ledger. Instead, every user logs the credit changes on her own credit links and transactions are jointly performed by users in the path from sender to receiver. Although this approach also enforces strong privacy guarantees, a setting without ledger breaks compatibility with the current ledger-based Ripple network.

In the realm of Bitcoin and other cryptocurrencies, several solutions [17, 18, 20, 21, 30, 31, 35, 39, 42, 51, 52, 61–64] have been proposed to overcome similar privacy issues: These solutions (e.g., Zerocoin [42] and Zerocash [18]) are tailored to the specifics of blockchain-based cryptocurrencies. Given the fundamental differences between credit networks and cryptocurrencies (see Section 2.1), it remains an interesting future work to study whether it is feasible to adapt the underlying ideas of these strong solutions to credit networks.

For example, it is conceivable that simple centralized mixing protocols such as Mixcoin [21] and Blindcoin [61], which do not rely on smart contracts, can be adapted to Ripple with non-trivial modifications. In these solutions, the mixing server can steal coins from the users, although such theft is accountable. In this work, instead, we instead strive for a solution where no theft is possible in the first place, and all existing theft-resistant mixing protocols for cryptocurrencies either rely on multi-input-multi-output transactions [39, 51, 52] or on script-based smart contracts [30, 31, 35, 64], none of which are supported in credit networks such as Ripple. Therefore, none of the privacy-enhancing technologies proposed for cryptocurrencies are directly applicable to path-based transactions in the Ripple network.

Contributions. We present PathShuffle, the first mixing protocol for path-based transactions in credit networks. Our contribution consists of the following parts.

- We introduce *path mixing*, our approach for anonymous transactions in credit networks. Our key observation is that IOU transaction paths that share a common node can be mixed.
- We propose PathJoin, a protocol to perform multi-input-multi-output transactions, which enables that n users transfer credit *atomically* from their input to their output wallets, thereby solving a standard fairness problem in mixing. PathJoin combines functionality available in Ripple and a distributed signature scheme. Atomic multi-input-multi-output transactions are interesting on their own for other applications, e.g., crowdfunding.
- We propose PathShuffle, the first decentralized path mixing protocol for credit networks. PathShuffle combines the DiceMix [52] P2P (message) mixing protocol with PathJoin, our novel protocol to perform atomic transactions in Ripple. In doing so, PathShuffle is asymptotically as efficient as the most efficient P2P Bitcoin mixing protocol in the literature [52]: it requires five rounds to perform settlement transactions anonymously over intersecting paths independently on the number of users participating in the path mixing, and $5+3f$ rounds in the presence of f disrupting users.
- Finally, we demonstrate with our proof-of-concept implementation that PathShuffle is fully compatible with Ripple. In particular, we have successfully carried out a mixing transaction in the real Ripple network. Moreover, PathShuffle can be generally applicable to other credit networks such as Stellar.

Organization. The rest of the paper is organized as follows. Section 2 gives the required background on credit networks, their anonymity issues, and multi-input-multi-output transactions. Section 3 defines path mixing and its desired properties. Section 4 describes the key ideas underlying our solution. Section 5 describes PathJoin and Section 6 gives the details of PathShuffle. Finally, Section 7 concludes this work.

2 Background

2.1 Ripple as a Credit Network

Credit Networks. A credit network is a weighted, directed graph $\mathbb{G} = (\mathbb{V}, \mathbb{E})$, where $\mathbb{V}$ is a set of wallets (user accounts) and $\mathbb{E}$ is a set of I Owe You (IOU) credit links between wallets. A credit link $(u_1, u_2) \in \mathbb{E}$ is la-

Unauthenticated
Download Date | 2/21/18 6:24 PM

beled with a dynamic non-negative scalar value α_{u_1,u_2} denoting the amount of *unconsumed* credit that u_1 has extended to u_2 (i.e., u_1 owes α_{u_1,u_2} to u_2). The available credit on an edge is lower-bounded by 0. Moreover, every credit link can additionally have an upper bound if adopted by the wallet owner.

A credit network is equipped with four operations: *chgLink* modifies the amount of IOU in a credit link; *testLink* allows to check the credit available in a single link; *pay* allows to transfer IOU between two wallets *only* across credit paths connecting those two wallets; *test* checks the available IOU along credit paths connecting two wallets.

A wallet can be used as an intermediate hop in a credit path to forward IOU from the incoming link to the outgoing link. This operation is called *rippling* (see [14] for details). In case a wallet wants to avoid unexpected (or even malicious) credit balances shifts, rippling can be deactivated. We use such feature of credit networks in one of our protocols.

The Ripple Network. The Ripple network is an instantiation of a generic credit network as defined above.

As in cryptocurrencies, a wallet in Ripple is associated with a verification key and its corresponding signing key. The wallet is then labeled with an encoding of the hashed verification key. A Ripple transaction contains a single sender and a single receiver. A transaction is valid when signed using the sender's signing key.

Assume that u_1 wants to transfer β IOU to u_n and that u_1 and u_n are connected through a path of the form $u_1 - \ldots - u_i - \ldots - u_n$. In the path finding algorithm, links are considered as undirected. However, the transaction is performed by updating the credit on each link depending on its direction as follows: Links in the direction from u_1 to u_n are increased by β, while reverse links are decreased by β. A transaction is successful if no link is reduced to a value less than 0 and no link exceeds the pre-defined upper bound on the link (if other than ∞). A transaction can be split among several paths such that the sum of credit available on all paths is at least β. Such a transaction contains one sender and one receiver but several paths from sender to receiver, along with the amount of IOU to be transferred in each path. We refer to [16, 45] for a more detailed description.

Ripple transactions are collected by Ripple validators. These publicly known validators run the Ripple Consensus Algorithm [54] to agree on the set of valid transactions. A transaction accepted by a majority of the validators is then applied in the Ripple network.

A new wallet in the Ripple network needs to receive IOU on a credit link to interact with other wallets. The Ripple network solves this *bootstrapping* problem by introducing gateways. A gateway is a well-known reputed service with a wallet in the Ripple network that several wallets can trust to create and maintain a credit link in a correct and consistent manner. To bootstrap, a user can send funds to the gateway (outside of the Ripple network) and the wallet of the gateway will extend credit to the wallet of the user (in the Ripple network). As gateways wallets are highly connected nodes, the thereby created credit link allows the new wallet to interact with the rest of the Ripple network.

Comparison with Cryptocurrencies. Bitcoin [46] is a decentralized cryptocurrency that supports online decentralized payments for the first time. Following the Bitcoin philosophy, several competitor cryptocurrencies have been created [1]. As Bitcoin and other cryptocurrencies, the Ripple network is a ledger-based consensus system that allows to transfer funds between different wallets. However, there are conceptual differences between cryptocurrencies and the Ripple network.

Most importantly, Ripple is not a currency by itself; it is a system that allows users to perform transactions in several existing currencies, using IOU relations. While cryptocurrencies such as Bitcoin allow to exchange funds between any two wallets in the system, a transaction in the Ripple network requires the existence of a path with enough credit between the sender and the receiver wallets.[1]

Additionally, Bitcoin supports a payment with multiple senders and receivers and has a built-in script language. Ripple, instead, does not define any script language and transactions are limited to a single sender and a single receiver.

2.2 Deanonymization Attacks in the Ripple Network

Recently, Moreno-Sanchez et al. [45] showed that the public nature of the Ripple ledger can be used to defeat the anonymity supposedly provided by the pseudony-

1 Technically, Ripple has its own native currency (XRP) to support direct payments without a credit path between two wallets. This currency is used to organize transactions fees for non-XRP transactions and other fees used to prevent DoS attacks in the Ripple network. In the following, we focus on path-based transactions and discuss direct XRP payments in Appendix A.

Unauthenticated
Download Date | 2/21/18 6:24 PM

mous identities in the Ripple network. In the following, we present a brief overview of the two proposed heuristics to identify wallets that belong to the same user.

First, the authors studied the interaction of users with online currency exchanges in order to deposit (or withdraw) cryptocurrencies to (or from) the Ripple network. In a deposit operation, a user pays to an online exchange a certain amount of bitcoins in the form of a Bitcoin transaction, and the online exchange issues the corresponding Bitcoin IOU to the user in the form of a Ripple transaction. The authors showed that it is possible to link Bitcoin and Ripple wallets belonging to the same user by examining such interactions in the publicly available Bitcoin and Ripple ledgers: The sender wallet in the Bitcoin transaction and the receiver wallet in the Ripple transaction belong to the same user, and the remaining two wallets belong to the online exchange.

Second, they studied the interactions among wallets in the Ripple network that implement the hot-cold wallet mechanism: A cold wallet is used as a reserve of IOU (with signing keys securely stored on offline hardware) while a hot wallet is used to perform the daily transactions (with signing keys in memory). Once the hot wallet runs out of IOU, the cold wallet is used to top it off. The authors observed that a cold wallet can be identified by examining the network topology as it only has outgoing links, while the associated hot wallets can then be identified by examining the transaction patterns: The cold wallet only sends IOU to hot wallets. Therefore, this technique can be used to link wallets belonging to the same user by examining the correlation between transactions and the credit network topology.

The results of carrying out the two aforementioned heuristics can be leveraged to deanonymize the user owning a set of Ripple (and possibly cryptocurrency) wallets. Overall, these results have attracted the attention of the Ripple community [3]. Towards mitigating this privacy breach, we observe that the heuristics rely primarily on the fact that the attacker can easily link the sender and the receiver of a Ripple transaction. In this work, we provide a protocol that breaks this linkability. This clearly reduces the applicability of the above heuristics; nevertheless, it is interesting to study the effectiveness of our protocol empirically as Ripple users start to use it.

2.3 CoinJoin: Coin Mixing in Bitcoin

CoinJoin [39] is a method in Bitcoin to join several transactions into a single one: Instead of having n transactions, each one transferring funds from an account in_i to an account out_i, all these transfers are combined in one single transaction.

The key feature of a CoinJoin transaction is the atomicity: Either all transfers happen or none of them happens. This *all-or-nothing* property is crucial in several approaches for coin mixing [51, 52], because it guarantees *fairness*, which can be illustrated by a simple example: Say Alice and Bob would like to mix one bitcoin to gain some anonymity; Alice is supposed to send her bitcoin to a freshly generated address of Bob and vice versa. The obvious question is "who sends first?", because the receiver of the first transaction may just keep the money and refuse to send.

In Bitcoin (and similar cryptocurrencies), CoinJoin is a very natural idea, because a transaction can have several inputs and outputs. However, none of the currently deployed credit networks offer a similar functionality. In order to build such functionality, we observe that anonymous transactions in Ripple can be achieved by *mixing the paths* used in a set of transactions.

3 Path Mixing in Credit Networks

In this section, we first present *path mixing*, our approach to improve anonymity in credit networks. We then describe the communication model, the security and privacy goals, and the threat model.

3.1 Path Mixing

Assume that each user has a pair of wallets, that we denote by *input* and *output* wallets. Furthermore, assume that users participating in the path mixing protocol have agreed beforehand on mixing β IOU.

Functionality. In this setting, a path mixing protocol transfers β IOU from every input wallet to every output wallet so that an adversary controlling the network and some of the participating users cannot determine the pair of input and output wallets belonging to an honest user. We denote this as a *successful* path mixing. Otherwise, no IOU must be transferred from any input wallet and the path mixing is *unsuccessful*.

Compatibility. The path mixing protocol must *only* require functionality already available in credit networks.

Unauthenticated
Download Date | 2/21/18 6:24 PM

3.2 Setup and Communication Model

We assume that users communicate to each other through a bulletin board. Additionally, we assume the bounded synchronous communication setting, where time is divided in fixed epochs: Messages broadcast by a user are available to all other users within the epoch and absence of a message from a user in an epoch indicates that the user is offline.

This bulletin board can seamlessly be deployed in practice using already deployed Internet Relay Chat (IRC) servers with appropriate extensions (see [52] for details). The bulletin board can be alternatively implemented by a reliable broadcast protocol [25, 58] at an increased communication cost.

We assume that users participating in the path mixing protocol have a verification/signing key pair (e.g., key pair for the input wallet). Moreover, we assume that each user knows other users' public verification keys and that all users have agreed on mixing a fixed amount β IOU prior to start executing the path mixing protocol.

Finally, we assume that there is a bootstrapping mechanism in place for users to know other users willing to carry out the path mixing protocol. A malicious bootstrapping mechanism could hinder the anonymity of an honest user by peering him with other users under the attacker's control. Although this is an important threat in practice, we consider it orthogonal to our work. Note that the fees needed to carry out the path mixing limit the number of mixings that the attacker can join.

In practice, we envision that the bulletin board enabling the communication between users also offers a service for users to register. The users could be then grouped together to carry out the path mixing protocol following a transparent mechanism (e.g., based on public randomness). Nevertheless, since it is an orthogonal problem, any bootstrapping mechanism with the desired properties could be used in our work.

3.3 Security and Privacy Goals

Unlinkability. If the path mixing is successful, it should not be possible for the attacker to determine which output wallet belongs to which honest user.

Correct Balance. No matter whether the path mixing is successful, the total credit available to a user should not change (except for possible transaction fees).

Termination. If the bulletin board is honest, the path mixing terminates successfully for all honest users.

3.4 Threat Model

We assume that the attacker controls an arbitrary number f of users participating in the path mixing protocol.

For unlinkability and correct balance we assume that the attacker also controls the bulletin board (and thus the network). The *anonymity set* of an honest user is the set of all honest users. Thus, in order to achieve any meaningful anonymity guarantee, we need that $f < n - 1$. In other words, we do not consider the $n - 1$ attack [55] in this work. Finally, for termination, we assume that the bulletin board is honest. Note that termination as a *liveness* property is not achievable against a malicious bulletin board which can just block all network traffic.

4 Towards a Path Mixing Solution

In this section we first show a straw man approach for path mixing to illustrate the challenges we have to overcome. Then, we overview our approach, a decentralized path mixing protocol.

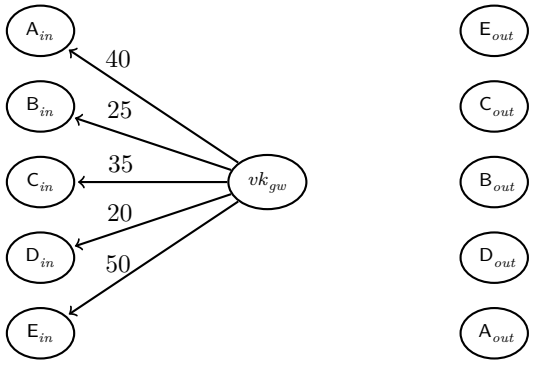
A Straw Man Path Mixing Approach. Path mixing can be achieved following a straw man approach as shown in Fig. 1. Assume that all users participating in the path mixing trust a third-party server to carry out the required operations on their behalf. Further assume that the server is a gateway in the Ripple network and that there exists a path from every input wallet to the gateway's wallet with a capacity of at least β IOU.

In this setting, first every user can send her output wallet to the gateway using an authenticated, private channel (e.g., TLS). An example of the protocol at this step is shown in Fig. 1a. Second, every user can transfer β IOU in the Ripple network from her input wallet to the gateway's wallet. Finally, the gateway, working as a mixing proxy, creates a credit link from each output wallet to the gateway's wallet with a credit upper limit of β IOU. In this manner, now every user can perform a transaction for up to β IOU using the gateway's wallet as the first hop in the transaction path (see Fig. 1b).

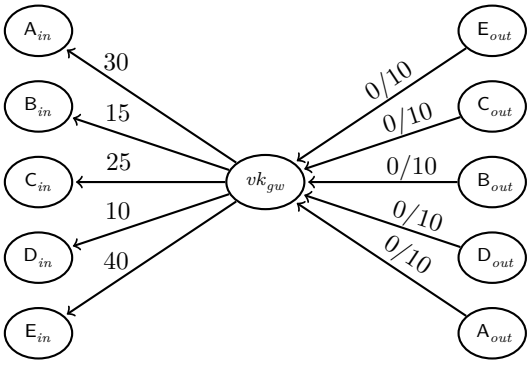
For every user i, the gateway must create a credit link from the output wallet $\text{VK}_{out}[i]$ to its own wallet of the form $vk_{gw} \leftarrow \text{VK}_{out}[i]$ (i.e., $\text{VK}_{out}[i]$ owes credit to vk_{gw}) to ensure unlinkability against an attacker observing the communication and the Ripple ledger.

To see that, assume for a moment that the gateway creates the credit link of the form $vk_{gw} \rightarrow \text{VK}_{out}[i]$. Such operation must be confirmed with a signature by

Unauthenticated
Download Date | 2/21/18 6:24 PM



(a) Credit network before the transactions are carried out in the straw man approach



(b) Credit network after carrying out the straw man path mixing

Fig. 1. An illustrative example of the straw man approach for path mixing to mix 10 IOU among five users. Solid arrows depict credit links between two wallets. Single values on edges denote the current balance and no upper limit. Values a/b on the links denote: a current balance and b upper limit. After finishing the straw man protocol, user A can perform a transaction for up to 10 IOU using A_{out} and vk_{gw} as the first hops in the transaction path.

the user i (see Section 5). Now, user i must submit the signed operation to the Ripple network. If a network attacker associates the signed message to the IP address of user i, he directly learns that $\text{VK}_{out}[i]$ belongs to user i. As the attacker also knows the input wallet belonging to user i, he trivially breaks the unlinkability property.

Drawbacks. In this straw man approach, the server is trusted for unlinkability and correct balance properties. First, the server must be trusted not to reveal the pair of input and output wallets belonging to a user. Second, after receiving the credit from the users' input wallets, the server is trusted not to steal it and instead create the credit link with the output wallets and set up the correct credit upper limit in each credit link.

Decentralized Path Mixing. We overcome the aforementioned drawbacks by designing a decentralized path mixing protocol, where the users jointly transfer credit from their input wallets to their output wallets without requiring *any* third-party mixing proxy. For that, the decentralized path mixing protocol must provide the two main functionalities provided by the trusted server in the straw man approach (see Fig. 2): *Atomic transactions* and *creating a set of output wallets in an anonymous manner*. In the following, we give an overview for each of the functionalities.

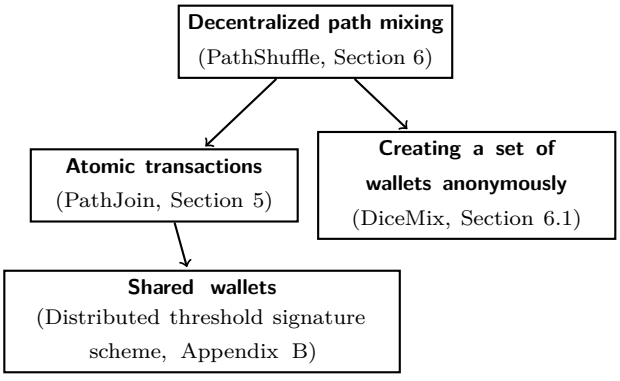


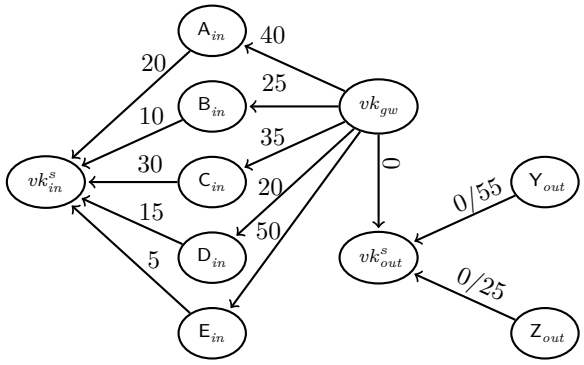
Fig. 2. Schema for a decentralized path mixing protocol. Boxes represent the functionalities denoted by the name in bold. Each functionality is implemented by the protocol in parentheses and it is described in the annotated section. An arrow $a \to b$ denotes that protocol a depends on b.

4.1 Atomic Transactions in Ripple

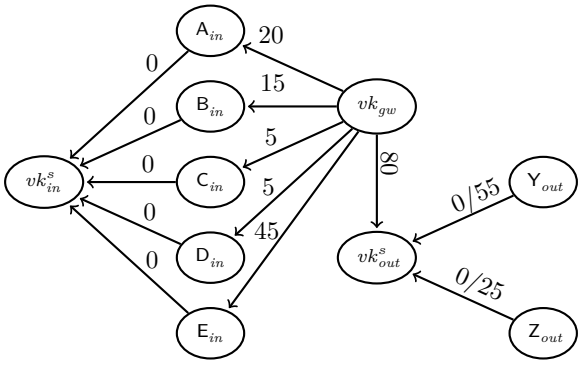
Assume a generic setting with a set of n input wallets $\text{VK}_{in}[\,]$ and a set of m output wallets $\text{VK}_{out}[\,]$. Moreover, assume that instead of a fixed amount of credit β, each input wallet must transfer $\beta_{in}[i]$ IOU and each output wallet must receive $\beta_{out}[j]$ IOU. Although the sets of input and output wallets might not be of the same size (i.e., n might not be equal to m), naturally the IOU to be transferred must be equal to the IOU to be received (i.e., $\sum_i \beta_{in}[i] = \sum_j \beta_{out}[j]$). In such setting, PathJoin, our novel protocol to enforce *atomic* transactions *fully compatible* with Ripple, must ensure that either all the $\sum_i \beta_{in}[i]$ IOU are transferred from input to output wallets or no IOU is transferred.

Using a Shared Wallet. It is possible to create a wallet shared among the users such that only when all users agree, a transaction involving the shared wallet is performed. This effectively allows to add one synchronization round: Each user i first transfers $\beta_{in}[i]$ IOU to a

Unauthenticated
Download Date | 2/21/18 6:24 PM



(a) Credit network after the set up of the shared wallets and the output wallets has been carried out



(b) Credit network after PathJoin has been carried out without any disruptive user

Fig. 3. An illustrative example of an atomic transaction using PathJoin. The input wallets A_{in}, B_{in}, C_{in}, D_{in} and E_{in} transfer 20, 10, 30, 15 and 5 IOU correspondingly. The output wallets Y_{out} and Z_{out} receive 55 and 25 IOU, respectively. Solid arrows depict credit links between two wallets. Single values on edges denote the current balance and no upper limit. Values a/b on the links denote: a current balance and b upper limit. After a successful execution of PathJoin, it is possible to perform a transaction from the output wallets (e.g., from Y_{out} for up to 55 IOU using vk^s_{out}, and vk_{gw} as the first hops in the transaction path).

shared wallet and only when $\sum_i \beta_{in}[i]$ IOU are collected, they are sent to the output wallets. This, however, does not solve the fairness problem either. Once all the IOU are collected in the shared wallet, a (malicious) user could collaborate with the rest to create and sign a transaction to one of the output wallets and then disconnect. In this manner, the IOU to be transferred to the rest of output wallets are locked in the shared wallet.

Solution: Two Shared Wallets. The idea underlying our approach for an atomic transaction is to use two synchronization rounds via two shared wallets (say vk^s_{in} and vk^s_{out}).

An example is depicted in Fig. 3: Five users with input wallets $\text{VK}_{in}[\,] := \{\mathsf{A}_{in}, \mathsf{B}_{in}, \mathsf{C}_{in}, \mathsf{D}_{in}, \mathsf{E}_{in}\}$ would like to transfer $\beta_{in}[\,] := \{20, 10, 30, 15, 5\}$ into two output wallets $\text{VK}_{out}[\,] := \{\mathsf{Y}_{out}, \mathsf{Z}_{out}\}$. These two output wallets must receive $\beta_{out}[\,] := \{55, 25\}$. To achieve that, in the first round users jointly create a credit link from each input wallet ($\text{VK}_{in}[i]$) to vk^s_{in} with $\beta_{in}[i]$ IOU on them. Moreover, users jointly create a credit link from each of the output wallets ($\text{VK}_{out}[j]$) to vk^s_{out} with no IOU on them but an upper limit of $\beta_{out}[j]$. At this point, credit at each $\text{VK}_{out}[j]$ cannot be issued as part of a transaction because vk^s_{out} does not have incoming credit yet (see Fig. 3a). The second synchronization round can be then used to overcome that. All users jointly create a transaction from vk^s_{in} to vk^s_{out} for a value of $\sum_i \beta_{in}[i]$ IOU. Then, vk^s_{out} gets enough credit that can be used by each of the output wallets $\text{VK}_{out}[j]$ (see Fig. 3b).

4.2 Creating the Set of Output Wallets Anonymously

The possibility of performing atomic transactions on its own does not provide a complete path-mixing solution. Assume an atomic transaction from n input wallets to n output wallets, where each wallet transfers a fixed amount of IOU β. Even then, a naive path mixing where each user publishes her output wallet in a manner that can be linked to her identity, clearly violates unlinkability in the presence of a network attacker. In order to overcome this challenge, users need to jointly come up with a set of their output wallets such that the owner of a given output wallet is not leaked to the rest of users.

Several P2P mixing protocols proposed in the literature [23, 51, 52] implement a permutation that ensures the aforementioned property as required in our decentralized path mixing protocol. Among them, we decide to use DiceMix [52] due to its efficiency, but in principle we could have used any P2P mixing protocol.

5 PathJoin: Enabling Atomic Transactions in Ripple

Here we describe the details of PathJoin, our novel protocol for atomic transactions in credit networks. It can be seen as the counterpart of CoinJoin for Ripple.

Unauthenticated
Download Date | 2/21/18 6:24 PM

Conventions for the Pseudocode. We write $\text{ARR}[i]$ for arrays where i is the index. We denote the full array (all its elements) as $\text{ARR}[\,]$.

Our protocols are supposed to terminate in the presence of malicious disruptive users, so they cannot just halt when some users send invalid messages or omit messages, but have to handle those cases explicitly (as also done in DiceMix [52], which we use as a framework for our protocols). Message x is broadcast using "**broadcast** x". The command "**receive** $\text{X}[p]$ **from all** $p \in P$ **where** $X(\text{X}[p])$ **missing** $C(P_{off})$" attempts to receive a message from all users $p \in P$. The first message $\text{X}[p]$ from user p that fulfills predicate $X(\text{X}[p])$ is accepted and stored as $\text{X}[p]$; all further messages from p are ignored. When a timeout is reached, the command C is executed, which has access to a set $P_{off} \subseteq P$ of users that did not send a (valid) message.

5.1 Building Blocks

Digital Signatures. We require a digital signature scheme (KeyGen, Sign, Verify) unforgeable under chosen-message attacks (UF-CMA). The algorithm KeyGen returns a private signing key sk and the corresponding public verification key vk. On input message m, $\mathsf{Sign}(sk, m)$ returns σ, a signature on message m using signing key sk. The verification algorithm $\mathsf{Verify}(vk, \sigma, m)$ outputs *true* iff σ is a valid signature for m under the verification key vk.

In practice, we rely on the existing signature scheme available in the credit network. The Ripple network supports ECDSA on the secp256k1 elliptic curve or EdDSA on Curve25519. We use EdDSA due to its support for simple distributed signatures, which will be required by our protocol.

Ripple Network Operations. We use the following operations available in the Ripple network.

$(vk, sk) := \mathsf{AccountGen}()$	Generate wallet keys
$tx := \mathsf{CreateTx}(vk_1, vk_2, v)$	Create path-based transaction
$tx := \mathsf{CreateLink}(vk_1, vk_2, v)$	Create link $vk_1 \to vk_2$ (limit v)
$tx := \mathsf{ChangeLink}(vk_1, vk_2, v)$	Modify link $vk_1 \to vk_2$ by v
$\{v, \bot\} := \mathsf{TestLink}(vk_1, vk_2)$	Query IOU on link $vk_1 \to vk_2$
$\{0, 1\} := \mathsf{Apply}(tx, \sigma)$	Apply signed tx to network

A transaction tx becomes valid when is signed by the appropriate wallet's signing key. A tx from CreateTx and ChangeLink must be signed by sk_1 (i.e., the signing key of wallet vk_1), whereas a tx from CreateLink must be signed by sk_2. Finally, a tx from TestLink does not require a signature. A transaction tx is applied to the Ripple network after invoking $\mathsf{Apply}(tx, \sigma)$ with the correct signature.

Shared Wallet. We manage a shared wallet using an interactive distributed signature scheme (SAccountCombine, SSign, Verify) that is fully compatible with the Ripple network. In a distributed signature scheme, every user creates a fresh pair of verification and signing keys, publishes the verification key, and combines the fresh verification keys from all users to derive the shared wallet's verification key. Every user then uses her fresh signing key to generate her signature (share) on a message m (e.g., a transaction agreed among all users). The combination of all these signature shares results in a new signature on the message m verifiable under the shared wallet's verification key. In the following, we summarize the required functionalities. We detail the distributed signature scheme in Appendix B.

A shared wallet is created as follows. First, each user locally creates a fresh EdDSA Ripple wallet (vk^*, sk^*), using AccountGen, that constitutes her share for the shared wallet vk^s. The shared wallet can be then calculated as $vk^s := \mathsf{SAccountCombine}(\text{VK}^*[\,])$, where $\text{VK}^*[\,]$ denotes the array containing one verification key share vk^* for each user.

Note that it is possible to construct only the verification key of a shared wallet but not the corresponding signing key. Instead, users can jointly create a signature σ on a message m verifiable by the shared wallet's verification key vk^s. For that, each user invokes $(P_{mal}, \sigma) := \mathsf{SSign}(P, my, \text{VK}_{in}[\,], sk_{in}, \text{VK}^*[\,], sk^*, m, sid)$, where P is the set of users participating in the protocol except the invoking user, $\text{VK}_{in}[\,]$ is the list of input wallets from all users, sk_{in} is the secret key for the invoking user's input wallet, $\text{VK}^*[\,]$ is the list of verification key shares for the shared wallet's verification key vk^s from all users, sk^* denotes the signing key share of the invoking user and sid is an identifier of the current session. If the signature σ cannot be created due to misbehaving or faulty users, the functionality SSign returns a set P_{mal} containing such users. Otherwise, it returns an empty set along with σ.

5.2 PathJoin

Assumptions. We assume that each user has an input wallet $\text{VK}_{in}[i]$ with an arbitrary amount of $\beta_{in}[i]$ IOU. We assume that all input wallets have a credit link with a common wallet (i.e., vk_{gw}). In practice, gateways can play the role of such common wallet as they are highly

Unauthenticated
Download Date | 2/21/18 6:24 PM

connected nodes in the Ripple network. Moreover, we assume that there is only one IOU currency (e.g., USD) over the credit links in the Ripple network, as otherwise unlinkability can be trivially broken: Input and output wallets using a distinct currency belong to the same user. Finally, for clarity of exposition, we assume that $\mathsf{Apply}(tx, \sigma)$ returns immediately after tx is applied to the Ripple network. In practice, a tx is applied in a matter of seconds [54].

In multiple steps of the protocol, each user will submit to the Ripple network a copy of the same correctly signed transaction. This does not have negative security implications: The transaction is only applied once to the Ripple network since every transaction contains a sequence number to avoid replay attacks.

Under these assumptions, the PathJoin protocol works as described below. A detailed pseudocode for the protocol is presented in Algorithm 1.

Phase 1: Create and Connect Input Shared Wallet. The users jointly create a shared input wallet, that we denote by vk^s_{in}. We require that only transactions starting at vk^s_{in} can be performed. For that, the *rippling* option (see Section 2.1) must be disabled at each credit link with vk^s_{in} wallet.

Then, users jointly create a credit link from each input wallet $\text{VK}_{in}[i]$ to vk^s_{in}. Such credit links are then signed by all users using their signing key shares for the input shared wallet. If a user generates a wrong partial signature, the honest users consider her to be malicious. Otherwise, these credit links along with their signatures are submitted to the Ripple network.

Additionally, each user p locally creates and signs a transaction that issues $\beta_{in}[p]$ credit to the recently created link $\text{VK}_{in}[p] \rightarrow vk^s_{in}$. Such signature is then broadcast to every other user in the protocol, what allows them to apply the funding transactions in the Ripple network. If some user refuses to fund such a credit link, the honest users consider her to be malicious.

Phase 2: Create and Connect Output Shared Wallet. The shared output wallet vk^s_{out} is created in the same manner as the shared input wallet vk^s_{in}. However, transactions that use vk^s_{out} as intermediate hop must be allowed in this case and for that, the rippling option must be enabled for the credit links of vk^s_{out}. Then, for each output wallet j, users jointly create a credit link from each $\text{VK}_{out}[j]$ to vk^s_{out} with an upper limit of $\beta_{out}[j]$. Moreover, the users jointly create a link $vk_{gw} \rightarrow vk^s_{out}$ with no IOU on it. These links will later allow to transfer up to $\beta_{out}[j]$ IOU from the wallet $\text{VK}_{out}[j]$.

The details of creating the links and verifying the corresponding signatures are similar to the previous case involving the input shared wallet. As before, users ensure that only links from known output wallets are created. If during this phase some user generates an invalid signature, the honest users consider her to be malicious.

Phase 3: Final Transaction. At this point, the vk^s_{out} wallet does not have any incoming credit and thus no transaction from an output wallet through vk^s_{out} can be performed yet. To solve this situation, the users jointly create a transaction transferring $\sum_j \beta_{out}[j]$ IOU from vk^s_{in} to vk^s_{out}. This transaction is possible using the n available paths through each of the users' input wallets. If some user does not sign such transaction, the honest users consider her to be malicious.

Interestingly, this transaction makes credit to flow from vk^s_{in} to vk^s_{out} so that the credit link between vk_{gw} and vk^s_{out} has now $\sum_j \beta_{out}[j]$ IOU. This fact enables now transactions from each output wallet to the rest of the credit network.

Correctness. The final transaction ensures that exactly $\beta_{in}[p]$ are transferred through the input wallet of the user p (i.e., $\text{VK}_{in}[p]$). Moreover, the upper limit on the links from each output wallet to vk^s_{out} ensures that wallet $\text{VK}_{out}[j]$ has only access to $\beta_{out}[j]$ IOU. This demonstrates the correctness of PathJoin.

5.3 Security Analysis

In this section, we first describe the notion of atomicity. We then argue that PathJoin achieves atomicity.

Atomicity. A path mixing protocol is atomic if either β IOU are transferred from input wallets to output wallets or no IOU is transferred.

In the following, we argue that PathJoin achieves atomicity. In order to see that, we make the following observations. First, the creation and set up of the shared wallets do not involve the credit to be transferred. Second, the deactivation of rippling option on vk^s_{in} credit links ensures that only transactions starting at vk^s_{in} are accepted by the Ripple network. This prevents a malicious user from stealing honest user's credit using vk^s_{in} as intermediate wallet, e.g., by means of a transaction with path: VK[malicious] – vk^s_{in} – VK[honest] – vk_{gw} – VK[malicious]. (Circular transactions are accepted and used in the Ripple network. For example, a transaction of the form VK[p] – vk_{gw_1} – ... – vk_{gw_2} – VK[p], where ... denotes an arbitrary set of wallets, can be used by user p to exchange IOU from gateway 1 to gateway 2.)

Unauthenticated
Download Date | 2/21/18 6:24 PM

Algorithm 1. PathJoin

procedure PJ($P, my, \mathrm{VK}_{in}[\,], sk_{in}, \beta_{in}[\,], \mathrm{VK}_{out}[\,], \beta_{out}[\,], sid$)
▷ Create shares for shared wallets and broadcast them
$(\mathrm{VK}^*_{in}[my], sk^*_{in}) :=$ AccountGen()
$(\mathrm{VK}^*_{out}[my], sk^*_{out}) :=$ AccountGen()
broadcast ($VK^*_{in}[my]$, $VK^*_{out}[my]$, Sign(sk_{in}, ($VK^*_{in}[my]$, $VK^*_{out}[my]$, sid)))
receive $(\mathrm{VK}^*_{in}[p], \mathrm{VK}^*_{out}[p], \sigma[p])$ **from all** $p \in P$
where Verify$(\mathrm{VK}_{in}[p], \sigma[p], (\mathrm{VK}^*_{in}[p], \mathrm{VK}^*_{out}[p]))$
missing P_{off} **do return** P_{off}
▷ Create shared wallets
$vk^s_{in} :=$ SAccountCombine$(\mathrm{VK}^*_{in}[\,], my, P)$
$vk^s_{out} :=$ SAccountCombine$(\mathrm{VK}^*_{out}[\,], my, P)$
▷ Create credit links $\mathrm{VK}_{in}[p] \to vk^s_{in}$
for all $p \in P \cup \{my\}$ **do**
$\mathrm{LINK}_{in}[p] :=$ CreateLink$(\mathrm{VK}_{in}[p], vk^s_{in}, \infty)$
$(\sigma_{in}[p], P_{mal}) :=$ SSign$(P, my, \mathrm{VK}_{in}[\,], sk_{in}, \mathrm{VK}^*_{in}[\,], sk^*_{in}, \mathrm{LINK}_{in}[p], (sid, 0, p))$
if $P_{mal} \neq \emptyset$ **then return** P_{mal}
Apply$(\mathrm{LINK}_{in}[p], \sigma_{in}[p])$
▷ Fund credit links $\mathrm{VK}_{in}[p] \to vk^s_{in}$
for all $p \in P \cup \{my\}$ **do**
$\mathrm{LINK}'_{in}[p] :=$ ChangeLink$(\mathrm{VK}_{in}[p], vk^s_{in}, \beta_{in}[p])$
$\sigma'_{in} :=$ Sign$(sk_{in}, \mathrm{LINK}'_{in}[my])$
broadcast σ'_{in}
receive $\sigma'_{in}[p]$ **from all** $p \in P$
where Verify$(\mathrm{VK}_{in}[p], \sigma'_{in}[p], \mathrm{LINK}'_{in}[p])$
missing P_{off} **do return** P_{off}
for all $p \in P \cup \{my\}$ **do**
Apply$(\mathrm{LINK}'_{in}[p], \sigma'_{in}[p])$
▷ Verify $\mathrm{VK}_{in}[p] \to vk^s_{in}$ link for every participant
$P_{mal} := \emptyset$
for all $p \in P$ **do**
$v :=$ TestLink$(\mathrm{VK}_{in}[p], vk^s_{in})$
if $v = \perp \ \vee \ v < \beta_{in}[p]$ **then** $P_{mal} := P_{mal} \cup \{p\}$
if $P_{mal} \neq \emptyset$ **then return** P_{mal}
▷ Create credit links $\mathrm{VK}_{out}[p] \to vk^s_{out}$
for $i := 1, \ldots, |\mathrm{VK}_{out}[\,]|$ **do**
$\mathrm{LINK}_{out}[i] :=$ CreateLink$(\mathrm{VK}_{out}[i], vk^s_{out}, \beta_{out}[i])$
$(\sigma_{out}[i], P_{mal}) :=$ SSign$(P, my, \mathrm{VK}_{in}[\,], sk_{in}, \mathrm{VK}^*_{out}[\,], sk^*_{out}, \mathrm{LINK}_{out}[i], (sid, 1, i))$
if $P_{mal} \neq \emptyset$ **then return** P_{mal}
Apply$(\mathrm{LINK}_{out}[i], (\sigma_{out}[i]))$
▷ Create link $vk_{gw} \to vk^s_{out}$
$\mathrm{LINK}_{gw} :=$ CreateLink$(vk_{gw}, vk^s_{out}, \infty)$
$(\sigma_{gw}, P_{mal}) :=$ SSign$(P, my, \mathrm{VK}_{in}[\,], sk_{in}, \mathrm{VK}^*_{out}[\,], sk^*_{out}, \mathrm{LINK}_{gw}, (sid, 2))$
if $P_{mal} \neq \emptyset$ **then return** P_{mal}
Apply$(\mathrm{LINK}_{gw}, \sigma_{gw})$
▷ Final transaction
$tx :=$ CreateTx$(vk^s_{in}, vk^s_{out}, \sum_{p \in P} \beta_{in}[p])$
$(\sigma_{tx}, P_{mal}) :=$ SSign$(P, my, sk^*_{in}, \mathrm{VK}_{in}[\,], sk_{in}, \mathrm{VK}^*_{in}[\,], sk^*_{in}, tx, (sid, 3))$
if $P_{mal} \neq \emptyset$ **then return** P_{mal}
Apply(tx, σ_{tx})
return $\emptyset$ ▷ Success!

Third, the transaction from vk^s_{in} to vk^s_{out} sends all the credit at once. Thus, either all users contribute the expected credit for the transaction or none of them do. Moreover, this transaction is created and submitted to the Ripple network only if there is a link from each output wallet to vk^s_{out} with the expected credit upper limit. In this manner, it is ensured that credit in the output wallets can be used later to perform a transaction to any other wallet in the credit network.

Note that the transaction from vk^s_{in} to vk^s_{out} is the last step of the protocol. Thus, whenever the current run of the protocol is disrupted by a malicious user, the credit on the links between the $\mathrm{VK}_{in}[\,]$ and vk_{gw} is not used and can be reused in another invocation of PathJoin. Finally, the links between $\mathrm{VK}_{in}[\,]$ and vk^s_{in} might stay funded after disruption is detected. However, this credit is created only for the purpose of running the protocol and it does not have value outside of it.

5.4 Extensions and Applications

Other Credit Networks. We have focused the description of PathJoin to the Ripple network since it is currently the most widely deployed credit network. Nevertheless, the same protocol can be used to achieve atomic transactions in other credit networks provided that they offer all the functionality required by PathJoin. For instance, PathJoin can be also deployed in the Stellar network. The Stellar network provides functionality to create links, set their upper limit and perform path-based transactions [4]. Moreover, Stellar implements a mechanism to enable and disable the rippling option as in Ripple [2]. Finally, Stellar supports the same digital signature schemes as Ripple and thus shared wallets can also be implemented in Stellar.

Crowdfunding Application. In this work, we use atomic transactions as a building block to achieve anonymous transactions. Nevertheless, we note that atomic transactions become of interest on its own for other scenarios. For example, they can enable a *crowdfunding* transaction in a credit network. Interestingly, the example depicted in Fig. 3 is indeed a crowdfunding transaction where the five input wallets are used to fund the two output wallets. PathJoin ensures that either every user participating in the crowdfunding transfers the expected amount of IOU into the crowdfunding wallets (e.g., Y_{out} and Z_{out}) or none of the users transfers any IOU.

Unauthenticated
Download Date | 2/21/18 6:24 PM

6 PathShuffle: A Decentralized Path Mixing Protocol

We use PathJoin as a building block to create PathShuffle, our fully-fledged path mixing protocol. What is left is to come up with a set of fresh output wallets anonymously, i.e., without revealing which output wallet belongs to which input wallet (or to which network identity, which is in turn linkable to the input wallet). We use DiceMix for this purpose.

6.1 Building Block: DiceMix

We require a P2P mixing protocol that given a set of n users, each having a Ripple wallet, allows them to agree on the set of all their wallets anonymously so that it is not revealed which wallet belong to which user. To achieve this, we use DiceMix [52], because it is the most efficient P2P mixing protocol in our setting. While other P2P mixing protocols [23, 51] require a number of communications rounds linear in the number of users and at least quadratic in the presence of misbehaving users, DiceMix runs in a constant number of rounds independently on the number of participating users and it grows only to a linear number of rounds in the number of misbehaving users. We refer the reader to [52] for more details.

Interface. Consequently, we specify PathShuffle in the framework of DiceMix. To do so, we need to specify two operations Gen() and Confirm(...), which depend on the application that uses DiceMix as a building block (PathShuffle in our case). First, Gen() specifies how to generate the messages to be used in consequent steps of the DiceMix protocol. We implement Gen() by invoking the wallet generation algorithm of Ripple to generate a fresh output wallet (vk_{out}, sk_{out}), and we return the verification key vk_{out} as the message to be mixed.

Second, the purpose of the operation Confirm$(P, \text{VK}_{out}[], my, \text{VK}_{in}[], sk_{in}, sid, runid)$ is to confirm the result of the message mixing, i.e., the anonymized set $\text{VK}_{out}[]$. The operation has access to the set P of users, their verification keys $\text{VK}_{in}[]$ (those associated with the input wallets), the identity my of the user and her signing key sk_{in} (associated with his verification key $\text{VK}_{in}[my]$); the parameters sid and $runid$ are session and run identifiers, respectively, used to properly distinguish between different sessions and confirmation tries (see Ruffing et al. [52] for details). We implement Confirm(...) by directly invoking PathJoin.

Contract. The operation Gen() must generate, after each invocation, a new random message with enough entropy to be unpredictable. In our case, this requirement is met by the Ripple wallet generation algorithm.

The operation Confirm(...) must meet certain natural requirements to ensure proper and secure functioning of DiceMix. First, Confirm(...) must ensure that it terminates only when all users call it with the same anonymized message set $\text{VK}_{out}[]$. This holds in our case: PathJoin will fail if the messages (i.e., the output wallets) are different, because then it is impossible for users to agree on which wallets must be connected to the shared wallet.

Second, in case confirmation fails because some user refuses to confirm, Confirm(...) must return the set of misbehaving or faulty users that did not allow the atomic transaction. This ensures that DiceMix can exclude these users, discard the messages and re-try with new messages generated by Gen().[2]

Confirm(...) can assume that it obtains the correct set of anonymized messages. This ensures that, in our context, the output wallet of every user is present in this set, and thus every user who refuses to sign the different transactions required in PathJoin can safely be reported as misbehaving or faulty to DiceMix. (And indeed, we have designed PathJoin with this requirement in mind. As specified in Section 5, PathJoin returns the set of users who refuse to participate in the mixing.)

6.2 PathShuffle

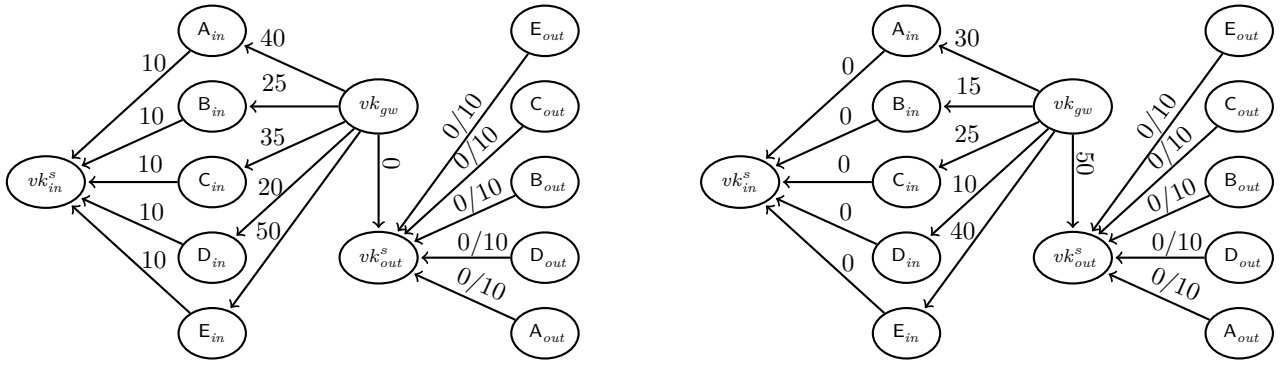
Assumptions. As PathShuffle uses PathJoin as building block, we make the same assumptions here. Additionally, we assume that during the bootstrapping process (see Section 3.2), the users participating in the PathShuffle protocol have agreed on sid, an identifier for the current execution of PathShuffle; and β, the amount of IOU (in some currency) to be mixed in the path mixing.

Description. The PathShuffle protocol works as defined in Algorithm 2. Additionally, we have depicted an example of execution in Fig. 4, where five users run the PathShuffle protocol to mix 10 IOU.

The PathShuffle protocol starts by having each user i invoking Start-DiceMix$(P, my, \text{VK}_{in}[], sk_{in}, sid)$, where P is the set of protocol participants except the invoking user, who is identified by my, $\text{VK}_{in}[]$ is an array of the verification keys associated with input wal-

2 Discarding the old messages is necessary for unlinkability [52].

Unauthenticated
Download Date | 2/21/18 6:24 PM



(a) Credit network after the set up of the shared wallets and output credit has been issued

(b) Credit network after carrying out PathShuffle without any disruptive user

Fig. 4. An illustrative example of PathShuffle to mix 10 IOU among five users. Solid arrows depict credit links between two wallets. Single values on edges denote the current balance and no upper limit. Values a/b on the links denote: a current balance and b upper limit. After a successful path mixing, user A can perform a transaction for up to 10 IOU using vk^s_{out} and vk_{gw} as first hops in the transaction path.

lets, sk_{in} is the secret key for the input wallet of the user invoking the function, and *sid* is the session identifier agreed among the users. Whenever functions Gen() or Confirm(...) are invoked, they are executed as described in Section 6.1.

Assume that a user wishes to anonymously pay for a service in the Ripple network. For that, the user first executes PathShuffle to transfer IOU into a fresh Ripple wallet of her own. This ensures that even if a round in PathShuffle is disrupted, the intended payee's wallet is not disclosed. After the credit has been transferred into an output wallet of the user, she can use it to anonymously pay for the service in the Ripple network.

Algorithm 2. PathShuffle

procedure PathShuffle($P, my, \mathrm{VK}_{in}[\,], sk_{in}, sid$)
  return Start-DiceMix($P, my, \mathrm{VK}_{in}[\,], sk_{in}, sid$)
procedure Gen()
  $(vk_{out}, sk_{out}) :=$ AccountGen() ▷ stores sk_{out} for latter use
  return vk_{out}
procedure Confirm($P, \mathrm{VK}_{out}[\,], my, \mathrm{VK}_{in}[\,], sk_{in}, sid, runid$)
  for all $p \in P$ **do**
    $\beta_{in}[p] := \beta$
  for all $p \in P$ **do**
    $\beta_{out}[p] := \beta$
  $P_{mal} :=$ PathJoin($P, my, \mathrm{VK}_{in}[\,], sk_{in}, \beta_{in}[\,], \mathrm{VK}_{out}[\,], \beta_{out}[\,], (sid, runid)$)
  ▷ Return set of malicious users; $\emptyset$ means success.
  return P_{mal}

6.3 Security and Privacy Analysis

In this section, we first argue how PathShuffle achieves the application requirements of *correct confirmation* and *correct exclusion* required by DiceMix, as described in [52, Section IV.B.2]. These two properties are necessary to ensure termination. We then discuss how PathShuffle achieves the security and privacy goals for a path mixing protocol as defined in Section 3.3.

6.3.1 Requirements Imposed by DiceMix

Correct Confirmation. PathShuffle must ensure that if a honest user invokes Confirm(...) on input (among others) the list of anonymously published fresh output wallets and is successful (i.e., returns an empty set P_{mal}), then all honest users have invoked Confirm(...) on the same list of output wallets.

To prove that, we first we observe that PathShuffle only invokes PathJoin to implement Confirm(...). Second, we observe that PathJoin requires that users jointly agree on each of the steps of the PathJoin protocol by creating a distributed signature on messages that are deterministically derived from the input. This only succeeds if every user has the same list of output wallets. Therefore, PathJoin fulfills *correct confirmation*.

Correct Exclusion. Assuming that the bulletin board is honest, PathShuffle must ensure that if Confirm(...) returns a set of malicious users P_{mal} for a honest user p, then it returns the same set for any other honest user p' and the set P_{mal} does not contain any honest user.

Unauthenticated
Download Date | 2/21/18 6:24 PM

We may assume that CONFIRM(...) is called with the same arguments by every honest user, and $\text{VK}_{out}[]$ represents the correct set of output wallets, i.e., it contains the output wallets of all honest users. This is guaranteed by DiceMix [52].

Observe that PathShuffle only invokes PathJoin to implement CONFIRM(...). First, we prove that CONFIRM(...) returns the same set P_{mal} for every honest user. Since the bulletin board is honest by assumption, all honest users receive the same broadcasts. Obverse that all honest users call PathJoin with the same arguments. By code inspection of PathJoin, the set P_{mal} depends only on these broadcast messages (even when P_{mal} is determined within SSign(...)), and on the result of calls to TestLink(...) with the same arguments for every honest user. Since all honest users receive the same broadcasts, and we assume that the credit network reaches consensus on the result of TestLink(...), the set P_{mal} is the same for every user honest.

Next we prove that P_{mal} does not contain an honest user. Since the bulletin board is honest, the network is reliable and messages from all honest users reach all honest users. Given that, it is easy to verify that an honest user p accepts all messages sent by another honest user p' and thus $p' \notin P_{mal}$ for the set P_{mal} returned by user p.

6.3.2 Security and Privacy Goals for Path Mixing

Unlinkability. Since PathShuffle fulfills the contract with DiceMix, this building block ensures that the output wallets are published without leaking the relation between a single output wallet and its owner. Moreover, a look at the pseudocode of the confirmation step (i.e., the PathJoin protocol) shows that operations on PathJoin are totally independent on who is the owner of each output wallet: Each input wallet transfers β IOU and each output wallet receives β IOU. Therefore, PathJoin does not leak the owner of any output wallet. (Actually DiceMix is designed such that the confirmation operation run by some user is not given the information which of the messages belongs to this user. Therefore, PathJoin cannot possibly leak this relation.) This proves that PathShuffle achieves unlinkability.

Termination. Since PathShuffle fulfills the contract with DiceMix, termination property of DiceMix carries over to PathShuffle.

Correct Balance. The DiceMix protocol does not perform any operation involving the credit of the users. DiceMix ensures for each user that CONFIRM(...) (i.e., PathJoin) is only called if the list of output wallets contains her own output wallet. Thus if the PathJoin succeeds, the same amount β of IOU that is taken from her input wallet is transferred to her output wallet. If PathJoin fails, no IOU is transferred at all. This proves correct balance.

6.4 Performance Analysis

In PathShuffle, we use the DiceMix protocol as defined in the original paper [52]. However, in our work we have implemented the functionality for CONFIRM(...) in a different manner. Instead of a single round collecting signatures from each user as in [52], PathShuffle implements the CONFIRM(...) functionality using the PathJoin protocol. Thus, in this section we restrict our analysis to PathJoin and the performance analysis for the core of DiceMix described in [52] carries over in our work.

Implementation. We have implemented PathJoin in JavaScript by modifying the current Ripple code [49]. In particular, we have implemented the shared wallet management by modifying the elliptic library, an implementation of the EdDSA digital signature scheme supported in Ripple. Moreover, we have used the API provided by the ripple-lib library [9] to implement the submission of transactions to the Ripple network. Our source code is publicly available [43] under the MIT license.

Implementation-level Optimizations. For readability, we have specified Algorithm 1 in sequential steps. However, several of these steps can be carried out in parallel, improving thereby the overall performance of the PathJoin protocol. First, both shared wallets vk^s_{in} and vk^s_{out} can be created in parallel. Second, the creation of links between vk^s_{in} and input wallets and the creation of links between vk^s_{out} and output wallets are independent operations and can be fully parallelized. Thus, it is possible to perform a single SSign(...) invocation to jointly sign the create link transactions for all of these links.

Additional optimizations are possible to reduce the number of communication rounds. In particular, the SSign(...) procedure requires two broadcast rounds (see Appendix B): One round to broadcast the randomness chosen by each user, and a second round to broadcast the signature share from each user. As the randomness is chosen independently of the message to be signed, this broadcast can be integrated with a previous communication round in the protocol. In this manner, a call to SSign(...) costs only one extra communication round.

Unauthenticated
Download Date | 2/21/18 6:24 PM

Communication. A protocol based on DiceMix needs $(c+3)+(c+1)f$ communications rounds, where c is number of communication rounds required by $\textsc{Confirm}(\dots)$ and f is the number of disrupting users.

In our case $c = 2$, so PathShuffle needs $5 + 3f$ communication rounds. As mentioned above, broadcast of random elements (e.g., shares for vk^s_{in} and vk^s_{out} and randomness for each of the invocations of $\mathsf{SSign}(\dots)$) can be carried out before PathJoin is invoked. Then, one communication round is required for each of the two times $\mathsf{SSign}(\dots)$ is invoked (see Appendix B for details): First to jointly sign the creation of the links $\textsc{Vk}_{in}[i] \to vk^s_{in}$, $vk_{gw} \to vk^s_{out}$, and $\textsc{Vk}_{out}[j] \to vk^s_{out}$; and second to jointly sign the final transaction that transfers IOU from vk^s_{in} to vk^s_{out}. Note that, as the credit links created in PathJoin are deterministically defined from the input of the protocol, the signatures on the funding transactions for the links $\textsc{Vk}_{in}[i] \to vk^s_{in}$ can be broadcast the first time $\mathsf{SSign}(\dots)$ is invoked.

Computation. In this test we measure the computation time required by each user on a computer with an Intel i7, 3.1 GHz processor and 16 GB RAM. Given the aforementioned implementation-level optimizations, we have studied the running time for a single run of $\mathsf{SAccountCombine}(\dots)$ and $\mathsf{SSign}(\dots)$ algorithms. This thus simulates the creation of a single shared wallet and the signature of a transaction involving a shared wallet. We have observed that even with 50 participants, $\mathsf{SAccountCombine}(\dots)$ takes 537 ± 66.8 milliseconds and $\mathsf{SSign}(\dots)$ takes 45 ± 3.57 milliseconds using our unoptimized implementation. It is important to note that it takes approximately 5 seconds for a transaction to be applied into the current Ripple network [54]. Thus, the overall running time of PathShuffle even considering the computation time required for DiceMix is mandated by the time necessary for the Apply operations at each communication round of PathJoin.

Time. We observe that each communication round in the confirmation algorithm requires to submit (possibly several parallel) transactions to the Ripple network. It takes approximately 5 seconds for a transaction to be applied to the current Ripple network. Therefore, we expect that this mandates the time per communication round. Altogether, we expect the protocol to run in under 20 s with a reasonable number of 50 non-disruptive users: Confirmation takes $2 \cdot 5$ s and the required functionality from DiceMix needs about 8 s to complete [52].

Scalability. The time to execute DiceMix is dominated by its communication cost, as it requires each user to send $n \cdot |m|$ bits, where n is the number of users and $|m|$ is the number of bits of the mixed message (e.g., a Ripple wallet in our case). Nevertheless, it has been shown that DiceMix can scale up to a moderate number of users (e.g., 50 users) [52].

In PathJoin, the execution time is dominated by the $\mathsf{Apply}(\dots)$ operations. Although PathJoin requires a number of credit links linear in the number of users, their corresponding operations can be parallelized so that only 5 seconds are needed per synchronization round. Overall, given the synchronization required for the broadcasts in DiceMix and the interaction with the Ripple network in PathJoin, we expect that PathShuffle provide anonymity guarantees to moderate size groups of users.

Compatibility. We have simulated a run of PathShuffle without disruption in the currently deployed Ripple network. In particular, we have successfully recreated the scenario depicted in Fig. 4. As a proof-of-concept, users are simulated by our JavaScript implementation in a single machine. The mixed IOU are denominated in PSH, a user-defined currency created for the purpose of this experiment. We describe in (Table 1, Appendix C) the mapping between wallets in Fig. 4 and the same wallets in the Ripple network. Furthermore, detailed information about the Ripple nodes and the transactions[3] involved in the test can be found using the Ripple Charts and Ripple RPC tools.

Generality. In our exposition of the PathShuffle protocol, we have focused on the Ripple network given that is currently the most widely deployed credit network. However, PathShuffle can be used to perform path mixing in other credit networks. We make two observations. First, the DiceMix protocol is application agnostic and can be executed in any application scenario as long as the $\textsc{Gen}(\dots)$ and $\textsc{Confirm}(\dots)$ achieve the expected requirements. Second, as we described in Section 5.4, PathJoin requires operations inherent to credit networks. Thus, the PathShuffle protocol is compatible with other credit networks (e.g., Stellar).

6.5 Practical Considerations

Handling Fees. Every wallet in a path might charge some fee as a reward for allowing a transaction. Thus, the amount of IOU received by the receiver might be lower than the amount sent by the sender. However,

3 See https://tinyurl.com/zc3yu8l, https://tinyurl.com/hb9722d

Unauthenticated
Download Date | 2/21/18 6:24 PM

PathShuffle requires that in the transaction from vk^s_{in} to vk^s_{out} at least $n \cdot \beta$ IOU are received by vk^s_{out}.

Nevertheless, this is not a burden to deploy PathShuffle in the Ripple network since it allows to check in real time the fees associated to a given payment path. Therefore, it is possible to set the necessary IOU between every input wallet and vk^s_{in} so that at least $n \cdot \beta$ IOU are received by vk^s_{out} in the final transaction of PathShuffle.

Funding New Wallets. Ripple applies reserve requirements to each new wallet in order to prevent spam or malicious usage [6]. At the time of writing, Ripple applies a base reserve of 20 XRP and an additional reserve of 5 XRP for each of the credit links associated to the wallet. PathShuffle can handle this reserve.

A XRP payment allows the direct exchange of XRP between two wallets. A payment of β XRP from the sender wallet to the receiver wallet is performed as follows. If the XRP balance of sender wallet is at least β, then β XRP are reduced from the sender wallet's balance and β XRP are added to the receiver wallet's balance.

Using XRP direct payments, shared wallets can be funded using any wallet belonging to the users. For example, each user can send its corresponding share of XRP reserve to each of the shared wallets. However, fresh output wallets from a user cannot be funded directly from the user input wallet, as this would break the unlinkability property we are after with PathShuffle.

Instead, since XRP payments are similar to Bitcoin payments, we envision that it is possible to create a transaction in Ripple, similar to a CoinJoin transaction, to anonymously send necessary XRP from input to output wallets. Alternatively, users could send the necessary XRP to the gateway, and trust it to fund all their output wallets. This is feasible because gateways are already trusted for the bootstrapping of credit links and the necessary monetary amount for funding a wallet is very small.[4]

If none of these options is available, and in order to maintain full compatibility with the current Ripple protocol, we propose a more elaborate XRP mixing protocol (see Appendix A) at the cost of increased communication complexity.

Censorship. A PathShuffle transaction is clearly distinguishable from other Ripple transactions. However, PathShuffle transactions cannot be easily blocked. As in Bitcoin, the rules for whether to apply a transaction into the Ripple network are publicly available. Potentially, every user can run one Ripple validator, a server that receives Ripple transactions from clients and forwards them to other validators to be added in the next run of the Ripple consensus. Therefore, if a validator secretly blocks a PathShuffle transaction, such transaction could still be added if sent to other validators. Alternatively, PathShuffle transactions could be explicitly blocked in the Ripple consensus rules, but this rule modification must be made public, thereby allowing to circumvent it and eventually starting a cat-and-mouse game.

4 At the time of writing, 20 XRP are necessary and they are worth about 13 US cents.

7 Summary

In this work, we present PathJoin, a novel protocol to perform atomic transactions in the Ripple network. The atomicity provided by PathJoin is of special interest not only for path mixing protocols, but also for several other applications such as crowdfunding.

We use PathJoin and DiceMix, the most efficient P2P mixing protocol existing in the literature, to build PathShuffle, a practical decentralized path mixing protocol for credit networks. PathShuffle requires only five communication rounds independently on the number of users and $5+3f$ in the presence of f misbehaving users.

We have implemented PathShuffle and carried out a path mixing transaction among five users in the currently deployed Ripple network, thereby demonstrating the practicality of PathShuffle to provide anonymous transactions in credit networks.

References

[1] Cryptocurrency market capitalization. http://coinmarketcap.com/currencies/views/all.

[2] Drop the concept of rippling in favor of long lived offers. GitHub Issue. https://github.com/stellar/stellar-protocol/issues/6.

[3] Linking wallets and deanonymizing transactions in Ripple. Thread in XRPChat. https://www.xrpchat.com/topic/1721-linking-wallets-and-deanonymizing-transactions-in-ripple.

[4] List of operations. In: Stellar Documentation. Stellar Development Foundation. https://www.stellar.org/developers/guides/concepts/list-of-operations.html.

[5] New payment solution with blockchain technology. SEB. https://sebgroup.com/press/news/new-payment-solution-with-blockchain-technology.

[6] Reserves. In: Ripple Documentation. Ripple Inc. https://ripple.com/build/reserves.

Unauthenticated
Download Date | 2/21/18 6:24 PM

[7] Ripple allows payments to any Bitcoin address straight from its client. https://gigaom.com/2013/07/02/ripple-allows-payments-to-any-bitcoin-address-straight-from-its-client.

[8] Ripple Charts. Ripple Inc. https://charts.ripple.com/.

[9] ripple-lib. A JavaScript API for interacting with Ripple in Node.js. https://github.com/ripple/ripple-lib.

[10] Ripple Strikes Multi-National Deal with SBI Holdings to Meet Growing Demand for Ripple Solutions Across Asia. https://ripple.com/ripple_press/ripple-strikes-multi-national-deal-with-sbi-holdings-to-meet-growing-demand-for-ripple-solutions-across-asia.

[11] Ripple website. Ripple Inc. https://ripple.com.

[12] Stellar network. Stellar Development Foundation. https://www.stellar.org/.

[13] Stellar partnerships. Stellar Development Foundation. https://www.stellar.org/blog/category/partnerships/.

[14] Understanding the "NoRipple" flag. In: Ripple Documentation. Ripple Inc. https://ripple.com/knowledge_center/understanding-the-noripple-flag/.

[15] Androulaki, E., Karame, G. O., Roeschlin, M., Scherer, T., and Capkun, S. Evaluating user privacy in Bitcoin. FC'13.

[16] Armknecht, F., Karame, G. O., Mandal, A., Youssef, F., and Zenner, E. Ripple: Overview and outlook. TRUST'15.

[17] Barber, S., Boyen, X., Shi, E., and Uzun, E. Bitter to better. how to make Bitcoin a better currency. FC'12.

[18] Ben-Sasson, E., Chiesa, A., Garman, C., Green, M., Miers, I., Tromer, E., and Virza, M. Zerocash: Decentralized anonymous payments from Bitcoin. S&P'14.

[19] Bernstein, D. J., Duif, N., Lange, T., Schwabe, P., and Yang, B. High-speed high-security signatures. *J. Cryptographic Engineering 2*, 2 (2012), 77–89.

[20] Bissias, G., Ozisik, A. P., Levine, B. N., and Liberatore, M. Sybil-resistant mixing for Bitcoin. In *WPES'14*.

[21] Bonneau, J., Narayanan, A., Miller, A., Clark, J., Kroll, J., and Felten, E. Mixcoin: Anonymity for Bitcoin with accountable mixes. In *FC'14*.

[22] Brickell, E. F., Lee, P. J., and Yacobi, Y. Secure audio teleconference. In *CRYPTO'87*.

[23] Corrigan-Gibbs, H., and Ford, B. Dissent: Accountable anonymous group messaging. CCS'10.

[24] Das, S. Ripple blockchain payment from Canada to Germany takes 20 seconds. https://www.cryptocoinsnews.com/ripple-blockchain-payment-transfer.

[25] Dolev, D., Reischuk, R., and Strong, H. R. Early stopping in byzantine agreement. *J. ACM 37*, 4 (1990), 720–741.

[26] Elison, M. Santander and Reisebank both recognized for innovation. Ripple Inc. https://ripple.com/insights/bankawards.

[27] Fugger, R. Money as IOUs in social trust networks & a proposal for a decentralized currency network protocol. http://archive.ripple-project.org/decentralizedcurrency.pdf, 2004.

[28] Gavin Wood. Ethereum: a secure decentralised generalised transaction ledger. http://gavwood.com/paper.pdf.

[29] Ghosh, A., Mahdian, M., Reeves, D. M., Pennock, D. M., and Fugger, R. Mechanism design on trust networks. In *WINE'07*.

[30] Heilman, E., Baldimtsi, F., Alshenibr, L., Scafuro, A., and Goldberg, S. TumbleBit: An untrusted tumbler for Bitcoin-compatible anonymous payments. In *NDSS'17*.

[31] Heilman, E., Baldimtsi, F., and Goldberg, S. Blindly signed contracts: Anonymous on-blockchain and off-blockchain Bitcoin transactions. In *FC'16*.

[32] Holley, E. Earthport launches distributed ledger hub via Ripple. http://www.bankingtech.com/420912/earthport-launches-distributed-ledger-hub-via-ripple/, 2016.

[33] Horster, P., Michels, M., and Petersen, H. Meta-multisignature schemes based on the discrete logarithm problem. In *IFIP/Sec'95*.

[34] Karlan, D., Mobius, M., Rosenblat, T., and Szeidl, A. Trust and social collateral. *The Quarterly Journal of Economics 124*, 3 (2009), 1307–1361.

[35] Kosba, A. E., Miller, A., Shi, E., Wen, Z., and Papamanthou, C. Hawk: The blockchain model of cryptography and privacy-preserving smart contracts. In *S&P'16*.

[36] Koshy, P., Koshy, D., and McDaniel, P. An analysis of anonymity in Bitcoin using P2P network traffic. In *FC'14*.

[37] Liu, A. Santander: Distributed ledger tech could save banks $20 billion a year. Ripple Blog. https://ripple.com/blog/santander-distributed-ledger-tech-could-save-banks-20-billion-a-year, 2015.

[38] Malavolta, G., Moreno-Sanchez, P., Kate, A., and Maffei, M. SilentWhispers: Enforcing security and privacy in decentralized credit networks. In *NDSS'17*.

[39] Maxwell, G. CoinJoin: Bitcoin privacy for the real world. Post in Bitcoin Forum. https://bitcointalk.org/index.php?topic=279249.

[40] Meiklejohn, S., and Orlandi, C. Privacy-enhancing overlays in Bitcoin. In *BITCOIN'15*.

[41] Meiklejohn, S., Pomarole, M., Jordan, G., Levchenko, K., McCoy, D., Voelker, G. M., and Savage, S. A fistful of bitcoins: Characterizing payments among men with no names. IMC'13.

[42] Miers, I., Garman, C., Green, M., and Rubin, A. D. Zerocoin: Anonymous distributed e-cash from Bitcoin. In *S&P'13* (2013).

[43] Moreno-Sanchez, P. PathJoin implementation. https://github.com/pedrorechez/PathJoin.

[44] Moreno-Sanchez, P., Kate, A., Maffei, M., and Pecina, K. Privacy preserving payments in credit networks. In *NDSS'15*.

[45] Moreno-Sanchez, P., Zafar, M. B., and Kate, A. Linking wallets and deanonymizing transactions in the Ripple network. In *PETS'16*.

[46] Nakamoto, S. Bitcoin: A peer-to-peer electronic cash system. https://bitcoin.org/bitcoin.pdf, 2008.

[47] Red. Introducing Goodwill. Post in Ripple Forum. https://forum.ripple.com/viewtopic.php?t=2895.

[48] Reid, F., and Harrigan, M. An analysis of anonymity in the Bitcoin system. In *SPSN'11*.

[49] Ripple. Ripple implementation. https://github.com/ripple.

[50] Rizzo, P. Royal Bank of Canada Reveals Blockchain Trial With Ripple. CoinDesk. http://www.coindesk.com/royal-bank-canada-reveals-blockchain-remittance-trial-ripple, 2016.

[51] Ruffing, T., Moreno-Sanchez, P., and Kate, A. CoinShuffle: Practical decentralized coin mixing for Bitcoin.

Unauthenticated
Download Date | 2/21/18 6:24 PM

ESORICS'14.

[52] Ruffing, T., Moreno-Sanchez, P., and Kate, A. P2P mixing and unlinkable Bitcoin transactions. In *NDSS'17*.

[53] Schnorr, C. P. Efficient signature generation by smart cards. *J. Cryptol. 4*, 3 (Jan. 1991), 161–174.

[54] Schwartz, D., Youngs, N., and Britto, A. The Ripple protocol consensus algorithm. https://ripple.com/files/ripple_consensus_whitepaper.pdf.

[55] Serjantov, A., Dingledine, R., and Syverson, P. F. From a trickle to a flood: Active attacks on several mix types. In *IH'02*.

[56] Southurst, J. Australia's Commonwealth bank latest to experiment with Ripple. CoinDesk. http://www.coindesk.com/australia-commonwealth-bank-ripple-experiment, 2015.

[57] Spagnuolo, M., Maggi, F., and Zanero, S. Bitlodine: Extracting Intelligence from the Bitcoin Network. In *FC'14*.

[58] Srikanth, T. K., and Toueg, S. Simulating authenticated broadcasts to derive simple fault-tolerant algorithms. *Distributed Computing 2*, 2 (1987), 80–94.

[59] Steven Goldfeder, Rosario Gennaro, Harry Kalodner, Joseph Bonneau, Joshua A. Kroll, Edward W. Felten and Arvind Narayan. Securing Bitcoin wallets via a new DSA/ECDSA threshold signature scheme. https://www.cs.princeton.edu/~stevenag/threshold_sigs.pdf.

[60] Syta, E., Tamas, I., Visher, D., Wolinsky, D. I., Jovanovic, P., Gasser, L., Gailly, N., Khoffi, I., and Ford, B. Keeping authorities "honest or bust" with decentralized witness cosigning. In *S&P'16*.

[61] Valenta, L., and Rowan, B. Blindcoin: Blinded, accountable mixes for Bitcoin. In *BITCOIN'15*.

[62] van Saberhagen, N. CryptoNote, 2013. https://cryptonote.org/whitepaper.pdf.

[63] Ziegeldorf, J. H., Grossmann, F., Henze, M., Inden, N., and Wehrle, K. CoinParty: Secure multi-party mixing of bitcoins. In *CODASPY'15*.

[64] Zyskind, G., Nathan, O., and Pentland, A. Enigma: Decentralized computation platform with guaranteed privacy.

A PathShuffle: Mixing XRP

The XRP currency in Ripple serves the purpose to protect the network from abuse and DoS attacks. A Ripple wallet needs to hold XRP for two reasons: the wallet is considered active only if it has a certain amount of XRP; moreover, the issuer of any transaction must pay a transaction fee in XRP.

As opposed to path-based transactions, XRP can be transferred directly from one wallet to another without a path. Assume that a user with wallet u wants to pay β XRP to some wallet v and that u has at least β XRP in her XRP balance. Then β XRP are removed from the XRP balance of u and added to the XRP balance of v. Notice that this type of transaction does not require the existence of any (direct or indirect) credit line between the sender and the receiver of the transaction.

A.1 Key Ideas

The key challenge for mixing XRP is to send funds *atomically* from n input wallets to n output wallets, essentially emulating a multi-input-multi-output transaction not natively supported by Ripple.

Naively Using Two Shared Wallets. A naive approach consists in using two shared wallets to perform the mixing of n XRP payments in a similar manner to what we defined for shuffling n path-based payments (see Section 5). There is however an important subtlety that appears when using XRP payments: a payment from an input wallet to the wallet vk^s_{in} implies actual sending of XRP. However, when XRP are sent to vk^s_{in} and a user disconnects, the XRP in vk^s_{in} are stuck, preventing the shuffling from finishing and forcing other users to lose their XRP. Thus, ensuring correct balance is a challenge in this scenario.

Use of Recovery Transactions. It is possible to create in advance (not yet applied) transactions from vk^s_{in} to every user's input wallet. Then, the user knows she can get her XRP back from vk^s_{in} if the mixing is not completed by submitting this recovery transaction to the Ripple network. Given that, she can safely send XRP from her input wallet's to vk^s_{in}.

Chaining and Tagging Recovery Transactions. However, recovery transactions introduce a different problem: When the recovery transaction is created for the second user, she can use it to steal the XRP from vk^s_{in} previously sent by the first user. To overcome this problem, we need to enforce an ordering for recovery transactions.

In the Ripple network, every transaction has a sequence number associated to it. A transaction with sequence number s performed by a given wallet is only valid if the last transaction performed by the same wallet has sequence number $s-1$. Using sequence numbers, it is possible to ensure that recovery transactions are executed in the correct order: The first user gets a recovery transaction with sequence number 1 and the second user gets recovery transaction with sequence number 2. This, however, does not totally solve the correct balance problem yet: The second user can only recover his XRP if the first one submits her recovery transaction. Otherwise, all XRP are locked at vk^s_{in}.

We can fully solve the correct balance problem as follows. The first user gets a recovery transaction with

Unauthenticated
Download Date | 2/21/18 6:24 PM

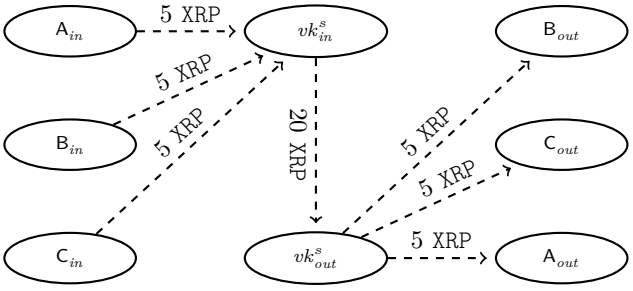


Fig. 5. The transactions in an example run of XRPJoin protocol to mix 5 XRP among three participants.

sequence number 1 and tagged with her identifier, i.e., the verification key of her input wallet. The second user gets two recovery transactions: One recovery transaction with sequence number 1 that returns the first user's XRP to her wallet, and a second recovery transaction with sequence number 2, that returns her own XRP; both recovery transaction are tagged with her identifier. This mechanism ensures that the second user recovers her own XRP even if the first user is going offline. Moreover, this mechanism ensures termination: given that transactions are tagged, a misbehaving user maliciously recovering his XRP can be easily detected.

A.2 Protocol Overview

Here we give an overview of a protocol XRPJoin that emulates atomic multi-input-multi-output XRP transactions, similar to what PathJoin does for path-based IOU transaction. By replacing PathJoin in PathShuffle by XRPJoin, one obtains a P2P protocol for XRP mixing.

In the following, we describe the protocol steps by means of an example. For simplicity, assume that there are three participants Alice, Bob, and Carol with Ripple wallets A_{in}, B_{in}, and C_{in}, willing to mix 5 XRP. In this setting, XRPJoin works as depicted in Fig. 5. In detail:

1. **Create Shared Wallets.** The users jointly generate two shared wallets vk^s_{in} and vk^s_{out} using a distributed signature scheme as in Appendix B.
2. **Create Transactions.** The users jointly create a transaction transferring 15 XRP from vk^s_{in} to vk^s_{out}. Moreover, they create transactions sending 5 XRP from vk^s_{out} to every participant's output wallet. These transactions are all signed by all participants using the distributed signature algorithm. As vk^s_{in} and vk^s_{out} do not have any funds at this point, all of these transactions cannot be accepted into the Ripple ledger yet.
3. **Transfer Input XRP to vk^s_{in}.** The main idea of this step is that every user first creates a transaction to recover her funds from the vk^s_{in}. After the transaction is correctly signed by every other user, she sends her funds to vk^s_{in}. In detail:
   (a) The users jointly create a recovery transaction sending 5 XRP from vk^s_{in} to A_{in}. The transaction is tagged with A_{in} and has sequence number 1. It is partially signed (using the distributed signature scheme) by all users except Alice, and it is not submitted to the Ripple network.
   (b) Alice creates a transaction sending 5 XRP from A_{in} to vk^s_{in}, and broadcasts it to the other users. All of the users submit the transaction to the Ripple network. (Note that Alice could also submit the recovery transaction created in the previous step. However, other users will see the tag and blame Alice of misbehavior.)
   (c) The users jointly create two recovery transactions with sequence numbers 1 and 2, sending 5 XRP from vk^s_{in} to A_{in} and to B_{in}, respectively. Both transactions are tagged with B_{in}. They are partially signed (using the distributed signature scheme) by all users except Bob, and they are not submitted to the Ripple network.
   (d) Bob creates a transaction sending 5 XRP from B_{in} to vk^s_{in}, and broadcasts it to the other users. All of the users submit the transaction to the Ripple network.
   (e) The users jointly create three recovery transactions with sequence numbers 1, 2, and 3, sending 5 XRP from vk^s_{in} to A_{in}, B_{in}, and C_{in}, respectively. All three transactions are tagged with C_{in}. They are partially signed (using the distributed signature scheme) by all users except Carol, and they are not submitted to the Ripple network.
   (f) Carol creates a transaction sending 5 XRP from C_{in} to vk^s_{in}, and broadcasts it to the other users. All of the users submit the transaction to the Ripple network.
4. **Perform Payments or Blame.** The users continue depending on the outcome of step 3.
   (a) If step 3 has been successful, then the wallet vk^s_{in} holds 15 XRP. The users submit the transactions generated in step 2 of the protocol, so that every output wallet receives 5 XRP. The users wait for the Ripple network to confirm the transaction from vk^s_{in} to vk^s_{out} and the transactions from vk^s_{out} to each of the output wallets. If they are confirmed, then the protocol was successful.

Unauthenticated
Download Date | 2/21/18 6:24 PM

If instead a malicious user submits one of her recovery transactions to the Ripple network and this is confirmed before the transaction from vk^s_{in} to vk^s_{out}, then the other users will blame and exclude her, and a new run of the protocol with fresh output addresses is started (following the execution pattern of PathShuffle). Note that the recovery transactions are tagged such that it is obvious to the honest users who signed and submitted the recovery transaction.

(b) If step 3 has not been successful, then at least one malicious user has refused to send a message (or has sent an unexpected message). This is detectable and the honest users will blame the malicious user and exclude her, and a new run of the protocol with fresh output addresses is started (following the execution pattern of PathShuffle). Moreover, the honest users can use their recovery transactions (in the right order) to recover their funds from vk^s_{in}.

B Handling Shared Accounts using Distributed Signatures

A wallet vk^s can be shared among a set of n users so that only when all users agree, a transaction involving the shared wallet vk^s is performed. We use a distributed signature scheme to achieve this functionality.

Distributed Signature Scheme. Currently Ripple supports two digital signature schemes ECDSA and EdDSA [19]. Although there exist distributed versions for ECDSA [59], we choose EdDSA as it is similar to the Schnorr signature scheme [53] and thus offers a simpler and more efficient distributed variant [22, 33].

In our specific setting, a distributed signature scheme has a verification algorithm (that of EdDSA) and two algorithms SAccountCombine and SSign. The SAccountCombine algorithm takes as input the tuple $(\mathrm{VK}^*[\,], my, P)$, where $\mathrm{VK}^*[\,]$ is the set of shares from the user, my is the invoking user's identifier and P is the set of other users to be included in the shared wallet. Then, the SAccountCombine algorithm returns the combined public key vk^s for the shared wallet.

The SSign algorithm takes as input the tuple $(P, my, \mathrm{VK}_{in}[\,], sk_{in}, \mathrm{VK}^*[\,], sk^*, m, sid)$, where P, my and $\mathrm{VK}^*[\,]$ are defined as before, $\mathrm{VK}_{in}[\,]$ is a set of verification keys associated to the users, sk_{in} is the signing key for the user's $\mathrm{VK}_{in}[my]$, sk^* is the secret key for the user's share, m is the message to be signed and sid is the session identifiers for the current session. Then, the SSign algorithm returns a signature σ on message m verifiable with the shared public key vk^s if no user disrupts the protocol or a set P_{mal} with the malicious users.

Algorithm 3. A distributed signature scheme for EdDSA

procedure SAccountCombine($\mathrm{VK}^*[\,], my, P$)
  return $\sum_{p\in P\cup\{my\}} \mathrm{VK}^*[p]$

procedure SSign($P, my, \mathrm{VK}_{in}[\,], sk_{in}, \mathrm{VK}^*[\,], sk^*, m, sid$)
  $k \xleftarrow{\$} \mathbb{Z}_q$
  $\mathrm{R}[my] := g^k$
  broadcast $(\mathrm{R}[my], \mathsf{Sign}(sk_{in}, (\mathrm{R}[my], sid)))$
  receive $(\mathrm{R}[p], \sigma[p], \sigma'[p])$ **from all** $p \in P$
    where $\mathsf{Verify}(\mathrm{VK}_{in}[p], \sigma[p], (\mathrm{R}[p], sid))$
  missing P_{off} **do**
    return $(\perp, P_{off})$
  $r := \sum_{p\in P\cup\{my\}} \mathrm{R}[p]$
  $vk^s := \mathsf{SAccountCombine}(\mathrm{VK}^*[\,], my, P)$
  $h := \mathsf{H}(r, vk^s, m)$
  $\mathrm{S}[my] := k + h \cdot sk^*$
  broadcast $\mathrm{S}[my]$
  receive $\mathrm{S}[p]$ **from all** $p \in P$
    where $g^{8\mathrm{S}[p]} = \mathrm{R}[p]^8 \cdot (\mathrm{VK}^*[p])^{8h}$
  missing P_{off} **do**
    return $(\perp, P_{off})$
  $\sigma := (r, \sum_{p_\in P\cup\{my\}} \mathrm{S}[p])$
  return $(\sigma, \emptyset)$

We describe the details of SAccountCombine and SSign in Algorithm 3. The protocol relies on a simple additive secret sharing, and is adapted to the specifics of EdDSA on the employed elliptic curve, e.g., the cofactor 8 [19]. Our description, here, follows the same notation as the rest of the paper (see Section 4). In particular, we stress that the algorithms Sign and Verify are defined as in the rest of the paper, i.e., they belong to the signature algorithm used by the input wallet, which is not necessarily EdDSA.

Note that this distributed signature scheme is vulnerable to a related-key attack pointed out by Horster, Michels, and Petersen [33]. A malicious rushing peer p can choose his verification key share $VK^*[p]$ such that he can create a signature valid under the combined verification key vk^s without the help of the other users; however, she cannot provide a partial signature under her verification key share. This attack is typically avoided by forcing everybody to prove the knowledge of the discrete logarithm of the verification key share $VK^*[p]$, e.g., us-

Unauthenticated
Download Date | 2/21/18 6:24 PM

ing a zero-knowledge proof or an ordinary signature on p valid under the verification key share $VK^*[p]$ [33, 60].

In our specific use of distributed signatures in PathJoin, we do not need this (additional) proof. A malicious user performing this attack will be caught by the honest users before she can do any damage; the malicious user cannot provide a valid partial signature for any of the transactions that create the links from the user's input wallets to the input shared wallet vk^s_{in} (see Algorithm 1). Therefore, the malicious user will be detected and excluded by honest users and the protocol run will be aborted. At this point, although the attacker indeed controls the shared wallet vk^s_{in}, it is not worth any IOU as honest users do not issue any credit.

C Testing PathShuffle in the Ripple Network

In Table 1, we describe the Ripple wallets that we have used in our simulation of PathShuffle within the currently deployed Ripple network. In particular, we show the mapping between wallets in the example depicted in Fig. 4 and the corresponding wallets in our experiment in the Ripple network.

Figure wallets	Ripple network wallets
A_{in}	rMDvFAhSPYEaUNxqrqo88xCwiZXG9P3LNK
B_{in}	rBnkLfFPvabAhEXnBH76UMoyygGUQeYZA3
C_{in}	rhgJUmYMAwQjq5mtnRyFE74fu8sKPf1Nmn
D_{in}	rs6MgypVJgpdDuFk5X8mwoywHezN1gh91Y
E_{in}	rDWuswR94qHuFD6GsGWxtujpZkSg3sNZnX
A_{out}	rK2ByNQvPQBEe1r2WHasY53Z7Tj9ZJrjr8
B_{out}	r9euchAFnRqYJDBngmKD4tXhuLEAcHtCRK
C_{out}	rz2TRTy1Y7b4t1ZEaG98s7brwwTXmi97f
D_{out}	rE6sMibroiWdZADCUqPmEemFeaohVJYWuR
E_{out}	rhjY2JshgYCgkhDVK3jSfZhiqQ3ZKsWUU8
vk^s_{in}	rM3U73YQWd4ewijyEsieDWaLf2ektvMmoK
vk^s_{out}	r3AJt7VUhK8e9BaBemKydfmHuLPZx2ibZd
vk_{gw}	rPBCgQXeXvcPhUzto8YhGoDYG9n9V6owRR

Table 1. Mapping between wallets in Fig. 4 and the same wallets in the Ripple network for our compatibility test. The hash for the final transaction from vk^s_{in} to vk^s_{out} is 21BCE61D6843F23D9A02D745AB788CFF679C6E99ECF70D56C141ACB8560AA370.

Unauthenticated
Download Date | 2/21/18 6:24 PM