8-2018

Credit Network Payment Systems: Security, Privacy and Decentralization

Pedro Moreno Sanchez
Purdue University

CREDIT NETWORK PAYMENT SYSTEMS: SECURITY, PRIVACY AND DECENTRALIZATION

A Dissertation

Submitted to the Faculty

of

Purdue University

by

Pedro Moreno Sanchez

In Partial Fulfillment of the

Requirements for the Degree

of

Doctor of Philosophy

August 2018

Purdue University

West Lafayette, Indiana

THE PURDUE UNIVERSITY GRADUATE SCHOOL
STATEMENT OF DISSERTATION APPROVAL

Dr. Aniket Kate, Chair
Department of Computer Science, Purdue University
Dr. Elena Grigorescu
Department of Computer Science, Purdue University
Dr. Ninghui Li
Department of Computer Science, Purdue University
Dr. Sonia Fahmy
Department of Computer Science, Purdue University
Dr. Matteo Maffei
Department of Computer Science, TU Wien

Approved by:

Dr. Voicu Popescu, by Dr. William J. Gorman
Head of the Department Graduate Program

A mis padres por todo su apoyo y ánimos para que consiga la mejor educación posible. Yo realmente aprecio los sacrificios inconmensurables que han hecho en su vida para hacer posible que yo escriba esta disertación. Gracias papá, gracias mamá.

ACKNOWLEDGMENTS

I heartfully want to thank my supervisor Aniket Kate. Apart from his indisputable research skills, I would like to highlight his honest and invaluable efforts to make me the best researcher possible and a better person. They have been crucial during my whole PhD journey. Learning from him during this time over two different continents has been an invaluable and unforgettable experience that will follow me the rest of my life. Thank you.

This dissertation would not have been possible without my collaborators. I would like to specially thank Giulio Malavolta and Tim Ruffing. They are undoubtedly the best co-authors and, overall friends, that one can have during his PhD journey. Also special thanks to Matteo Maffei, his support during these years has been of vital importance. Finally, I am really thankful to the selfless help from Ian Goldberg, Sonia Fahmy, Navin Modi, Kim Pecina, Srivatsan Ravi, Stefanie Roos, Raghuvir Songhela and Muhammad Bilal Zafar.

I feel really lucky to have met numerous friends during these years in both Saarbruecken and West Lafayette. This dissertation would be endless if I mention all of them here and I firmly believe that the time spent with them is among the most enriching experiences during this journey. From my time in Saarbruecken, I will never forget the long movie nights at Aurela and John's house where they, along with Harrys and Yuliya, attempted to improve my English pronunciation. The vibrant and long discussions with Mariona are also unforgettable. My time in West Lafayette has also brought good friends and enriching experiences. In special, I want to thank Glebys and Maria Leonor for the many moments we have spent together, my favorite being the day where we showed to everybody that canoeing is not that easy as it looks.

Special thanks also to Sze Yiu for suffering me as housemate and to all the colleagues at the Freedom Lab for the endless geek conversations.

Last, but definitely not least, I would like to thank my father Antonio Jesus, my mother Herminia and my brother Antonio for their continuous support. Their unconditional encouragement to pursue and achieve all the goals I set in my life has been crucial to be writing this dissertation. *Papá, mamá, hermano, lo he conseguido y vuestro apoyo ha sido fundamental. Muchas gracias!*

TABLE OF CONTENTS

Page

Page

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

Moreno Sanchez, Pedro PhD, Purdue University, August 2018. Credit Network Payment Systems: Security, Privacy and Decentralization. Major Professor: Aniket Kate.

A credit network models transitive trust between users and enables transactions between arbitrary pairs of users. With their flexible design and robustness against intrusions, credit networks form the basis of Sybil-tolerant social networks, spam-resistant communication protocols, and payment settlement systems. For instance, the Ripple credit network is used today by various banks worldwide as their backbone for cross-currency transactions. Open credit networks, however, expose users' credit links as well as the transaction volumes to the public. This raises a significant privacy concern, which has largely been ignored by the research on credit networks so far.

In this state of affairs, this dissertation makes the following contributions. First, we perform a thorough study of the Ripple network that analyzes and characterizes its security and privacy issues. Second, we define a formal model for the security and privacy notions of interest in a credit network. This model lays the foundations for secure and privacy-preserving credit networks. Third, we build PathShuffle, the first protocol for atomic and anonymous transactions in credit networks that is fully compatible with the currently deployed Ripple and Stellar credit networks. Finally, we build SilentWhispers, the first provably secure and privacy-preserving transaction protocol for decentralized credit networks. SilentWhispers can be used to simulate Ripple transactions while preserving the expected security and privacy guarantees.

1 INTRODUCTION

Credit networks [1–3] model transitive trust among users through pairwise credit allocations. A credit network user expresses trust on another user in the form of a credit value that she is willing to extend to the other user, and by indicating commitments to allow transactions across her different credit links. This enables that the credit network users perform transactions over paths consisting of several intermediate users. Moreover, by introducing suitable definitions of transaction, credit networks have been shown useful in a plethora of scenarios. In fact, sybil-tolerant and spam-resistant systems based on the concept of credit network have been proposed in the last few years, such as Bazaar [4] and Ostra [5]. Prominently, credit networks are also leveraged in two growing payment settlement systems: Ripple [6] and Stellar [7].

The Ripple Network The Ripple network has seen a widespread adoption as an interesting alternative to avoid large fees charged by intermediate banks today while performing world-wide transactions. The Kansas-based CBW Bank and Cross River Bank [8], the Royal Bank of Canada [9] or Santander [10] are a few examples of banks using the Ripple network after exploring the numerous available blockchain options [11–14]. Companies are also using advantages of Ripple (e.g., fast and low-cost international transactions) to build better cross-border payment services. For example, Earthport [15, 16] has adopted Ripple to perform transactions between banks over several countries over the world, while Saldo.mx uses the Ripple network to improve cross-border transactions between USA and Mexico [17]. Moreover, Microsoft has partnered with Ripple to use part of its Azure BaaS to contribute to the execution of the Ripple network [18].

1.1 Challenges in Credit Networks

The Privacy Challenge Most of the credit network designs proposed so far (e.g., Bazaar and Ostra) are centralized, i.e., the credit network is maintained entirely in a server environment. The others, such as Ripple and Stellar, make their entire sets of transactions as well as the credit network topology visible in a publicly available log (i.e., the blockchain) to establish credibility through transparency. As a result, credit networks today cannot provide any meaningful *privacy* guarantee. This state of affairs clearly conflicts with the desire of users, who instead strive for hiding their credit links and their transactions: Businesses and customers are interested in hiding their credit information and transactions from competitors and even service providers, while regular users aim to protect their transactions as they might reveal personal information (e.g., medical bills or salary).

Designing a privacy-preserving solution for credit networks is technically challenging. Simple anonymization methods such as the pseudonyms employed in Ripple are ineffective, as all transactions remain linkable to each other and they are susceptible to deanonymization attacks [19]. Other techniques proposed in academia [20–23] do not fully solve the problem either. For instance, providing the server environment only with the topological network graph while keeping credit values private opens the system up to correlation attacks that ultimately reveal the partners' real identities [20, 21]. Perturbing the links or their credit values by means of differential privacy techniques [22, 23] would yield stronger privacy guarantees, but this is often unacceptable in payment scenarios as it implies unexpected redistributions of credit.

The Decentralization Challenge A natural direction to overcome the privacy challenge consists in envisioning a decentralized credit network, where each user locally stores and maintains her own credit links. A decentralized credit network design fits better the nature of certain applications such as the current financial ecosystem, where each user is responsible for her own credit while each financial institution is responsible for the credit with its customers. However, building a decentralized credit network

is not straightforward. There are several credit network operations that are trivially solved in a centralized setting but become challenging in a decentralized design.

For instance, while a centralized service provider can easily determine a transaction path between two users, it is not trivial to define how the routing information is spread along a decentralized credit network. Similarly, a decentralized setting requires finding out how much credit is available on the paths between any two users to perform transactions. Finally, a decentralized setting requires a solution to ensure the correctness of the transactions, while maintaining the credit balances of honest users in the presence of malicious and offline users.

The Security Challenge Designing a secure solution for credit networks becomes challenging in a decentralized setting, where every user maintains her own credit links and relies on other users to carry out her transactions, some of which may behave arbitrarily. For instance, a transaction over a path can be easily disrupted by a malicious intermediate user aiming at a credit benefit or by a (possibly honest) user that simply goes offline. Such behavior can lead to credit losses by honest users as well as severely hamper the availability of the overall credit network. A decentralized credit network must thus ensure atomic payments so that every credit link in a path is correctly updated to carry out a successful transaction or no credit link is modified otherwise. A decentralized credit network should ensure thereby that no honest user incurs credit loss.

1.2 Our Contributions

This dissertation focusses on demonstrating the following statement:

> *Current credit network deployments provide limited guarantees in terms of security, privacy or decentralization. It is possible to build a secure, privacy-preserving and decentralized credit network system.*

In the following, we briefly describe our contributions to demonstrate the veracity of this statement.

A Security and Privacy Study of the Ripple Network In Chapter 2, we present the first thorough study that analyzes the globally visible blockchain in the Ripple network and characterizes the security and privacy issues related to it. In particular, we have extensively studied the effect of unexpected redistribution of credit, the effect of faulty gateways and the effect of stale offers and shown their consequences in terms of credit loss by Ripple users. Regarding privacy, we define two novel heuristics and perform clustering to group wallets owned by the same user. We then propose reidentification mechanisms to deanonymize the operators of those clusters and show how to reconstruct the financial activities of deanonymized wallets.

Security and Privacy Definitions for Credit Networks In Chapter 3, we lay the foundations for security and privacy in credit networks, presenting a definitional framework. In particular, we formalize the notions of integrity, value privacy and sender/receiver privacy. Intuitively, we say that a credit network maintains integrity if no honest user loses credit when cooperating as intermediate user in a payment path. Moreover, we say that a credit network maintains value privacy if the adversary cannot determine the value of a transaction between two non-compromised users. Finally, we say that a credit network maintains sender (correspondingly receiver) privacy if the adversary cannot determine the actual sender (receiver) of a transaction.

PathShuffle: Anonymous Transactions in the Ripple Network In Chapter 4, we present PathShuffle, the first privacy-enhancing protocol compatible with the Ripple network. As its essential building block, we propose a novel technique to perform atomic transactions in credit networks and extend it to build a decentralized protocol for anonymous transactions. We demonstrate the practicality of PathShuffle by executing an instance of the protocol in the current Ripple network. This protocol thereby provides a functionality otherwise missing in the current Ripple network.

SilentWhispers: A Decentralized Architecture to Enforce Security and Privacy in Credit Networks In Chapter 5, we present SilentWhispers, the first decentralized and provably secure and privacy-preserving transaction protocol for credit networks. Partial solutions like PathShuffle are an interesting approach for raising the privacy bar in the currently deployed Ripple network. However, the distinguishing feature of SilentWhispers is the assurance of strong security and privacy guarantees in credit networks without requiring a global, transparent and publicly accessible blockchain. SilentWhispers can be used to simulate the Ripple transactions in real time and therefore can be deployed as an online credit network.

1.3 Outline of the Dissertation

This dissertation is organized in two parts. Part I includes our study and definitional work on security and privacy for credit networks and comprises Chapter 2 and Chapter 3. In particular, in Chapter 2 we describe the background on the Ripple network and our security and privacy study of the Ripple network; and in Chapter 3 we propose the security and privacy notions of interest in a credit network.

Part II focuses on the description of our novel systems that provide secure and privacy-preserving transactions in credit networks and comprises Chapter 4 and Chapter 5. In Chapter 4 we describe PathShuffle, the privacy-enhancing protocol fully compatible with the current Ripple network. In Chapter 5 we detail SilentWhispers, a decentralized architecture that provides strong security and privacy guarantees for transactions in credit networks.

Finally, we summarize this dissertation in Chapter 6.

Part I

SECURITY AND PRIVACY IN CREDIT NETWORKS: STUDY AND DEFINITIONS

2 REAL-WORLD CASE STUDY: RIPPLE

The properties inherent to a credit network have been found of great utility in a plethora of applications. In particular, several academic efforts [4,5,24–27] have shown that by adapting the notion of credit to the specifics of a certain application, it is feasible to come up with a tailored credit network to enforce key functionalities for such application. Nevertheless, none of these academic proposals have been deployed in practice at the time of writing. Existing deployments for a credit network have focussed so far on the *settlement of payments* between users. In such application, credit between two users is defined as the trust in each other in terms of the amount of *I Owe You* (IOU) funds they are willing to extend to each other. Building upon this concept of credit, a credit network can be leveraged to settle payments between any two users improving upon many of the drawbacks of the alternative systems available today (e.g., SWIFT) such as slowness, expensiveness and prone to thefts [28, 29].

Currently, Ripple and Stellar are the most prominent examples of credit networks in practice. The Stellar network is still in an early but expanding stage and it has got the support of a few financial institutions and payment aggregators, mostly focussed on developing countries. The Ripple network instead, has gained more traction and it has been tested and adopted by several major banks and financial institutions that see in the publicly available Ripple network an alternative to improve the processing of payments. For instance, the Spanish bank Santander has claimed that adopting Ripple could save them $20 billion a year [10]. The Kansas-based CBW Bank and Cross River Bank [8] are the first American banks to adopt Ripple. The Royal Bank of Canada [9] has chosen Ripple over other settlement solutions after exploring the numerous available blockchain options. And these are just a few examples in a still

growing list [11–14]. As of August 2017, the Ripple network cates a user base of more than $180,000$ accounts, $350,000$ credit links and more than $29M$ transactions during the period January 2013 – August 2017.

The potential of Ripple is not limited to the traditional banking system. Other financial institutions are also using the advantages of Ripple to build better cross-border payment services. For example, Earthport [30] has chosen Ripple to perform cross-border transactions between more than 60 countries worldwide. Moreover, companies such as Microsoft and universities such as MIT are using part of its computational resources to contribute to the execution of the Ripple network [31, 32]. Given the wider development in practice, we use the Ripple network as case study of credit network in this dissertation. We note, however, that most of the descriptions apply also to the Stellar network as both build upon (mostly) the same principles. In the rest of this chapter, we overview the different components of the Ripple network. We then study the security and privacy of the Ripple network.

2.1 Description of the Ripple Network

The Ripple network heavily relies in three components as shown in Figure 2.1: a *credit graph*, a set of *operations* and a *consensus protocol*. The credit graph reflects the IOU relations among payment providers, financial institutions and customers among others. This financial network is updated by means of operations that allow to create new IOU relations, update existing ones, settle credit among any two users or provide currency exchange services, utterly necessary for liquidity and remittance. These transactions would be, however, useless without a way to globally agree on their validity. The Ripple consensus protocol provides a mean to create a blockchain that unequivocally stores the set of valid transactions, as agreed by a set of protocol players from all around the world in a decentralized fashion. In the rest of this section, we describe in detail each of the components separately.

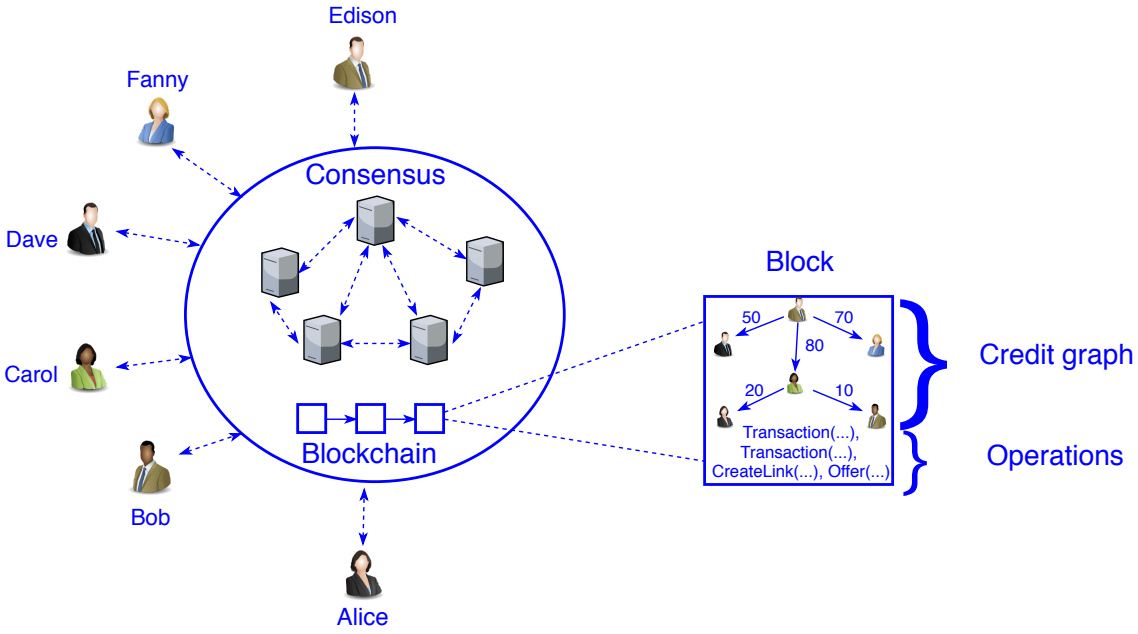


Fig. 2.1. Illustrative example of the Ripple ecosystem. Dashed arrows represent communication between parties. Filled arrows represent credit links. The *Ripple credit graph* represents wallets and credit links among them. The graph is updated by means of *operations*. The operations are submitted by the corresponding users to the participants in the *consensus protocol*. Only validated operations are added to the *blockchain* (or ledger in Ripple terms).

2.1.1 Credit Graph

At the core of the Ripple network lies a weighted, directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$. The set $\mathcal{V}$ of vertices represents the accounts (or wallets in Ripple terms) in the network, and the set $\mathcal{E}$ of directed and weighted edges represents the IOU credit links between wallets. A Ripple wallet is initialized with a pair of private (signing) and public (verification) keys. The wallet is then labeled with an encoding of the hashed public key. The knowledgeable reader might have noticed that a Ripple wallet is created and labeled similar to a cryptocurrency account such as Bitcoin.

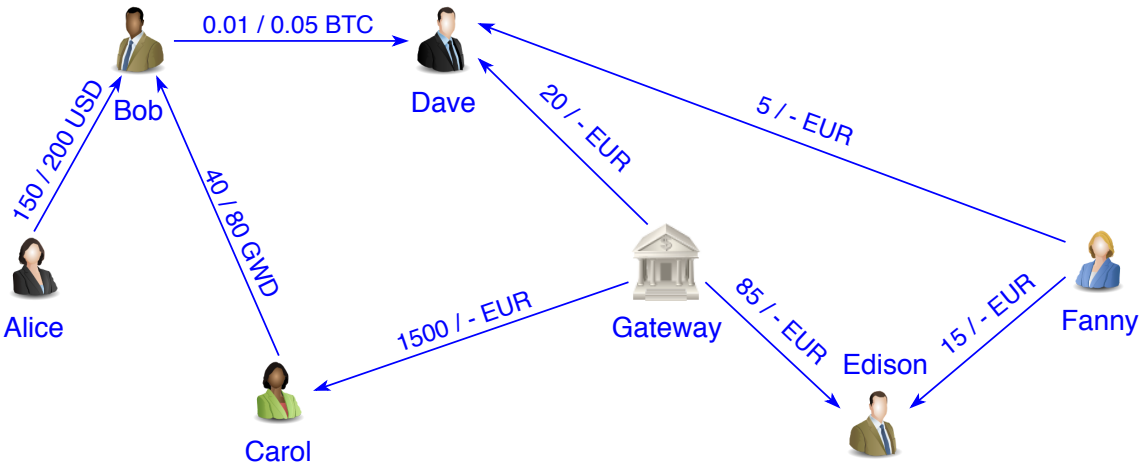


Fig. 2.2. An illustrative example of the Ripple credit graph. For readability, every wallet is represented by a name instead of the hashed public key used in practice. Edges represent credit links between pairs of connected nodes. The edges are labeled with tuples (x / y XYZ), where x denotes the balance, y denotes the upper bound (- represents a not upper-bounded credit link), and XYZ denotes the currency.

A directed edge $(\mathsf{u}_1, \mathsf{u}_2) \in \mathcal{E}$ is labeled with a tuple (*balance*, *currency*, *upper-limit*), where *balance* is a dynamic scalar value $\alpha_{\mathsf{u}_1\mathsf{u}_2}$ indicating the amount of *unconsumed* credit that wallet u_1 has extended to wallet u_2 (i.e., u_1 owes $\alpha_{\mathsf{u}_1\mathsf{u}_2}$ to u_2); *currency* denotes the denomination for such credit. The credit available on an edge is lower-bounded by 0 and is upper-bounded by ∞ by default, while a more strict upper bound can optionally be adopted by the creditor's wallet (i.e., u_2 in the previous example) by setting the *upper-limit* field. Additionally, every wallet has associated with it a positive amount of XRP.

Therefore, Ripple supports its own native cryptocurrency, called XRP. Contrary to other cryptocurrencies such as Bitcoin or Ethereum, where coins are continuously being created until a predefined limit number of coins is reached, a fixed amount of XRP were initially created and no more XRP will be ever created according to the current protocol description. The XRP currency was initially envisioned as a

mean to protect the network from abuse and Denial-of-Service (DoS) attacks. In this sense, the purpose of XRP is twofold. First, a wallet is considered active only if it has a certain amount of XRP, deterring thereby the creation of a massive number of wallets by a single user for spamming purposes. Second, each transaction requires a certain transaction fee in XRP, hindering thereby massive spamming transactions. Nevertheless, the financial usefulness of XRP is yet to be thoroughly studied.

An illustrative example of the Ripple network is shown in Figure 2.2. Here, consider that Alice and Bob are two users in the Ripple network. Further consider that Alice owes Bob 150 USD. This illustrative example is reflected in the Ripple network as one edge Alice → Bob with a balance of 150 USD. Note that the balance on an edge is tagged with the corresponding currency, as well as the upper bound. In this case, Bob allows Alice to owe him up to 200 USD. This value effectively limits the trust that Bob has on Alice. The rest of credit links can be interpreted in a similar manner.

In summary, the Ripple credit graph is a directed graph where credit links expressed in many different currencies coexist. It provides enough expressiveness to emulate the complex relations among the different players in a plethora of financial applications.

2.1.2 Key Roles

The Ripple network heavily relies on two roles played by some of the wallets: *gateway* and *market maker*. A *gateway* is a well-known reputed wallet established to create and maintain a credit link in a correct manner with new users. Gateways are therefore the counterparts of user-facing banks and loan agencies in the physical world. As an illustrative example, consider a new user that wants to join for the first time the Ripple network. For that, she would need to create a fresh wallet, create a new credit link and get some balance on it from another wallet in the system. This is known as *bootstrapping* problem and it is a common problem in many networks such as social networks or communication networks. The Ripple network solves this bootstrapping problem by introducing gateways. Therefore, the user can trust the gateway to create

a credit link and issue her first IOUs on such link. As wallets for gateways are highly connected wallets in the Ripple network, the thereby created credit link will allow the new wallet to interact with the rest of the Ripple network.

A *market maker* is a wallet that has credit links with balance denominated in more than one currency and performs a currency exchange service, that is, it receives a certain currency on one of its credit links and exchanges it for another currency on another of its credit links. Market makers are therefore the counterparts of physical currency exchange services. For instance, in the illustrative example depicted in Figure 2.2, Bob can act as a market maker by accepting the exchange among USD, BTC and GWD currencies. The role of market makers is crucial to provide liquidity and enable cross-currency transactions in the Ripple network.

2.1.3 Operations

The Ripple network graph is updated by means of *operations*. For the ease of exposition, we classify these operations in two groups: *single-wallet* and *multi-wallet* operations. In a nutshell, single-wallet operations update a single wallet and the credit links associated to it in the Ripple network. Single-wallet operations represent the counterpart of bank account management operations in the real world. Instead, multi-wallet operations may update several wallets and credit links among them to effectively represent the settlement of funds among wallets in the Ripple network.

The Ripple network supports several single-wallet operations. First, the *AccountSet* operation allows a user to update a wallet she possesses. Second, *OfferCreate* enables the creation of an exchange offer. An additional *OfferCancel* operation can be used to cancel a previously created offer. Finally, *TrustSet* operation allows the creation of a credit link between two wallets if such credit link does not exist yet, or the update of a credit link in case it already exists. For the sake of brevity, we omit a detailed description of these operations and refer the reader to [33] for further details.

Table 2.1.
Ripple transaction examples for both direct XRP payments and path-based settlement transactions. In the direct XRP payment, 20 XRP are sent from Alice to Bob. In the settlement transaction, 10 EUR are transferred from Dave to Edison via Gateway. The notation Alice denotes the Ripple wallet owned by Alice. Irrelevant transaction fields have been omitted.

	XRP Payment	**Path-based Settlement Transactions**
Sender	Alice	Dave
Receiver	Bob	Edison
Amount	20 XRP	10 EUR
Path	–	Dave ← Gateway → Edison
SigningPubKey	Alice's public key	Dave's public key
Tx Signature	*752EF7...3402D1*	*42EF56...34DDFF*

The core activity in the Ripple network centers around multi-wallet operations. As described earlier in this chapter, a wallet u_i can hold two types of funds: XRP coins and IOU credit issued by other wallets in the Ripple network connected to u_i through a direct link. As they are conceptually different, the Ripple network handles them by two types of operations: *direct XRP payments* and *path-based settlement transactions*. Intuitively, a direct payment involves a transfer of XRP between two wallets which may not have a credit path between them. Path-based settlement transactions transfer IOUs defined in any currency (fiat currencies, cryptocurrencies and user-defined currencies) between two wallets having a suitable set of credit paths between them.

We now describe both types of payments by an illustrative example. We start with direct XRP payments. Assume that a wallet u_1 wants to pay β XRP to another wallet u_2 and that u_1 has at least β XRP in its XRP balance. Then, β XRP are removed from u_1's XRP balance and added to u_2's XRP balance. For example, in the transaction showed in Table 2.1, 20 XRP are about to be transferred from Alice to Bob. Given that Alice's XRP balance is high enough, 20 XRP are taken from Alice's XRP

balance and added to Bob's XRP balance. Notice that this type of transaction does not require the existence of any (direct or indirect) credit line between the sender and the receiver. Therefore, the *Path* field of the transaction is not used.

Path-based settlement transactions totally depart from direct payments as they use the credit links available in the Ripple network. Assume that u_1 wants to pay β IOUs to u_n and that u_1 and u_n are connected by a path of the form $\mathsf{u}_1 - \mathsf{u}_2, \ldots, \mathsf{u}_{n-1} - \mathsf{u}_n$. Edges are considered undirected to find a path from the sender u_1 to the receiver u_n through intermediaries $\{\mathsf{u}_i\}_{i\in\{2,\ldots,n-1\}}$. In order to perform the transaction, the weight (i.e., credit value) on every edge in the path from u_1 to u_n is updated depending on the direction of the edge as follows: edges in the direction from u_1 to u_n are increased by β, while reverse edges are decreased by β. For the settlement transaction to be successful, weights must always remain non-negative and must not exceed the pre-defined upper bound of the edge (if the upper bound is other than ∞).

In the settlement transaction shown in Table 2.1, assume that Dave wants to pay 10 EUR to Edison. This transaction can be routed using the path Dave ← Gateway → Edison (see Figure 2.2). Since credit link Dave ← Gateway holds 20 EUR (i.e., > 10 EUR) and credit link Gateway → Edison has no upper bound, the transaction can be performed and credit links are updated as follows: link Dave ← Gateway is decreased to 10 EUR while link Gateway → Edison is increased from 85 to 95 EUR.

It is not necessary to find a single path with available credit along each credit link. Instead, the settlement transaction can be split across multiple paths such that the sum of credit available on all paths is larger than or equal to β. For example, in the network from Figure 2.2, assume now that Dave wants to pay 25 EUR to Edison. The settlement transaction now can be split into two settlement transactions with amounts of 20 EUR and 5 EUR. The 20 EUR settlement transaction can be performed as explained earlier, while the 5 EUR settlement transaction is carried out over the path Dave ← Fanny → Edison. In Ripple, it is possible to include the information about the several paths used in a single settlement transaction: the list of paths are included

in the *Path* field annotated with the amount of credit used per path. The *Amount* field still indicates the total amount of transacted IOUs.

Moreover, in our running example we have assumed that all the links have a common currency. In the Ripple community, *rippling* denotes the redistribution of credit for each intermediate wallet as a consequence of a transaction [34]. Rippling can only occur between two credit links that belong to the same wallet and have credit in the same denomination. Nevertheless, several rippling operations can be concatenated to carry out a transaction with several intermediate wallets, as described above. We expect that rippling is allowed by gateways; however, less active users may opt for disabling this rippling functionality, effectively avoiding undesired balance shifts.

Nevertheless, settlement transactions are not restricted to same-currency transactions. It is possible that the sender uses some of her IOU in any given currency and the receiver receives the corresponding amount of IOU in any other currency, carrying out thereby a *cross-currency* settlement transaction. Such transaction is possible only if at least one of the intermediate wallets acts as *market maker*.

In summary, for completeness in this section we have described both payments and settlement transactions. However, in the rest of this dissertation we focus on settlement transactions as they are the only ones that transfer IOU credit among wallets in the network. Moreover, we observe that the XRP currency might not be required for implementing transaction fees. Instead, fees can be embedded in the IOU settlement of the transaction itself: When a user in the path from the sender to the receiver gets β IOU, she can forward only $\beta - \alpha$ IOU, effectively charging α IOU as fee. This way of handling fees comes with the advantage that every intermediate user can charge a custom fee according to her own criteria (e.g., the transacted amount, transacted currency or transacting users).

2.1.4 Consensus Protocol

Inspired from the success of cryptocurrencies like Bitcoin, all Ripple operations are also logged in a public available blockchain called *Ripple ledger*. The Ripple ledger is thereby an immutable log that keeps track of all wallets, credit links and exchange offers in the Ripple network as well as their evolution over the Ripple timespan. The *Ripple consensus protocol* is carried out by a set of (somewhat fixed) participants called *validators* and it is used to decide the set of operations that are added to the Ripple ledger. In the following, we overview the Ripple consensus protocol.

An operation is authorized by the sending wallet's owner by signing it with the corresponding signing key. Such operation is then forwarded to a validator. Validators are thus in charge of receiving authorized operations from users and validate their correctness according to the current state of the ledger and the consensus rules. Note that since different wallet's owners might forward their operations to different validators, they might end up with different sets of operations. The consensus protocol must then ensure that all validators agree on an unique ledger.

The consensus protocol proceeds in rounds. In the first round, each validator broadcasts its own *candidate set* of operations, that is the set of validated operations that it has received so far and are not added to the ledger yet. Successive rounds are used to vote a *candidate ledger* that contains the subset of operations from all candidate sets that have been voted by "enough" validators. When a candidate ledger is voted by $50\%, \ldots, 80\%$ of the validators (increased by 10% per round), it is considered final and it is added to the Ripple ledger. The operations that do not make it into the ledger are either discarded or added to the next protocol iteration.

Although several permissioned consensus algorithms, such as Byzantine Fault Tolerance (BFT) consensus [35], have been throughly studied in the distributed systems literature over the last 35 years, the Ripple consensus protocol is a novel approach informally presented in a whitepaper [36] along with an open source implementation. Moreover, only some preliminary analysis have been performed so far [37–40]. In

general, the lack of formal definitions and security studies of the Ripple consensus protocol, makes its safety and liveness analysis an interesting open problem.

2.2 Security Study

In this section, we overview the possible vulnerabilities of the Ripple network to attacks that affect the IOU credit of users' wallets and we refer the reader to [41] for a more detailed discussion. In particular, we find that about 13M USD are at risk in the current Ripple network due to inappropriate configuration of the rippling flag on credit links, facilitating undesired redistribution of credit across those links. Although the Ripple network has grown around a few highly connected hub (gateway) wallets that constitute the core of the network and provide high liquidity to users, such a credit link distribution results in a user base of around $112,000$ wallets that can be financially isolated by as few as 10 highly connected gateway wallets. Indeed, today about 4.9M USD cannot be withdrawn by their owners from the Ripple network due to PayRoutes, a gateway tagged as faulty by the Ripple community. Finally, we observe that stale exchange offers pose a real problem, and exchanges (market makers) have not always been vigilant about periodically updating their exchange offers according to current real-world exchange rates. For example, stale offers were used by 84 wallets to gain more than 4.5M USD from mid-July to mid-August 2017. Our findings should prompt the Ripple community to improve the health of the network by educating its users on increasing their connectivity, and by appropriately maintaining the credit limits, rippling flags, and exchange offers on their IOU credit links.

2.2.1 Effect of Unexpected Balance Shifts

Although a settlement transaction maintains the net balance of intermediate wallets, its use is not innocuous for intermediate wallets. The main issue is that the market value and stability of the IOU depends on the issuer of such IOU. In our illustrative example of the Ripple network in Figure 2.2, Edison may trust the credit

from the gateway more than the credit from Fanny. Therefore, a transaction involving rippling among the two corresponding credit links can induce a redistribution of IOU from a more valuable to a less valuable issuer without the specific consent of the involved wallet's owner. We expect gateways to allow rippling; however, less active users may wish to avoid balance shifts not initiated by them.

As a countermeasure, each credit link is associated with a flag no_ripple. When no_ripple is set, the corresponding credit link cannot be part of a rippling operation. This flag was first added in December 2013, and was updated in March 2015 to have a default state of "set" (i.e., no rippling allowed by default), so users could selectively opt-out and allow rippling. Additionally, a wallet has a new flag called defaultRipple that, if set, enables rippling among all the wallet's credit links. Gateway wallets, for instance, follow this pattern [42].

In this state of affairs, we aim to identify wallets other than gateways that allow rippling, and to extract how much credit they put at risk doing so. For that, we proceed as follows. First, the credit links not including no_ripple flag are tagged as no_ripple = false. Second, for each wallet that has the defaultRipple flag set, we set no_ripple = false (i.e., rippling is allowed) on all its credit links. Third, we use the no_ripple flag for the remainder of the links as specified in the Ripple network. Now, we say that a wallet is prone to rippling if it has at least two credit links with no_ripple = false (i.e., they allow rippling) and they hold credit in the same currency.

Following this methodology, we find that more than $11,000$ wallets are prone to rippling and are not associated with well-known gateways. Moreover, more than 13M USD are prone to rippling, counting only the credit links that wallets prone to rippling have directly with gateways, as they are associated with real-world deposits. This gives a lower bound on the amount of credit at risk, and the actual value could be higher, if we count credit at risk with wallets other than the gateways. This result demonstrates that unexpected balance shifts in the Ripple network can still affect a significant number of wallets, and more importantly, their credit.

We also observe that many wallets prone to rippling maintain credit links with a low balance (even zero), but with upper limit set to a value larger than zero. The gap between balance and upper limit on these credit links can be used to shift the balances of wallets, thus increasing the risk.

As a possible countermeasure, the users have the possibility of disabling the rippling functionality on their credit links completely. Therefore, less active users may opt for disabling rippling among their credit links to avoid balance shifts not initiated by them. Moreover, more active users can also opt for dynamically adjust the amount of credit prone to rippling and add a rippling fee to it. Finally, users with credit links holding zero balance should reduce their upper limit to effectively void them.

2.2.2 Effect of Faulty Gateways

The gateway wallets are highly connected wallets included in the core of the Ripple network and significantly contribute to the liquidity of the network. A faulty gateway can disable rippling on most credit links of its wallet, ensuring that transactions routed through it are no longer possible and effectively freezing the balance held at credit links of its wallet [43, 44]. This would not only severely affect the liquidity of the network, but also lead to monetary losses to the neighboring wallets, as they no longer can use the credit issued by the compromised wallet.

Given that, we aim to study the effect of faulty gateway wallets (e.g., as a result of adversarial wallet compromise) and the resilience of the Ripple network to them. Towards this goal, we first select 100 candidate faulty wallets from the Ripple network according to two different criteria: (i) Wallets with highest degree (100-deg) and (ii) Wallets involved in most of the transactions (100-ftx). Although other strategies to select wallets are definitely possible, these strategies lead us to select the key players in the current Ripple network: Gateways and market makers. Second, we assess the most disruptive set of wallets by removing them from the Ripple network and observing how the network connectivity is affected. Figure 2.3 depicts the size of the largest

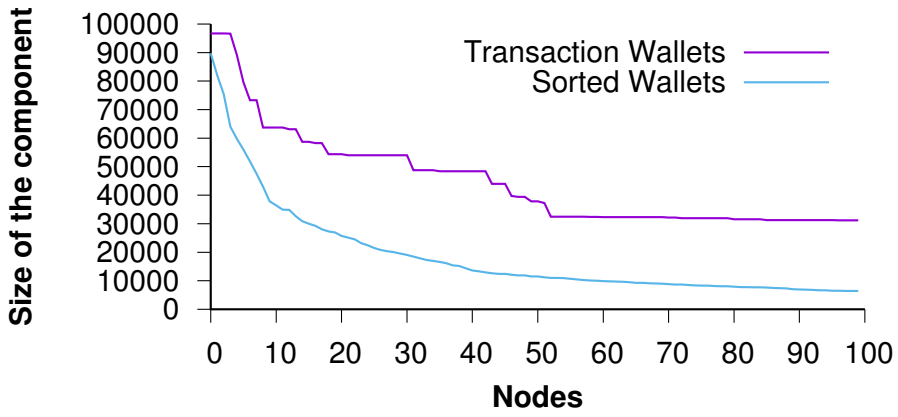


Fig. 2.3. Size of the largest connected component after removing wallets sorted by number of credit links (blue) and number of appearances in transactions (purple).

connected component after removing the wallets in 100-deg and 100-ftx. Intuitively, the smaller the component, the fewer the possible transactions, since only wallets in the same component can transact with each other. From this experiment, we conclude that wallets included in 100-deg have a more profound impact on the connectivity of the Ripple network (and therefore on the transactions) than wallets included in 100-ftx. Therefore, we use 100-deg in the rest of this section.

Afterwards, we define the *resilience factor* (rsl-factor) as the ratio between the component size in the most disruptive splitting of the network after removing a wallet (i.e., splitting the network in two components of equal size) and the size of the actual largest component after removing a wallet. Therefore, the rsl-factor can take values in the range $[0.5, 1]$. Values close to 1 indicate that the network has a low resilience, as the removal of a wallet resulted in a component with (close to) half of the wallets of the network. Conversely, values close to 0.5 indicate that the network has a high resilience, as the largest component after removing a wallet is (close to) the entire graph.

We observe that the rsl-factor of the Ripple network is maintained in the range $(0.5, 0.6)$ after the removal of each wallet in 100-deg, demonstrating that the core of

the Ripple network has high resilience. We conclude that we can divide the Ripple network into: (1) A small network core of around 65,000 wallets (36% of the total) that includes the key wallets with high connectivity. This core is highly resilient to the removal of highly connected wallets, and (2) A large set of around 112,000 wallets that can be easily disconnected from the network after removal of key wallets. Yet, these highly vulnerable wallets have more than 42M USD of credit with the gateways, which is at risk.

This result shows that the Ripple network still has a few wallets that are "too big to fail." The more centralized a credit network is, the higher the impact of a failing well-connected wallet. This could effectively be comparable to a very large bank failing in the traditional banking world (e.g., the case of Lehman Brothers). As a countermeasure, it is necessary for many users to increase their connectivity and split their credit among different credit links to avoid losses due to the failure of a handful of wallets.

2.2.3 Effect of the PayRoutes Gateway

While studying the Ripple network, we observed that the Ripple community had reported the unresponsiveness of the company running the gateway PayRoutes when contacted regarding the credit issued by it [45]. We also emailed them, but got no answer at the time of this writing. In this state of affairs, we study PayRoutes as an example of a faulty gateway.

In a bit more detail, we consider two questions. First, we aim to find the amount of credit in the Ripple network that can only be withdrawn with the cooperation of PayRoutes and, given the unresponsiveness of the gateway, this credit is "stuck" in the Ripple network. Second, we study why wallets with stuck credit obtained it in the first place, even though PayRoutes was already reported as faulty. We describe our methodology and results for each goal separately in the following two sections.

Credit with PayRoutes Here, we are interested in credit links of the form PayRoutes $\rightarrow u_i$ where PayRoutes has disabled rippling. This implies that the credit on these links can only be used in a withdrawal operation jointly with PayRoutes: u_i sets the credit on the link to 0 to obtain the corresponding amount in the real world from PayRoutes. However, as PayRoutes is a faulty gateway, this operation is no longer available and the credit is stuck. Given that, we first address the question: *how much credit is stuck on credit links with PayRoutes?*

In order to answer this question, we first pick the credit links with PayRoutes as counterparty and positive balance, and derive the status of their rippling flag (as described in Section 2.2.1). Then, we classify the neighbor wallets of PayRoutes into two groups as follows. First, we identify those wallets that have a credit link with PayRoutes for which rippling is not allowed, i.e., no_ripple is set to true. We denote this set of wallets by *wallets-no-rippling*. Second, we consider the set of wallets that are not in *wallets-no-rippling* but yet cannot perform a transaction for an amount equal to the balance on their credit link with PayRoutes. We denote this second set as *wallets-rippling-no-tx*. As the wallets in either *wallets-no-rippling* or *wallets-rippling-no-tx* cannot transfer the (entire) credit they have on a credit link with PayRoutes to another wallet in the Ripple network, the only way for them to get their credit back is to contact PayRoutes in the real world and withdraw the corresponding funds. However, as PayRoutes is unresponsive, such credit is "stuck."

As result of this procedure, we observe that, out of the $2,958$ wallets that have at least one credit link with PayRoutes, there exist 621 wallets in either *wallets-no-rippling* or *wallets-rippling-no-tx*, and therefore with stuck credit. We observe that the stuck credit on these credit links is around 4.9M USD.

It is important to note that the PayRoutes case is not typical in the Ripple network. There have been other gateways that have ceased their activities during the Ripple network lifetime, but have not caused such an effect. We consider DividendRippler as an example of such a gateway. The difference from PayRoutes is that before shutting

down, DividendRippler publicly announced it and mandated its clients to proceed to withdraw the credit available in their credit links with DividendRippler.

We conduct the same above experiment for DividendRippler, and observe that, although 665 wallets have credit stuck with DividendRippler, such credit accounts for around 1,000 USD only. This is how much DividendRippler currently owes to the rest of wallets. This demonstrates that wallets followed the announcement of the gateway and successfully managed to withdraw most of their credit before the gateway closed.

Obtaining Credit from PayRoutes In this part we focus on answering the question: *How did wallets with stuck credit obtain such credit in the first place?*

For that, we first investigate how new credit links were created with PayRoutes over the lifetime of the Ripple network. We observe a spike of 2,527 credit links created in October 2016 from a total of 1,805 wallets. Out of these, 186 credit links were created by 133 wallets and have balance stuck in PayRoutes. This implies that 21% of the wallets with stuck balance created credit links with PayRoutes during that month. We denote these by stuck-wallets-Oct-16.

Given this unusual behavior, we study how those 133 wallets obtained credit. We identify two possibilities: (i) A path-based transaction from another wallet in the Ripple network; (ii) A circular transaction (i.e., sender and receiver of the transaction are the same wallet), where a wallet pays a certain amount of XRP (or any currency issued by a gateway other than PayRoutes) in exchange for credit issued by PayRoutes on a credit link with it.

As a result from this study, we observe that wallets in stuck-wallets-Oct-16 do not receive significant credit from other wallets in the Ripple network during October 2016. In particular, we find only three transactions with credit values of 10 USD, 100 ILS and 5 ILS. Instead, wallets in stuck-wallets-Oct-16 get their credit through circular transactions. We find that 51 wallets perform a total of 286 circular transactions, where these wallets received around 12,000 USD in exchange for approximately 300 CNY and 12,000 XRP.

Table 2.2.
Summary of the exchange offers between XRP and USD created in the Ripple network during October 2016. Each row represents the combination of all offers exchanging the corresponding pair of currencies.

Pay Val	Pay Cur	Get Val	Get Curr	Ratio
1062738.51	XRP	17009.50	USD	62.48 to 1
59678.62	USD	33194.62	XRP	1.78 to 1

In essence, wallets in stuck-wallets-Oct-16 invested mostly XRP to obtain USD from PayRoutes. We find that the exchange rate XRP/USD in the Ripple network was considerably "better" than in the real world at that time: In the Ripple network at that time, a wallet could get 0.73 USD for 1 XRP on average, with a minimum of 0.14 and a maximum of 2.87 USD using stale offers available in the network. However, in the real world, one could get less than 0.01 USD for 1 XRP at the average exchange rate at that time and up to 0.28 USD for 1 XRP, even considering the best exchange rate over the entire Ripple network lifetime.

The results presented above describe the origin of a small fraction of the credit stuck on credit links with PayRoutes. We repeated the same experiment over the complete Ripple network lifetime and observed similar patterns. First, the credit links with stuck credit are involved in a total of 278 transactions where other wallets in the Ripple network are sending credit to victim wallets at a favorable rate: The receiver gets more credit than actually sent by the sender. Those transactions account for around $158,000$ USD. Second, the highest amount of credit is received as a result of circular transactions that use advantageous offers. In particular, we find that credit links with stuck credit are involved in a total of $16,469$ transactions where they gained more than 63M USD over the complete Ripple network lifetime.

Although wallets with stuck credit at PayRoutes obtained considerable revenue, a broader perspective reveals that it was a risky operation. For instance, as a possible countermeasure to this issue it is possible to check the exchange rates available in the

Ripple network at October 2016 to determine how likely it is to get the USD credit back. In particular, we observe that although wallets in stuck-wallets-Oct-16 managed to get "cheap" USD, the market values were not favorable to get them back: New exchange offers created in the Ripple network in October 2016 (as shown in Table 2.2) demonstrate this.

2.2.4 Effect of Stale Offers

Exchange offers and rippling are the key operations that enable path-based transactions. The previous two sections investigated the security of rippling, so we now investigate the safety of exchange offers, which are set by the owners of wallets at their own discretion. Naturally, proposed offers should match those of the corresponding currencies in the real world or even be in favor of market makers so that they get credit for their exchange services. Otherwise, cunning users can leverage stale offers to gain credit, while market makers may go bankrupt. This would adversely impact the liquidity and availability of the Ripple network.

In this state of affairs, we aim to determine whether there are stale offers in the Ripple network and, if so, study to what extent devilry users have taken advantage of them. Here, we consider the coin market capitalization (`https://coinmarketcap.com/`) as representative source to know the prices for cryptocurrencies outside the Ripple network. In order to achieve our goal, we first search for sudden changes in the currency's market capitalization. We observed several such changes. We first examine a spike in the price of XRP in late 2013: During a period of ten days (Nov 20th–30th, 2013), the price of 1 XRP with respect to BTC increased by 380%, i.e., 1 XRP was exchanged at 0.00001 BTC at the beginning of the period but within a week, 1 XRP was exchanged at 0.000038 BTC. Given that, we extract all the transactions that occurred during this ten-day period, obtaining a total of $1,932$ transactions. We prune this dataset by considering only cross-currency transactions that exchange XRP for BTC or vice versa. We obtain a total of 112 transactions.

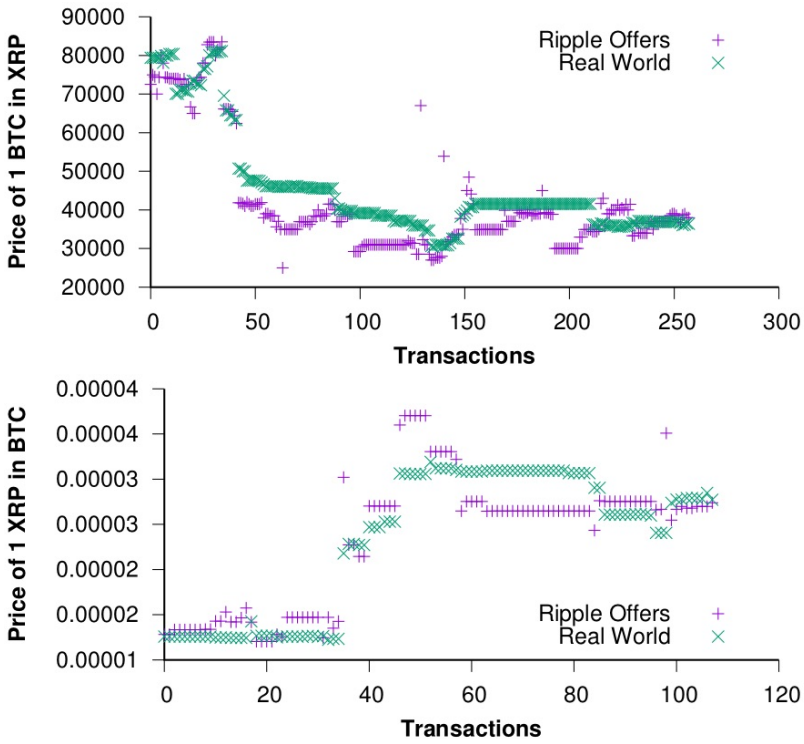


Fig. 2.4. Illustrative example of exchange offers in the Ripple network. Market maker accepts XRP and pays BTC (top); market maker accepts BTC and pays XRP (bottom). If the purple point (offer in Ripple) is below the green point (offer in real world), the transacting user gained credit. Otherwise, the market maker gained credit. These transactions took place between November 20th and 30th, 2013.

We compare the exchange rate between XRP and BTC used in each transaction to the exchange rate in the real world at the same time, as shown in Figure 2.4. In both figures, a purple point represents the exchange rate in a Ripple transaction while the corresponding green point denotes the exchange rate in the real world at the same time. For both graphs, if the purple point is higher than the green point (Ripple's offer is more expensive than the real world offer), the market maker made money. In contrast, if the purple point is below the green point, the user who conducted the transaction gained credit.

We analyzed the transactions in which a sender gained credit by exploiting stale offers. We make two observations. First, users could have gained up to around 250,000 USD by fully exploiting XRP/BTC stale offers during the specified period. In other words, market makers put at risk around 250,000 USD due to stale offers. Second, 24 different wallets made a monetary benefit of at least 7,500 USD by exploiting XRP/BTC stale offers (and other offers available in the network at that time). Here, we calculate the USD value by converting the BTC and XRP to their real world exchange rates at the corresponding times. In summary, even in the nascent stages of the Ripple network, when the transaction volume was considerably low, market makers risked significant loss of credit by letting exchange offers become stale.

To confirm these results, we explored another, more recent, substantial change in a currency exchange rate. We found a sudden increase in the price of BTC compared to XRP in 2017, concretely during the period July 16th – August 16th: The value of 1 BTC went from 11,713 XRP to 25,735 XRP, that is, an increase of 120%. As before, we extracted the transactions during that period of time and compared the exchange rates of XRP from/to BTC in the Ripple network and in the real world. We observe that market makers put at risk around 500,000 USD due to stale offers exchanging XRP to BTC and vice versa. Moreover, we observe that 84 wallets exploited these stale offers (and possibly other offers) to gain more than 4.5M USD. These results confirm that stale offers continue to be a risk for market makers. In fact, the effect of stale offers is now amplified given the growth of the Ripple network and transactions.

As a possible countermeasure to this problem, a market maker can update a previously offered exchange rate at any time. Therefore, a market maker should continuously monitor the price for the currencies involved in its offers and promptly update its exchange offers in the Ripple network when a sudden change occurs in the real world. The gaps between exchange rates in the Ripple network and real world are thereby reduced, and with them, the windows for cunning users to gain credit.

2.3 Privacy Study

The Ripple ledger serves as a unique and append-only log that keeps track of all wallets in the Ripple network, credit links between them and all valid operations that ever happened in the Ripple network. Remember that a wallet in Ripple is represented by the hash of the corresponding public key that effectively serves as a pseudonym for the wallet. Therefore, although pseudonyms are not directly tied to real world identities, it is possible to reconstruct the complete financial activity performed by a single pseudonym. Although several research works have shown privacy breaches in Bitcoin due to the use of pseudonyms in the blockchain [46–52], the important issue of privacy in credit networks has not been thoroughly studied yet. This state of affairs leaves open important questions such as *Is privacy a real problem in Ripple? Can we measure it?*

In this section, we overview our study of the Ripple network that sheds light to these questions [19]. In particular, we propose two heuristics based on observations of the interactions between Ripple wallets themselves and interactions of these wallets with online exchange services to deposit and withdraw cryptocurrencies in and from the Ripple network. By doing so, we show that it is possible to cluster wallets that belong to the same user across different systems (not only Ripple but also cryptocurrencies). Additionally, we propose deanonymization mechanisms to reveal the identity behind the clustered wallets. These results show the privacy breaches inherent to a publicly available ledger, a practice followed by other credit networks as well such as Stellar.

2.3.1 Heuristic 1: Interledger Linkability

Our first heuristic can be illustrated with *the tale of two logs*, as shown in Figure 2.5. Assume that Alice has certain bitcoins in her Bitcoin wallet. Using the Bitcoin system, she can only pay for services that accept payments in BTC. However, she cannot pay to Bob as he only accepts payments in USD. In this state of affairs, Alice can instead transfer these bitcoins into the Ripple network, getting thereby the corresponding

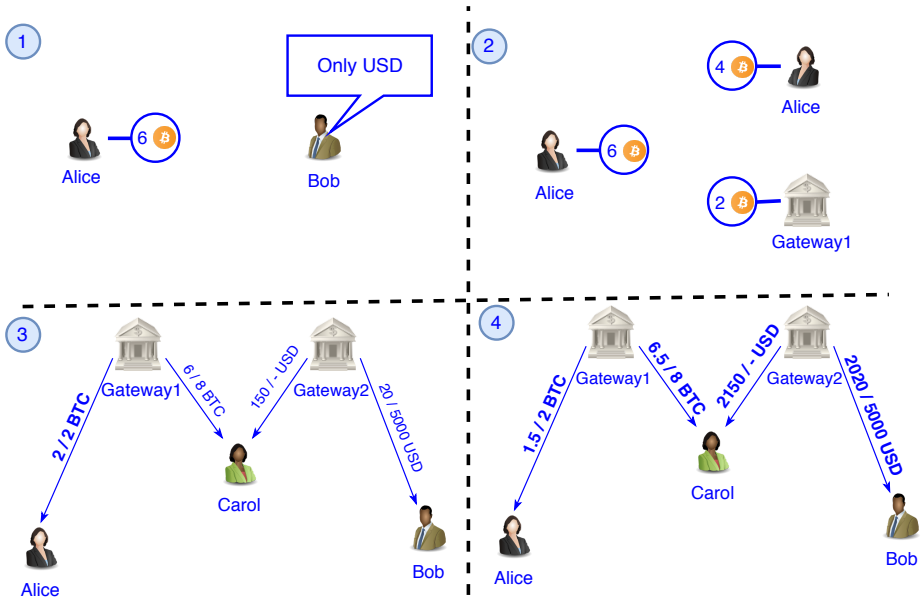


Fig. 2.5. The tale of two logs. Alice cannot pay bitcoins to Bob as he only accepts USD (top-left). Out of her 6 bitcoins, Alice pays 2 bitcoins to a gateway in the Bitcoin network and keeps the other 4 bitcoins for her (top-right). Alice gets the corresponding 2 BTC from the gateway in the Ripple network (bottom-left). Finally, Alice performs a cross-currency transaction to settle her debt with Bob (bottom-right).

amount of BTC IOU. For that, she needs to interact with a gateway that provides the service of exchange funds among the two networks. In particular, Alice can pay to the gateway in bitcoins within the Bitcoin network. The gateway in turn pays back to Alice the corresponding BTC IOU in the Ripple network. Now, she is able to pay to Bob as the Ripple network allows the exchange from BTC IOU into USD using the currency exchanges offered by market makers (Carol in this example).

There are several gateways (e.g., Bitstamp and GateHub) which offers users the possibility to transfer bitcoins (or any of the altcoins) into the Ripple network and vice versa. As mentioned before, Alice can pay the gateway a certain amount of bitcoins. The gateway, upon reception of the bitcoins, issues the corresponding BTC IOU to the credit link that Alice has previously formed with the gateway. We call this transaction

deposit transaction. On the other hand, Alice could send (part of) her BTC IOU to the gateway which in turn, transfers back the corresponding amount of bitcoins to the Alice's Bitcoin wallet. We call this transaction *withdrawal transaction*.

The key observation here with respect to privacy is that the a priory independent payments carried out during deposit and withdrawal transactions are logged in the corresponding ledgers and can be linked together following the aforementioned mechanics. For instance, in the example depicted in Figure 2.5, the payment from Alice to the gateway (top-right) is logged in the Bitcoin blockchain while the issue of credit from the gateway to Alice (bottom-left) is logged in the Ripple ledger. Although Alice must use different wallets in Bitcoin and Ripple, the fact that both logs are publicly available allows an observer to link both operations together and in turn determine Alice's wallets in both systems. As the reader can imagine, this process can be extended to link wallets that belong to the gateway as well as to extract other blockchain-based cryptocurrencies (i.e., altcoins) wallets that can further be linked.

Heuristic 1 [Deposit and withdrawal at the gateway] *The heuristic for deposit operations to link Bitcoin and Ripple wallets belonging to the same user involves the following steps:*

1. *Assume w_g is a Ripple wallet owned by the gateway. Extract the set of all transactions in the Ripple network where w_g is the sender. We denote this set by $T_s(w_g)$. Moreover, for every transaction $t \in T_s(w_g)$, obtain the corresponding Bitcoin transaction. We denote the corresponding Bitcoin transaction by t_b.*

2. *For every transaction $t \in T_s(w_g)$ create a pair $(w_g, rcv(t_b))$, where $rcv(t_b)$ is the receiver of the Bitcoin transaction t_b corresponding to t. All these pairs thereby created correspond to Ripple, Bitcoin wallets belonging to the gateway. On the other hand, for every transaction $t \in T_s(w_g)$, create a pair $(rcv(t), sdr(t_b))$, where $rcv(t)$ denotes the receiver wallet of the Ripple transaction t and $sdr(t_b)$ denotes the sender wallet of the corresponding Bitcoin transaction. The two wallets of such a pair are owned by the same user.*

The heuristic for withdrawals to link together Bitcoin and Ripple wallets belonging to the same user involves the following steps:

1. *Assume that w'_g is a Ripple wallet owned by the gateway. Extract the set of all transactions in the Ripple network where w'_g is the receiver. We denote this set by $T_r(w'_g)$. Moreover, for every transaction $t' \in T_r(w'_g)$, obtain the corresponding Bitcoin transaction, which we denote by t'_b.*

2. *For every transaction $t' \in T_r(w'_g)$ create a pair $(w'_g, sdr(t'_b))$, where $sdr(t'_b)$ is the sender of the Bitcoin transaction t'_b corresponding to t'. All these pairs thereby created correspond to Ripple, Bitcoin wallets belonging to the gateway. On the other hand, for every transaction $t' \in T_r(w'_g)$, create a pair $(sdr(t'), rcv(t'_b))$, where $sdr(t')$ denotes the sender wallet of the Ripple transaction t' and $rcv(t'_b)$ denotes the receiver wallet of the corresponding Bitcoin transaction. The two wallets contained in such a pair are owned by the same user.*

Figure 2.6 (top) shows an illustrative example for a deposit transaction. Assume Alice wants to get 2 BTC IOU into her Ripple wallet $\mathsf{Alice}_1^{\divideontimes}$. To achieve that, she first creates a Bitcoin transaction where she transfers 2 BTC from her Bitcoin wallet $\mathsf{Alice}_1^{\text{฿}}$ to the gateway's Bitcoin wallet $\mathsf{Gw}_1^{\text{฿}}$. Once the gateway has checked the validity of the Bitcoin transaction, it creates a Ripple settlement transaction where it issues 2 BTC IOU from its Ripple wallet $\mathsf{Gw}_1^{\divideontimes}$ to Alice's Ripple wallet $\mathsf{Alice}_1^{\divideontimes}$. This implies that $\mathsf{Alice}_1^{\text{฿}}$ and $\mathsf{Alice}_1^{\divideontimes}$ are owned by Alice while $\mathsf{Gw}_1^{\text{฿}}$ and $\mathsf{Gw}_1^{\divideontimes}$ are owned by the gateway. Moreover, following the heuristics regarding Bitcoin change addresses proposed by Meiklejohn et al [47], we can infer that $\mathsf{Alice}_2^{\text{฿}}$ also belongs to Alice.

Figure 2.6 (bottom) shows a withdrawal transaction. Assume Alice wants to withdraw 1 BTC IOU from the Ripple network into her Bitcoin wallet. For that, she first sends 1 BTC IOU from her Ripple wallet $\mathsf{Alice}_2^{\divideontimes}$ to the gateway's Ripple wallet $\mathsf{Gw}_2^{\divideontimes}$. Once the gateway has received the BTC IOU, it transfers 1 BTC from its Bitcoin wallet $\mathsf{Gw}_2^{\text{฿}}$ to Alice's Bitcoin wallet $\mathsf{Alice}_3^{\text{฿}}$. The withdrawal implies that

DEPOSIT

Bitcoin Transaction	
Input	**Output**
$\mathsf{Alice}_1^{฿}$: 6	$\mathsf{Gw}_1^{฿}$: 2
	$\mathsf{Alice}_2^{฿}$: 4

Ripple Transaction	
Field	**Value**
Sender	Gw_1^{*}
Receiver	Alice_1^{*}
Path	$\mathsf{Gw}_1^{*} \rightarrow \mathsf{Alice}_1^{*}$
Amount	2 BTC IOU

WITHDRAWAL

Ripple Transaction	
Field	**Value**
Sender	Alice_2^{*}
Receiver	Gw_2^{*}
Path	$\mathsf{Alice}_2^{*} \leftarrow \mathsf{Gw}_2^{*}$
Amount	1 BTC IOU

Bitcoin Transaction	
Input	**Output**
$\mathsf{Gw}_2^{฿}$: 15	$\mathsf{Alice}_3^{฿}$: 1
	$\mathsf{Gw}_3^{฿}$: 14

Fig. 2.6. An illustrative example of deposit and withdrawal processes in a gateway. For a deposit, first Alice sends 2 BTC to the gateway and then, the gateway sends 2 BTC IOU in the Ripple network to Alice. For a withdrawal, first Alice sends 1 BTC IOU to the gateway within the Ripple network and then the gateway sends 1 BTC back to Alice in Bitcoin.

$\mathsf{Alice}_3^{฿}$ and Alice_2^{*} are owned by Alice while $\mathsf{Gw}_2^{฿}$ and Gw_2^{*} are owned by the gateway. Moreover, as mentioned before, we can infer that $\mathsf{Gw}_3^{฿}$ belongs to the gateway.

We tested the Heuristic 1 in the gateway DividendRippler. Although this gateway is not currently active, at the time of our experiment it was one of the most active gateways in terms of deposit and withdrawals of cryptocurrencies from and to the Ripple network. We limit our description to how we have extracted the necessary information for the steps defined in Heuristic 1 for the deposit process (i.e., steps 1-2). The heuristic for the withdrawal process has been implemented in a similar manner.

1. The DividendRippler wallet (i.e., w_g) is publicly available at its website. The set $T_s(w_g)$ has been obtained from our crawled Ripple transactions.

2. Every deposit has its own page in the DividendRippler's website. This page details both the Bitcoin (correspondingly the Altcoin) and the Ripple transaction involved. Therefore, the t_b corresponding to every transaction $t \in T_s(w_g)$ can be obtained from it. Later in this section, we discuss how to generalize this step to get the Bitcoin transaction corresponding to a Ripple settlement transaction even if the gateway does not publicly show it.
3. For every transaction $t \in T_s(w_g)$, $sdr(t)$ and $rcv(t)$ have been obtained from our Ripple database. The transaction t's webpage also contains a link to the Bitcoin (correspondingly the altcoin) block where the corresponding t_b is stored. From this block, we have obtained the fields $sdr(t_b)$ and $rcv(t_b)$.

Our heuristic finds out a total of 435 Ripple wallets involved in trading with the gateway DividendRippler. Moreover, we have been able to extract 3,145 Bitcoin wallets and 1,173 altcoin wallets divided into 841 Litecoin wallets, 178 Terracoin wallets and 154 Namecoin wallets.

This heuristic impacts the privacy provided by Ripple. In particular, this heuristic enlarges the set of wallets among different cryptocurrencies that can be linked to a given user. This fact has several privacy implications. First, it paves the way to reconstruct the business of a company in a more accurate manner. It is interesting to note that since a business must publicly announce at least one wallet to its customers, the complete (and possible large) set of wallets linked to it are deanonymized. Second, larger sets of linked wallets among different systems affect also the privacy of users. For instance, even if a given user has private wallets in Bitcoin (e.g., she always uses mixing techniques for her transactions), deanonymizing one of her Ripple wallets directly deanonymizes her Bitcoin wallets as well.

Although in this experiment we use a gateway that publishes the Ripple and Bitcoin transactions involved in deposits and withdrawals, our heuristics are also applicable to gateways not publishing this information. In such case, it is possible to collect the Ripple transactions performed by the gateway and link them with high probability to Bitcoin transactions issued in a similar time and transacting the

corresponding amount of bitcoins. This approach leads, however, to a probabilistic guarantee on accuracy and might include false positives in the results. Moreover, as mentioned earlier, our heuristic enables to link not only Ripple and Bitcoin wallets, but also wallets corresponding to other transaction networks (e.g., Stellar) and other cryptocurrencies (e.g., Litecoin, Namecoin or Terracoin).

2.3.2 Heuristic 2: Hot-cold Wallets Linkability

The concept of hot-cold wallet is associated to a behavior that many of us have in our daily life. Instead of carrying all of her funds in her pocket wallet, Alice carries only part of it and spends it on her daily purchases. If she runs out of cash in her pocket wallet, she goes to her bank and withdraws more funds to top-off her wallet. This behavior is also present in the interactions between wallets in the Ripple network, a fact that we leverage in our novel heuristic to link Ripple wallets controlled by the same user.

In a nutshell, users willing to use the Ripple network to attract new business must publicly announce (at least) one of their wallets (i.e., issuing wallet) so that future clients can create credit link with those. For example, gateways publicly advertise their issuing Ripple wallet in their websites. Then, the issuing wallet's owner can issue credit to the clients through the newly created links. However, this practice has two main drawbacks.

First, the issuing wallet becomes an attractive target for an attacker: if the secret key of such wallet gets compromised, the attacker can freely issue an amount of unauthorized IOUs bounded only by the upper bound on these wallet's links. This problem is even more prominent given that upper bounds in the links are set to ∞ by default unless the user changes them. Such an attack has already been observed in the Ripple network and the stolen wallet's owner has gone bankrupt. Second, as the Ripple ledger is publicly available, announcing ownership of a wallet and using it to carry out all the settlement transactions clearly leads to privacy leaks: everybody can track

the settlement transactions of the issuing wallet and reconstruct the complete activity of the given user. Nevertheless, current businesses (such as banks and gateways) seek to maintain privacy of their activities while using the Ripple network.

In order to overcome these issues, Ripple defines the hot-cold wallet security mechanism to issue IOUs of any currency [53, 54]. The cold wallet is publicly linked to a certain user. However, actual issuing of the IOUs in a credit link extended to the cold wallet is performed by the hot wallet as follows. First, the hot wallet creates a credit link with the cold wallet. Then, when the owner of the cold wallet must extend credit to a user, she uses the hot wallet to extend that credit, using for this settlement transaction the existing path (hot wallet) $\leftarrow$ (cold wallet) $\rightarrow$ (user wallet).

The hot wallet is therefore considered to be online as it is used for daily settlement transactions. For example, the secret key of the hot wallet might be used by a web application to automatically perform settlement transactions to other users when requested. When the credit link between the hot and cold wallet runs out of IOUs, the cold wallet extends extra IOUs. This operation happens, however, less often and can be performed offline (e.g., signing locally the necessary transaction). Thus, the cold wallet is considered offline.

Following this mechanism, if the thief steals the private key of the hot wallet, he can issue a number of unauthorized IOUs bounded by the IOUs extended from the cold wallet to the hot wallet. Two observations are important here. First, this bound is normally notably smaller than the bound on the number of IOUs a cold wallet can issue. Second, the maximum number of IOUs in the link between hot and cold wallet is totally controlled by the owner of the cold wallet. She, however, does not have any control over the upper bound with the credit links created with the rest of the users.

With respect to privacy, we note that a settlement transaction from the hot wallet to any other user's wallet has the same path structure as a settlement transaction between any two users (i.e., (sender wallet) $\leftarrow$ (cold wallet) $\rightarrow \ldots \rightarrow$ (receiver wallet)). Thus, in principle, settlement transactions from the hot wallet to any user cannot be directly linked to cold wallet's owner. However, we observe that implementing

the hot-cold wallet mechanism forces the user to use her Ripple wallets following a pattern that makes it possible to link her wallets together. Intuitively, first our heuristic detects the possible cold wallets. Then, it checks settlement transactions where the cold wallet is the sender. The receivers of these transactions are the possible hot wallets. Finally, our heuristic links together hot and cold wallets that belong to the same user.

Heuristic 2 [Hot and cold wallets] *The heuristic to link hot and cold wallets belonging to the same user involves the following steps:*

1. *Extract the wallets that only have outgoing credit links in the Ripple network. They form the initial set of potential cold wallets and we denote it by CW. Among the wallets connected to a cold wallet in CW, those that have being paid at least once by such cold wallet are potential hot wallets, which we denote by HW. The rest of the connected wallets (say, a set $\overline{HW}$) are discarded as they are wallets from users other than cold wallet's owner.*

2. *Reduce the set of potential hot wallets HW to those that are paying to other wallets connected to the cold wallet (i.e., the set $HW \cup \overline{HW}$). Let HW' be the thereby reduced set of potential hot wallets. Discarded wallets in this step (i.e., HW- HW') are added to $\overline{HW}$, obtaining the set $\overline{HW'}$.*

   This step intuitively ensures that potential hot wallets are being used to issue IOU to other wallets.

3. *Reduce the set of potential cold wallets CW to those that have less potential hot wallets than discarded hot wallets. In other words, for each cold wallet $cw_i \in CW$, accept cw_i only if $|HW'(cw_i)| < |\overline{HW'}(cw_i)|$. Let CW' be the thereby reduced set of cold wallets. This step ensures there are indeed many wallets demanding IOUs, which are then supplied using a few hot wallets.*

4. *For each cold wallet $cw_i \in CW'$, create pairs (cw_i, hw_j) for each hot wallet $hw_j \in HW'(cw_i)$. Here, each pair of wallets thereby created belongs to the same user.*

Figure 2.7 depicts an illustrative example of Heuristic 2. The wallet $\mathsf{Carol}_1^{\divideontimes}$ is the cold wallet of Carol as it does not have any incoming link in the Ripple network. In other words, the cold wallet can issue IOUs to other wallets in the network, but no other wallet can issue IOUs to it.

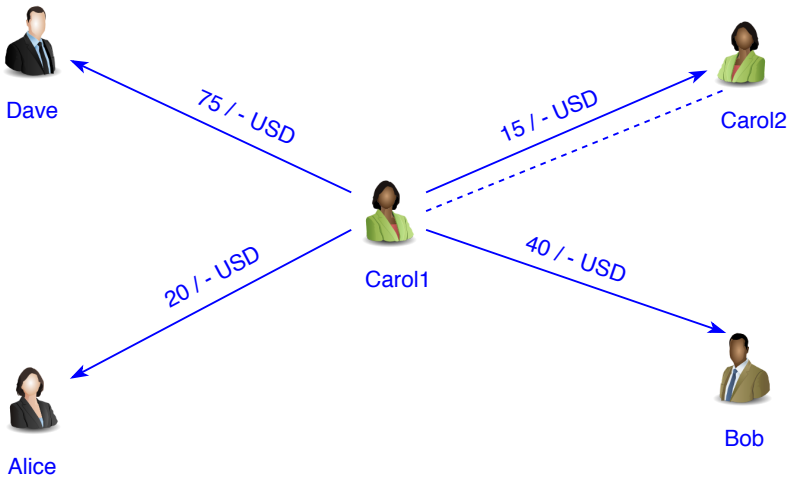
Carol uses her cold wallet ($\mathsf{Carol}_1^{\divideontimes}$) to fund her hot wallet ($\mathsf{Carol}_2^{\divideontimes}$) with 80 and 70 credits in two settlement transactions, while no other wallet is paid by the cold wallet. Then, $\mathsf{Carol}_2^{\divideontimes}$ is used to issue credit to wallets that have extended a credit line with the cold wallet $\mathsf{Carol}_1^{\divideontimes}$, in this example Alice, Bob and Dave. Interestingly, although Bob transfers credit to Alice, it is not linked to Carol given that Bob does not receive any settlement transaction from Carol's cold wallet $\mathsf{Carol}_1^{\divideontimes}$.

Our heuristic can thereby derive the fact that $\mathsf{Carol}_1^{\divideontimes}$ and $\mathsf{Carol}_2^{\divideontimes}$ belong to the same user (i.e., Carol), even though settlement transactions from $\mathsf{Carol}_2^{\divideontimes}$ to other users follow the same path structure as transactions among other users (e.g., settlement transaction from Bob to Alice).

We aim to devise the effect of our heuristic in terms of privacy breaches when applied to the Ripple network. For that, we run the Heuristic 2 to link wallets that belong to the same user. Our algorithm results in 261 cold wallets, 268 hot wallets, having a total of 529 Ripple wallets that have been clustered. Although the results of this heuristic in practice has resulted in a low percentage of clustered wallets, they cover a large number of settlement transactions as we show in Section 2.3.3.

The hot-cold wallet mechanism is a rather recent addition to the Ripple network, and it is not yet extensively applied by the Ripple users. Therefore, it is important to avoid false positives while applying this heuristic. In the following, we describe our mechanism to handle false positives.

During our process to handle false positives we apply the principle of being as strict as possible in order to reduce the number of them. Moreover, from our results



Ripple Ledger					
Sender	**Receiver**	**Amount**	**Sender**	**Receiver**	**Amount**
Carol_1^*	Carol_2^*	\$80	Carol_2^*	Bob	\$50
Carol_2^*	Alice	\$10	Carol_2^*	Dave	\$75
Carol_1^*	Carol_2^*	\$70	Bob	Alice	\$10

Fig. 2.7. An illustrative example of Heuristic 2. The arrows show the credit links. The dashed line represents the wallets linked by the heuristic. Cold wallets (Carol_1^*) do not have incoming credit links. Hot wallets (Carol_2^*) receive credit from the cold wallets. XRP balances are omitted as they are not used in this heuristic.

we observe that false positives fall into two categories: wallets that do not follow the hot-cold wallet mechanism yet and wallets that follow such mechanism but have used the cold wallet to make sporadic payments to wallets other than the hot wallets. We perform the following steps to detect false positives.

First, we calculate the distribution of settlement transactions from cold wallets to potential hot wallets. In the absence of significant ground truth data, we use three gateways (Bitstamp, RippleFox and SnapSwap) well known in the Ripple community for using the hot-cold wallet mechanism, to bootstrap a minimal ground truth for the settlement transaction distributions. Their settlement transaction distributions resemble the Poisson distribution with parameter $\lambda = 1$. We then compute the

divergence of each distribution and the Poisson distribution to detect falsely tagged cold wallets.

In detail, we calculate the statistical distance between two distributions using the Kullback–Leibler (KL) divergence [55] as a measure. Then, we flag a cold wallet as false positive if its settlement transaction distribution diverges from Poisson more than a threshold T. We set up T as the maximum divergence value between our ground truth distributions and Poisson with $\lambda = 1$.

This mechanism has flagged as false positives those cold wallets that do not follow the hot-cold wallet mechanism. In such case, the cold wallet is used to transfer IOUs to many other wallets with a somewhat equal probability, thus having a diversion from Poisson greater than T. We believe that these gateways' behavior is transient and that eventually they will correctly apply the hot-cold security mechanism. Otherwise, as it has happened already, they risk huge losses and the possibility of even going out of business in case their wallet's key is stolen.

In addition, we observe some wallets following the hot-cold mechanism sporadically paying other wallets other than the hot wallets. We conjecture that these cases represent anomalous settlement transactions. A reason for having anomalous transactions is that, in early stages, users employ the hot-cold wallet mechanism in a non-consistent manner. However, we expect that over the period they will start using this hot-cold wallet mechanism correctly and in a consistent manner; otherwise, they may risk huge credit losses and even bankruptcy as it has been already observed in the Ripple network. Moreover, for known gateways using the hot-cold wallet mechanism, we have observed that percentage of anomalous transactions is fairly small. In order to flag these anomalous cases as false positives, we rely on the fact that cold wallet must refund the hot wallet repeatedly over time.

In detail, we consider 3 months (i.e., an economic quarter) as a time frame. Then, only potential hot wallets that are refunded by the cold wallet at least once per quarter for a period of at least two quarters are flagged as real hot wallets. The rest are flagged as false positives. There is a tradeoff choosing these thresholds. First, enforcing a

less frequent refund or a shorter time frame would tag less wallets as false positives, decreasing thus the accuracy of the approach. Enforcing that hot wallets are refunded periodically from when they are created until today would tag real hot wallets as false positives, reducing also the accuracy: Ripple developers suggest to have several hot wallets [53], so that some cold wallets use one hot wallet for a period of time and then change to another hot wallet. Moreover, thresholds for this criteria have been selected following our design principle of being as strict as possible considering the fact that there are path-based settlement transactions in Ripple only for less than 2 years.

Finally, we study the impact of this heuristic in the privacy guarantees of the Ripple network. The hot-cold wallet mechanism has been proposed by Ripple aiming at disassociating settlement transactions from hot wallet and cold wallet so that privacy for cold wallet's owner is increased. However, our heuristic shows a novel technique to link back hot and cold wallets belonging to the same user, thus allowing to reconstruct the complete business (see Section 2.3.3). Thus, our heuristic shows that hot-cold wallet mechanism does not increase privacy in practice.

Moreover, linking hot and cold wallets using our heuristic leads to hinder the security supposedly provided by the hot-cold wallet mechanism. Using our heuristic, an attacker can lucratively target the hot wallets belonging to the target business in order to compromise their private keys and use them to issue unauthorized IOUs. This forces the attacked business to create new hot wallets. This simple countermeasure however does not help as the attacker can repeat the linkability process described in this section and focus his efforts to target the newly created wallet belonging to the target business.

Additionally, our heuristic works for any IOU network following the hot-cold wallet mechanism as described earlier. We focus on the Ripple network as it is currently the most widely deployed credit network in practice. However, we observe that the hot-cold wallet mechanism is also present in the Stellar network so that our heuristic will directly apply to it when they grow to the level of Ripple network.

2.3.3 Deanonymization of Ripple Businesses

We have presented two heuristics that enable the finding of a set of Ripple wallets as well as cryptocurrencies wallets which are owned by a certain user. Table 2.3 shows a summary of our findings as of December 2015, when these experiments where carried out. This process has allowed us to cluster a total of 959 Ripple wallets, 3,113 Bitcoin wallets and 1,130 altcoin wallets. Moreover, Ripple wallets clustered by our heuristics are involved in 161,624 XRP payments and 772,860 settlement transactions. Our clustered wallets were jointly involved in a bit more than 7% of the transactions in the Ripple network.

In the rest of this section, we describe how we leveraged this clustering to deanonymize the business of most of the main gateways. This implies that anybody accessing the publicly available Ripple data can reconstruct the total number of transactions carried out by a gateway, and not only transactions associated to the gateway's public wallets, thereby having a significant privacy breach. Remember that the gateways and their associated transactions represent the main activity for the core of the current Ripple network. They are used to transfer value from the real world into Ripple and vice versa, a crucial task to create liquidity in any starting transaction network such as Ripple or Stellar.

Table 2.3.
Number of wallets clustered in the different heuristics. In Altcoins we consider Litecoin, Namecoin and Terracoin. Finally, for each heuristic and for their grouping, we show the number of Ripple transactions where either the sender or the receiver is a clustered wallet.

Heuristic	**Ripple**		**Bitcoin**	**Altcoins**
	Wallets	*Transactions*	*Wallets*	*Wallets*
1	435	96,009	3,145	1,173
2	529	863,614	–	–
Grouped	959	934,484	3,113	1,130

Single Gateway Business We first consider the deanonymization of business of a single gateway at a time for both DividendRippler and Bitstamp. Although DividendRipple is currently out of business, it was one of the most active gateways for deposits and withdrawals of cryptocurrencies at the time of the experiment. Bitstamp continues being today one of the most active gateways over the complete Ripple network.

In the deanonymization process of the businesses associated to DividendRippler, we observed that DividendRippler publicly announced only one Bitcoin wallet. Extracting the transaction history of such wallet from the Bitcoin blockchain, we observe that more than $1,000$ bitcoins have been transacted. However, this is only a partial view of the gateway's business. As shown in Table 2.4, the transaction history of Bitcoin wallets linked to the gateway by our heuristics shows that more than $5,000$ bitcoins have been ever transacted at this gateway. These results have been possible given the wallets linked by Heuristic 1.

At the time of these experiments, Bitstamp had only published its cold wallet and one of its hot wallets, for which we observed that there have been $72,042$ transactions. However, our Heuristic 2 flagged another Ripple wallet as belonging to Bitstamp. Using this extra information, it is possible to derive that Bitstamp has instead been involved in $132,543$ transactions. Therefore, our heuristics enable the finding of $60,501$ extra transactions involving Bitstamp. During our deanonymization process, we considered transactions where either the sender or the receiver is the linked wallet by our heuristic.

Table 2.4.
Deanonymization of Dividendrippler Bitcoin business.

	Total Sent	**Total received**	**Total Balance**
Public wallets	1062.29	1064.08	1.79
Clustered wallets	5724.38	5724.41	0.03

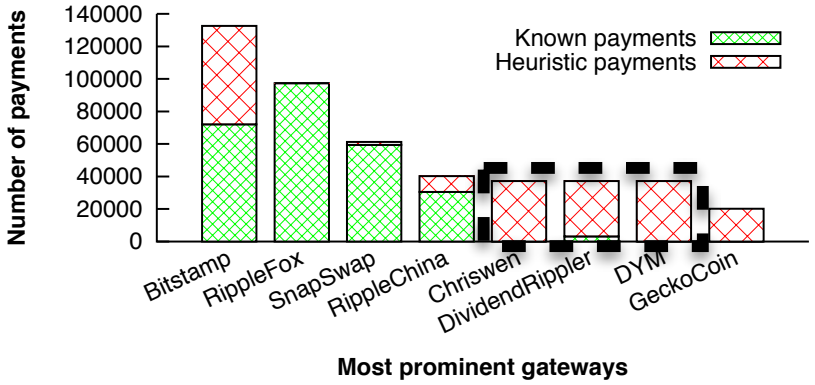


Fig. 2.8. Comparison of the number of transactions associated to publicly known gateways' wallets (i.e., Known payments) and transactions performed with wallets clustered by our heuristics to those gateways (i.e., Heuristic payments). Dashed line groups gateways sharing an owner.

We observed that it is possible to monitor the gateway's business even further. Once the clustering is performed, it is possible to monitor the network to notice every time a transaction is received by a given Ripple wallet. Using this approach it is possible to monitor the complete set of wallets in the cluster of a given gateway, and thereby its full activity in real time.

Several Gateways Business We have carried out the reconstruction of the business associated to the most widely deployed gateways in the same manner we did with Bitstamp's business. We show the most interesting results in Figure 2.8.

We make the following observations. First, there are gateways for which the numbers of publicly available transactions are different. However, adding up the transactions performed with the wallets resulting from our heuristics (Figure 2.8, red bar), they have performed the same total amount of transactions. DividendRippler, DYM and Chriswen constitute an example of this observation. We have verified that indeed DividendRippler and DYM are operated by the same owner. Chriswen has been linked due to the combined results of both heuristics presented in this work: the

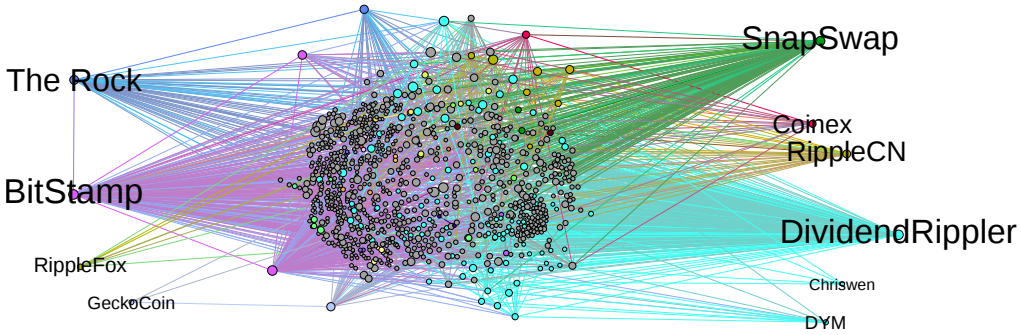


Fig. 2.9. A visualization of the deanonymization process over *our* clustered graph. The sizes of the nodes correspond with the number of transactions involving the nodes. Nodes with the same color belong to the same cluster. Gray nodes depict wallets not deanonymized by our heuristics. Links are colored with the color of the sending wallet.

hot wallet for Chriswen extracted from Heuristic 2 has been used in DividendRippler and it appears in the cluster for DividendRippler and DYM resulting from Heuristic 1.

Second, there are gateways with a few transactions made by their public wallets. However, when adding the payments associated to wallets clustered to them by our heuristics, the number of transactions increases. This is the case, for example, for RippleChina. Finally, we observe that no gateway (except for DividendRippler) publishes its Bitcoin wallets. As our heuristics link Bitcoin and other cryptocurrencies wallets to them, we can further deanonymize their financial activities.

In summary, we have deanonymized 85, 962 XRP payments and 649, 640 settlement transactions, which jointly represent the 78.7% of the total transactions we have considered in our de-anonymization process (see Table 2.3). We have also studied the interactions between the clusters we have obtained from our deanonymization process, as shown in Figure 2.9. We observe that Bitstamp is the gateway with the largest amount of transactions within our cluster. Moreover, we have deanonymized 98 Ripple wallets belonging to the gateway DividendRippler (Figure 2.9, blue nodes). We have observed that most of these wallets were clustered to DividendRippler by the Heuristic 1. In general, these results follow the fact that the probability that

a Ripple wallet gets deanonymized is bigger when the wallet is clustered with our heuristics. This is an important privacy breach: we have shown how to use it to reproduce the business of gateways. Moreover, we contacted several gateways with the list of Ripple wallets linked to them by our heuristic aiming at validating our deanonymization results. We have received responses from two of them (i.e., Bitstamp and RippleFox) and both have confirmed the ownership of such wallets. Moreover, these response do not include any wallet missed by our heuristics.

2.3.4 Deanonymization Using a Ripple Server

In the literature, there are several attacks based on maliciously including certain nodes in a network to deanonymize other nodes in the same network. For example, in the case of the Bitcoin network, a series of works [51,52,56] have shown that by including a few machines in the Bitcoin network it is possible to link Bitcoin transactions to their corresponding source IP addresses. Our results increase the privacy breach resulting from these techniques since if a Bitcoin wallet is deanonymized, the complete cluster (including Ripple and other cryptocurrency wallets) is deanonymized.

Ripple transactions are collected by Ripple validator servers. Similar to Bitcoin, it is possible to further deanonymize Ripple transactions and wallets by deploying a Ripple validator server. As of today, validator servers are run by the core Ripple team (e.g., s-west.ripple.com) and by a few big gateway owners (e.g, SnapSwap). These parties can leverage our heuristics to further deanonymize Ripple wallets, and users are particularly vulnerable to deanonymization by them.

Assume we deploy one Ripple server. Then, a Ripple client can create an IP connection to our deployed server to send us the Ripple transactions. As Ripple transactions are sent in the clear, we can inspect them, and by looking at the *Sender* field (see Table 2.1) it is possible to associate the IP address of the incoming connection to the Ripple wallet specified in the *Sender* field.

This privacy breach can be further exploited to link more than one Ripple wallet to a certain IP address. In detail, assume that different connections from the same IP address submits n transactions $\{t_1, \ldots, t_n\}$, where t_i has a Ripple wallet w_i specified in the *Sender* field. This assumption is realistic: the currently Ripple web clients (e.g., *RippleTrade*) issue all the transactions by default to the same Ripple server. Given this scenario, it is likely that all the w_i are owned by the same person and we can further associate this cluster of wallets to the IP address used to establish the connection with our Ripple server.

Although the possibility to employ an anonymous communication network (e.g., Tor [57]) to forward the transactions to the transaction collecting server has been explored, such techniques are found to be vulnerable to denial of service and blacklisting attacks [58].

2.4 Related Work

Since its inception, questions regarding the security and privacy of the Bitcoin system have attracted interest from the research community. Barber et al. [46] observed that Bitcoin exposes its users to the possible linking of their Bitcoin wallets. Thus, recent works [47–50] have proposed simple heuristics to thwart anonymity in Bitcoin. In a somewhat different direction, other recent works [51, 52] show the possibility of identifying ownership relationships between Bitcoin wallets and IP addresses. Although it is possible to extract lessons from those works, the conceptual differences between cryptocurrencies such as Bitcoin and the Ripple network mandate a dedicated look. Our novel heuristics are focused and have special interest for transaction networks such as Ripple, including the integration of several available cryptocurrencies.

There is limited work studying the evolution, security and privacy of the Ripple network. Di Luzio et al. [38] consider two aspects of the Ripple network. They study the evolution of the amount and behavior of participants in the consensus protocol used to add transactions to the ledger during the first three years of the Ripple network.

They also propose a novel technique to deanonymize the transactions of a given user, leveraging side-channel information (e.g., the amount of a recent transaction performed by the victim).

Armknecht et al. [37] present an overview of the Ripple network and give statistics about the number of transactions, and types of transactions and exchanges. The work is limited to the first two years of operation of the Ripple network. The work also demonstrates the conditions under which the Ripple consensus protocol fails, leading to a situation where the Ripple ledger might be forked. We consider this orthogonal to the content of this dissertation.

3 CREDIT NETWORKS: SECURITY AND PRIVACY

After we have overviewed the Ripple network, an example of credit network deployed in practice, and the possible security and privacy issues with the current deployment, we are in position of building the foundations for credit networks as well as their security and privacy notions of interest.

For starters, we envision a credit network as a combination of two main blocks: *routing* and *graph management*. In a nutshell, routing enables to construct credit paths between two users in the credit network; graph management allows to update the credit network upon operations queried by the users. In particular, *payment* transfers credit between a sender and a receiver through a credit path, *change link* updates the credit held at a credit link, *test credit* calculates the available credit that can be transferred in a payment between any two users in the credit network, and *test link* provides the credit available in a credit link.

In the rest of this chapter, we first formally define the concept of credit network along with the aforementioned operations. We then introduce security and privacy notions of interest in a credit network in the form of ideal functionalities. An ideal functionality represents the expected behavior of each operation in an idealized world, simplifying thereby its description. In subsequent chapters, we detail how to construct a credit network that *realizes* this ideal functionality and thus achieves the expected security and privacy guarantees. More details can be found in [59].

3.1 Credit Network Definition

We denote the security parameter of our system by λ. Let $poly(\cdot)$ and $\nu(\cdot)$ be a polynomial function and a negligible function, respectively. Let $\{A_\lambda\}_{\lambda\in\mathbb{N}}$ and $\{B_\lambda\}_{\lambda\in\mathbb{N}}$ be two distribution ensembles indexed by λ. Then, we say that $A_\lambda \approx B_\lambda$ if for all probabilistic polynomial time algorithms $\mathcal{A}$, there exists a negligible function ν such that

$$|\mathsf{Pr}[\mathcal{A}(x) = 1 \mid x := A_\lambda] - \mathsf{Pr}[\mathcal{A}(x) = 1 \mid x := B_\lambda]| \leqslant \nu(\lambda).$$

A credit network is defined as follows.

Definition 1 (Credit network) *A credit network* $nw := \mathcal{G}(\mathcal{V}, \mathcal{E})$*, where* $\mathcal{V}$ *is the set of users and* $\mathcal{E}$ *is the set of credit links, is a graph equipped with the six operations* $(\mathsf{setup}, \mathsf{route}, \mathsf{pay}, \mathsf{chgLink}, \mathsf{test}, \mathsf{testLink})$ *described below:*

$\mathsf{setup}(1^\lambda) \rightarrow$ *params: On input of a security parameter, output a set of public parameters params.*

$\mathsf{route}($*params*$)$*: On input a set of public parameters, initializes the routing information required by each node in the credit network.*

$\mathsf{pay}(u_1, u_2, v) \rightarrow \{0, 1\}$*: On input of two user identifiers* $u_1, u_2 \in \mathcal{V}$ *and the credit value* v*, if the payment is approved by* u_1 *and if there exists enough credit flow between* u_1 *and* u_2*, perform a payment from* u_1 *to* u_2 *of value* v *and return* 1*. Otherwise, return 0.*

$\mathsf{chgLink}(u_1, u_2, v) \rightarrow \{0, 1\}$*: On input of two user identifiers* u_1, u_2 *and a credit value* v*, if* u_1 *approves the operation, modify the link* $u_1 \rightarrow u_2 \in \mathcal{E}$ *by* v *and return* 1*. Otherwise, return* 0*.*

$\mathsf{test}(u_1, u_2) \rightarrow v$*: On input of two user identifiers* u_1, u_2*, if* u_1 *approves the operation, return the available credit flow between* u_1 *and* u_2*.*

$\mathsf{testLink}(u_1, u_2) \rightarrow v$*: On input of two user identifiers* u_1, u_2*, if one of the users approves the operation, return the credit available in the link* $u_1 \rightarrow u_2$*.*

Correctness For a given credit network nw, let $v \leftarrow \mathsf{test}(\mathsf{u}_i, \mathsf{u}_j)$. A credit network is considered *correct* if the following equalities hold for all $\mathsf{chgLink}$ and pay operations performed on it for any two users u_i and u_j.

- Let nw' be the network obtained after performing $\mathsf{pay}(\mathsf{u}_i, \mathsf{u}_j, v')$ on nw. Then, for the $v'' \leftarrow \mathsf{test}(\mathsf{u}_i, \mathsf{u}_j)$ computed on nw', $v'' = v$ if the pay operation is unsuccessful, else $v'' = v - v'$.

- Let nw' be the resultant network after performing $\mathsf{chgLink}(\mathsf{u}_i, \mathsf{u}_j, v')$ on nw. Then, for the $v'' \leftarrow \mathsf{test}(\mathsf{u}_i, \mathsf{u}_j)$ computed on nw', $v'' = v$ if the $\mathsf{chgLink}$ operation is unsuccessful (due to disapproval by u_i), else $v'' = (v + v')$.

3.2 Security and Privacy in Credit Networks

In this section, we introduce the security and privacy notions of interest in a credit network. As a warm up, we first introduce them informally and later we formally describe them as ideal world functionalities. In particular, here we identify *serializability* as an important system property and *integrity* as a fundamental security property for a credit network. Additionally, we characterize two privacy requirements for transactions: *Value privacy* and *sender/receiver anonymity*. In the following we provide an intuitive description of these properties.

3.2.1 Attacker Model

We consider a decentralized network where the adversary can potentially corrupt, spawn, or impersonate an arbitrary set of users. The adversary is allowed to adaptively choose the set of corrupted parties. This models the fact that the adversary can include her own users in the credit network and that the adversary might also compromise some of the honest users' machines. We consider only passive, but still adaptive, corruption of a minority (less than half of the total set) of the landmark users, which are thus assumed to be honest-but-curious. We assume that the non-corrupted

landmarks execute the algorithms according to our specifications and do not share private information among each other (i.e., they do not collude). In our vision, landmarks represent the root of trust in our network and they can be seen as the network operators (e.g., banks are the natural candidate to serve as landmarks in a transaction system). We advocate that it is in the interest of the landmarks to follow the protocol in order to maintain the availability of their network. That being said, it is easy to extend our definitions to fit a full corruption of the landmark users. Furthermore, we remark that integrity guarantees, i.e., the fact that credit cannot be stolen, must hold unconditionally in our model.

3.2.2 Goals Overview

Integrity A credit network achieves integrity if for all pairs of sender and receiver users $(\mathsf{u}_1, \mathsf{u}_2) \in \mathcal{V}^2$, for all values $v \in \mathbb{N}$, for all successful payments $\mathsf{pay}(\mathsf{u}_1, \mathsf{u}_2, v)$ and for all intermediate honest users $\mathsf{u}_3 \in \mathcal{V}$, such that $\mathsf{u}_3 \neq \mathsf{u}_1$ and $\mathsf{u}_3 \neq \mathsf{u}_2$, the following holds: Let $\mathsf{u}_4, \mathsf{u}_5 \in \mathcal{V}^2$ be the predecessor and successor of u_3 in the payment path. Then, if there exists some credit value $x \in \mathbb{N}$ such that $\mathsf{u}_4 \rightarrow \mathsf{u}_3$ is decreased by x, then $\mathsf{u}_3 \rightarrow \mathsf{u}_5$ is decreased by x.

Serializability Transactions in a credit network are serializable if, for all sets of pay and $\mathsf{chgLink}$ operations *successfully* performed in a concurrent manner, there exists a serial ordering of the same operations with the same outcome (i.e., changes in the credit available in the corresponding paths).

Value Privacy We say that a credit network maintains value privacy if for all pairs of honest users $(\mathsf{u}_1, \mathsf{u}_2) \in \mathcal{V}^2$ and for all pairs of values $(v, v') \in \mathbb{N}^2$ it holds that

$$\mathsf{pay}(\mathsf{u}_1, \mathsf{u}_2, v) \approx \mathsf{pay}(\mathsf{u}_1, \mathsf{u}_2, v')$$

to the eyes of any malicious user not involved in the transaction, as long as both operations are either successful or fail.

Sender/Receiver Anonymity We say that a credit network has sender anonymity if for all pairs of honest users $(\mathsf{u}_0, \mathsf{u}_1, \mathsf{u}_2) \in \mathcal{V}^3$ and for all values $v \in \mathbb{N}$ we have that, for any two simultaneous and successful payments,

$$\mathsf{pay}(\mathsf{u}_0, \mathsf{u}_2, v) \approx \mathsf{pay}(\mathsf{u}_1, \mathsf{u}_2, v)$$

to the eyes of any malicious intermediate user involved either in both or none of the two transactions, such that the honest neighbors of such a corrupted node are the same for both transactions. Receiver anonymity is defined along the same lines.

3.2.3 Formal Definitions

We formally define the security and privacy goals of decentralized credit networks according to the Universal Composability (UC) paradigm [60]. The main idea of this security notion is to compare a real protocol τ with some ideal world Φ, the so-called ideal functionality. The ideal functionality can be seen as a trustworthy entity that implements the intended behavior of the protocol. Given a real protocol τ and an ideal protocol Φ, we say that τ UC-realises Φ if for any adversary $\mathcal{A}$ attacking the protocol τ there is a simulator $\mathcal{S}$ performing an attack on the ideal protocol Φ such that no environment $\mathcal{E}$ can distinguish between τ running with $\mathcal{A}$ and Φ running with $\mathcal{S}$. Here $\mathcal{E}$ may choose the protocol inputs and read the protocol outputs and may communicate with the adversary or simulator (but $\mathcal{E}$ is, of course, not informed whether it communicates with the adversary or the simulator). This is different from the traditional settings in that the environment may communicate with the adversary during the protocol execution and that the environment does not need to choose its inputs at the beginning of the protocol execution. Instead, it may adaptively send inputs to the protocol parties at any time, and it may choose these inputs depending on the outputs and the communication with the adversary. This formalization has the advantage of modelling attacks that exploit parallel instances of the protocol and

therefore it allows one to reason about security also in presence of parallel execution. Due to the distributed nature of credit networks, we believe that it is of paramount importance to capture the presence of interleaving executions also in the definition of security. We informally define UC-Security in the following and we refer the reader to the work of Canetti [60] for a comprehensive discussion on the matter.

Definition 2 (UC-Security) *Let* $\mathsf{EXEC}_{\tau,\mathcal{A},\mathcal{E}}$ *be the ensemble of the outputs of the environment* $\mathcal{E}$ *when interacting with the adversary* $\mathcal{A}$ *and parties running the protocol* τ *(over the randomness of all the involved machines). A protocol* τ *UC-realizes an ideal functionality* Φ *if for any adversary* $\mathcal{A}$ *there exists a simulator* $\mathcal{S}$ *such that for any environment* $\mathcal{E}$ *the ensembles* $\mathsf{EXEC}_{\tau,\mathcal{A},\mathcal{E}}$ *and* $\mathsf{EXEC}_{\Phi,\mathcal{S},\mathcal{E}}$ *are computationally indistinguishable.*

We describe in the following the ideal functionality $\mathcal{F}_{\text{CN}}$, which models the intended behavior of all the components of a credit network, in terms of functionality, security, and privacy. We consider a connected network of n nodes where each node is labeled either as a standard end-user (u) or as a landmark (LM). We denote by landmark a well-connected node in the credit network. For instance, a gateway could carry out the role of a landmark in the Ripple network whereas a bank could be the landmark in the current network of financial institutes. We model the synchronous network as an ideal functionality $\mathcal{F}_{\text{NET}}$ as well as the secure and authenticated channels that connect each pair of neighboring nodes, $\mathcal{F}_{\text{SMT}}$, as proposed in [60]. In our abstraction, messages between honest nodes are directly delivered through $\mathcal{F}_{\text{SMT}}$, i.e., the adversary cannot identify whether there is a communication between two honest users. The attacker can corrupt any instance by a message $\mathsf{corrupt}$ sent to the respective party ID. The functionality $\mathcal{F}_{\text{NET}}$ hands over to the attacker all the static information related to ID. In case ID is a standard node, all its subsequent communication is routed through $\mathcal{A}$, which can reply arbitrarily (active corruption). If ID is a landmark, all its subsequent communication is recorded and the transcripts are given to $\mathcal{A}$ (i.e., thereby modeling passive corruption).

Functionality $\mathcal{F}_{\text{ROUT}}$

1) LM sends to $\mathcal{F}_{\text{ROUT}}$ two tuples of the form $(\mathsf{u}_1, \ldots, \mathsf{u}_m)$, indicating the sets of neighbors of LM in the BFS trees.

2) $\mathcal{F}_{\text{ROUT}}$ runs a BFS algorithm over the links among registered users to construct an arborescence and an anti-arborescence rooted at the landmark $\mathsf{ID}_{\mathsf{LM}}$.

3) Specifically, the algorithm operates on a set of users to be visited, initially set to the one specified by the landmark. For each user u in this set, $\mathcal{F}_{\text{ROUT}}$ sends her a message $(\mathsf{sid}, \mathsf{ID}_{\mathsf{LM}}, h, \mathsf{u}_{\mathsf{p}})$ via $\mathcal{F}_{\text{SMT}}$, where h is the number of hops that separates u from $\mathsf{ID}_{\mathsf{LM}}$ and u_{p} is the parent node on that path. u can either send $(\perp, \mathsf{sid})$, causing $\mathcal{F}_{\text{ROUT}}$ to roll back to the previous user, or $(\mathsf{u}', \mathsf{sid})$ to indicate the next user u' to visit, which is thus added to the set. The algorithm terminates when the set is empty.

Fig. 3.1. Description of the ideal functionality for routing $\mathcal{F}_{\text{ROUT}}$

Ideal Functionality Our ideal functionality for a credit network, $\mathcal{F}_{\text{CN}}$, maintains locally the static information about nodes, credit links, and their credit using a matrix. Additionally, $\mathcal{F}_{\text{CN}}$ logs all of the changes to the credits between nodes that result from successful transactions. $\mathcal{F}_{\text{CN}}$ is composed by a set of functionalities ($\mathcal{F}_{\text{ROUT}}$, $\mathcal{F}_{\text{PAY}}$, $\mathcal{F}_{\text{TEST}}$, $\mathcal{F}_{\text{CHGLINK}}$, $\mathcal{F}_{\text{TESTLINK}}$) that interact as follows: $\mathcal{F}_{\text{CN}}$ periodically executes a functionality to update the routing information of the nodes in the network ($\mathcal{F}_{\text{ROUT}}$) using $\mathcal{F}_{\text{NET}}$ as a mean of synchronization. Nodes can contact the ideal functionality to perform transactions ($\mathcal{F}_{\text{PAY}}$), test the available credit ($\mathcal{F}_{\text{TEST}}$), update the credit on a link ($\mathcal{F}_{\text{CHGLINK}}$) or to test the credit available in a link ($\mathcal{F}_{\text{TESTLINK}}$). Under these assumptions, we describe the routines executed by $\mathcal{F}_{\text{CN}}$ in the following.

$\mathcal{F}_{\text{ROUT}}$ The routing algorithm as described in Figure 3.1 allows the ideal functionality to construct multiple spanning trees (i.e., Breadth-First Search trees) in the credit network, each spanning tree encoding transaction paths between pairs of nodes. The landmark fixes the set of children nodes for the computation of the Breadth-First

Search (BFS) (step 1) and the ideal functionality executes the BFS (steps 2-3) by exchanging messages with each node in the network, starting from the set specified by the landmark. Each node can decide whether to interrupt the algorithm or to indicate the next node to visit. This models possible disruptive users in a distributed credit network. At the end of the execution each node learns its parent from and to the input landmark. Two types of BFS trees are created: *Arborescence* tree considers the credit links in the direction from a landmark to the users; *anti-arborescence* tree considers the credit links in the direction from the users to a landmark.

$\mathcal{F}_{\mathbf{PAY}}$ The algorithm shown in Figure 3.2 constitutes the ideal functionality of the pay operation in a distributed credit network. The protocol is initiated by the Sdr that communicates the two ends of the transaction to the ideal functionality $\mathcal{F}_{\text{PAY}}$ (step 1). For each landmark, $\mathcal{F}_{\text{PAY}}$ derives two paths connecting the sender to the landmark (resp. the receiver to the landmark) in a distributed fashion (step 2): the functionality interacts with each intermediate node that can choose the next node where to route $\mathcal{F}_{\text{PAY}}$, until the landmark is reached (or the maximum length of the path is exceeded). Again, each node along the path can arbitrarily delay the operation and potentially choose any next node to visit, to model possibly malicious nodes. $\mathcal{F}_{\text{PAY}}$ computes then the total amount of credit associated with each of the derived paths and sends the information to the sender (step 3) who can either interrupt the execution or inform $\mathcal{F}_{\text{PAY}}$ of the values to transfer through each path (step 4). $\mathcal{F}_{\text{PAY}}$ informs the nodes of the value transacted through them and the receiver of the total amount of transacted credit (steps 5-6). Each node involved in this phase can either confirm or abort the operation if the transacted amount exceeds the capacity of some link. If all of the nodes accept, $\mathcal{F}_{\text{PAY}}$ updates the credit information of each node involved consistently with the transacted amount. Then $\mathcal{F}_{\text{PAY}}$ informs the set of nodes that participated to the protocol (starting from the receiver) of the operation's success (step 7). This is done again iteratively such that any node can interrupt the communication, if traversed.

Functionality $\mathcal{F}_{\text{PAY}}$

1) For each LM, a sender Sdr sends the tuple $(\mathsf{Sdr}, \mathsf{Rcv}, \mathsf{Txid}, \mathsf{ID}_{\mathsf{LM}})$ to $\mathcal{F}_{\text{PAY}}$, where Rcv, Txid, and $\mathsf{ID}_{\mathsf{LM}}$ denote the receiver, the transaction identifier, and the landmark identifier of the transaction.

2) For each LM, $\mathcal{F}_{\text{PAY}}$ derives the path from Sdr to Rcv, by concatenating the respective paths to LM, as follows: starting from Sdr and Rcv, $\mathcal{F}_{\text{PAY}}$ sends (Txid, $\mathsf{ID}_{\mathsf{LM}}$, u) via $\mathcal{F}_{\text{SMT}}$, where u is the previous user in the chain, if any. Each node can either send $(\perp, \mathsf{Txid}, \mathsf{ID}_{\mathsf{LM}})$, to have $\mathcal{F}_{\text{PAY}}$ ignoring the path, or $(\top, \mathsf{Txid}, \mathsf{ID}_{\mathsf{LM}})$ to let the functionality follow the path constructed by $\mathcal{F}_{\text{ROUT}}$, or $(\mathsf{u}', \mathsf{Txid}, \mathsf{ID}_{\mathsf{LM}})$ to indicate the next user on the path to LM. $\mathcal{F}_{\text{PAY}}$ proceeds until it reaches LM from both ends (or the maximum length of the path is exceeded) and it computes the minimum value v_{LM} among credits of the links on the path to LM.

3) For each LM, $\mathcal{F}_{\text{PAY}}$ calculates the set of tuples $P = \{\mathsf{ID}_{\mathsf{LM}}, v_{\mathsf{LM}}\}$, where v_{LM} is the credit associated to the path from the Sdr to the Rcv through LM ($\mathsf{path}_{\mathsf{LM}}$). $\mathcal{F}_{\text{PAY}}$ sends then (P, Txid) to the Sdr via $\mathcal{F}_{\text{SMT}}$.

4) The Sdr can either abort by sending $(\perp, \mathsf{Txid})$ to $\mathcal{F}_{\text{PAY}}$ or send a set of tuples $(\mathsf{ID}_{\mathsf{LM}}, x_{\mathsf{LM}}, \mathsf{Txid})$ to $\mathcal{F}_{\text{PAY}}$ via $\mathcal{F}_{\text{SMT}}$.

5) For each LM, $\mathcal{F}_{\text{PAY}}$ informs all the nodes in $\mathsf{path}_{\mathsf{LM}}$ of the value x_{LM} by sending $(x_{\mathsf{LM}}, \mathsf{ID}_{\mathsf{LM}}, \mathsf{Txid})$ via $\mathcal{F}_{\text{SMT}}$. Each node can either send $(\perp, \mathsf{ID}_{\mathsf{LM}}, \mathsf{Txid})$ to abort the transaction, or $(\mathsf{accept}, \mathsf{ID}_{\mathsf{LM}}, \mathsf{Txid})$ to carry out the transaction. In the latter case $\mathcal{F}_{\text{PAY}}$ checks whether for the corresponding edge $\mathsf{e} : v_{\mathsf{e}} \geqslant x_{\mathsf{LM}}$, and if yes $\mathcal{F}_{\text{PAY}}$ subtracts x_{LM} from v_{e}. If one of the conditions is not met or there is at least one $(\perp, \mathsf{ID}_{\mathsf{LM}}, \mathsf{Txid})$ message, then $\mathcal{F}_{\text{PAY}}$ aborts the transaction and restores the credits on the corresponding links of $\mathsf{path}_{\mathsf{LM}}$.

6) $\mathcal{F}_{\text{PAY}}$ sends to Rcv the tuple $(\mathsf{Sdr}, \mathsf{Rcv}, v, \mathsf{Txid})$ via $\mathcal{F}_{\text{SMT}}$, where v is the total amount transacted to Rcv. Rcv can either abort the transaction by sending $(\perp, \mathsf{Txid})$ or allow it by sending $(\mathsf{success}, \mathsf{Txid})$.

7) For each LM, $\mathcal{F}_{\text{PAY}}$ sends either $(\mathsf{success}, \mathsf{Txid})$ (or $(\perp, \mathsf{Txid})$ depending on the outcome of the transaction) to each user in the path from the Rcv to the Sdr, starting from the Rcv. Such a user can either reply with $(\perp, \mathsf{Txid})$ to conclude the functionality or with $(\mathsf{accept}, \mathsf{Txid})$ to have $\mathcal{F}_{\text{PAY}}$ passing the message $(\mathsf{success}, \mathsf{Txid})$ (or $(\perp, \mathsf{Txid})$) to the next user until Sdr is reached.

Fig. 3.2. Description of the ideal functionality for payments $\mathcal{F}_{\text{PAY}}$

$\mathcal{F}_{\textbf{TEST}}$ The test algorithm computes the credit available on the paths connecting any two nodes in the network, and it works analogously to the steps 1-3 in $\mathcal{F}_{\text{PAY}}$.

$\mathcal{F}_{\textbf{TESTLINK}}$ At any point of the execution, each node can query the $\mathcal{F}_{\text{TESTLINK}}$ functionality to obtain informations about her adjacent links.

$\mathcal{F}_{\textbf{CHGLINK}}$ Each pair of neighboring nodes can jointly query $\mathcal{F}_{\text{CHGLINK}}$ to update their link or generate a new one.

3.2.4 Discussion

In the following we motivate our choices in the design of the ideal functionality for a distributed credit network. First of all, we point out that in the transaction functionality $\mathcal{F}_{\text{PAY}}$ we let each node decide 'on the fly' the next node where to route the transaction in the path from the sender to the receiver: this captures the distributed nature of the network where each node can route transactions arbitrarily. Nevertheless, note that any malicious attempt to redirect the transaction would fail unless the receiver is eventually reached, in which case the functionality of the network is not harmed. We do not see any reason why a node should not have the capability to switch between paths if it wishes to. It must be also pointed out that malicious nodes cannot cause the ideal functionality to run indefinitely on a path: $\mathcal{F}_{\text{PAY}}$ will ignore the path after a certain maximum length is reached and the landmark is not an intermediate node. Another controversial point is the possibility for each node to cause an abort of the transaction that traverses it at several points of the executions. In this case a similar reasoning as above holds: we first note that such an attack is confined to links that the adversary is connected to, so it would require to establish many trust relationships with honest nodes to carry out a denial of service on a large scale. Additionally, we believe that each node must be able to decide whether it wants to take part to a transaction: although its total balance remains intact, some credit is shifted from one node to another and this may be undesirable. Such a behavior

can also easily be detected by other nodes in the network who can eventually route subsequent transactions to other paths not traversing the faulty node.

What is left to be shown is that our ideal functionality captures the security and privacy properties that one would expect for a credit network.

Integrity In the ideal world, integrity is guaranteed by the ideal functionality, who maintains a database of the link values and updates them consistently with the successful transactions.

Serializability We observe that any set of chgLink operations on the same link is executed serially by the ideal functionality. Assume for the moment that only chgLink operations are performed: as any two concurrent operations are necessarily executed on two different links, it is easy to find a scheduler that returns the same outcome by performing those operation in some serial order (i.e., any order). Since a pay operation can be represented as a set of chgLink operations performed atomically (due to the integrity notion), the property follows.

Value Privacy We observe that the only information revealed to the nodes about a transaction is the value of the transaction that traverses them (while the total amount of transferred credit is kept local by the ideal functionality). It is unavoidable to leak this information to each node since it affects its direct links and thus the leakage for the transaction value in our protocol is optimal.

Sender/Receiver Anonymity For sender/receiver anonymity it is enough to observe that the ideal functionality addresses each transaction with a uniformly sampled id that does not contain any information about the identity of the sender nor of the receiver. Thus in the ideal world each user does not learn any information beyond the fact that some transaction has traversed some of her direct links, which is inevitable to disclose.

Part II

SECURE AND PRIVACY-PRESERVING TRANSACTION PROTOCOLS

4 PATHSHUFFLE: ANONYMITY FOR RIPPLE TRANSACTIONS

While the Ripple network offers several benefits to the current financial industry, the public nature of its transaction ledger exposes its individual users, groups, organizations, and companies to the same severe privacy attacks as already observed in Bitcoin. The privacy study of the Ripple network described in Section 2.3 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. In this state of affairs, we require a privacy solution for the currently deployed Ripple network.

In order to fill this gap, in this chapter we overview PathShuffle, the first mixing protocol for path-based transactions in credit networks that is fully compatible with the current Ripple network. A full description can be found in [61].

4.1 Cryptographic Background and Notation

In this section, we introduce the notation that we follow in the rest of this chapter.

4.1.1 Ripple Network Operations

We use the operations available in the current Ripple network as described in Table 4.1. A transaction tx becomes valid when it is signed by the appropriate wallet's signing key. A transaction tx from $\mathsf{CreateTx}$ and $\mathsf{ChangeLink}$ must be signed by sk_1 (i.e., the signing key of wallet vk_1), whereas a transaction tx from $\mathsf{CreateLink}$ must be signed by sk_2. Finally, a transaction tx from $\mathsf{testLink}$ does not require a signature. A transaction tx is applied to the Ripple network after invoking $\mathsf{Apply}(\mathsf{tx}, \sigma)$ with

Table 4.1.
Description of the Ripple network operations.

Operation	Description
$(\mathsf{vk}, \mathsf{sk}) := \mathsf{AccountGen}(\,)$	Generate wallet keys
$\mathsf{tx} := \mathsf{CreateTx}(\mathsf{vk}_1, \mathsf{vk}_2, v)$	Create path-based transaction
$\mathsf{tx} := \mathsf{CreateLink}(\mathsf{vk}_1, \mathsf{vk}_2, v)$	Create link $\mathsf{vk}_1 \rightarrow \mathsf{vk}_2$ (limit v)
$\mathsf{tx} := \mathsf{ChangeLink}(\mathsf{vk}_1, \mathsf{vk}_2, v)$	Modify link $\mathsf{vk}_1 \rightarrow \mathsf{vk}_2$ by v
$\{v, \perp\} := \mathsf{testLink}(\mathsf{vk}_1, \mathsf{vk}_2)$	Query IOU on link $\mathsf{vk}_1 \rightarrow \mathsf{vk}_2$
$\{0, 1\} := \mathsf{Apply}(\mathsf{tx}, \sigma)$	Apply signed *transaction* to network

the correct signature. For simplicity, with $\mathsf{Apply}(\mathsf{tx}, \sigma)$ we abstract away the process of being written to the blockchain after agreed upon consensus nodes. Moreover, for clarity of exposition, we assume that $\mathsf{Apply}(\mathsf{tx}, \sigma)$ returns immediately after tx is applied to the Ripple network. In practice, tx is applied in a matter of seconds [36].

4.1.2 Digital Signature Scheme

A digital signature scheme allows a *signer* who has established a public key vk to sign a message m using the associated secret key sk and creating thereby a signature σ. A digital signature further allows then anyone with access to the message m, the signature σ and the public key vk to verify that the *signer* correctly signed the message m. A bit formally, a digital signature scheme Π consists of three algorithms $(\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ defined as follows:

$\mathsf{KeyGen}(\lambda) \rightarrow \mathsf{sk}, \mathsf{vk}$**:** The key generation algorithm takes as input the security parameter λ and returns a pair of public key vk and secret key sk.

$\mathsf{Sign}(\mathsf{sk}, \mathsf{m}) \rightarrow \sigma$**:** The signing algorithm takes as input a secret key sk and a message m and returns a signature σ.

$\mathsf{Verify}(\mathsf{vk}, \mathsf{m}, \sigma) \rightarrow \{0,1\}$**:** The verification algorithm takes as input a public key vk, a message m and a signature σ and returns 1 if σ is a valid signature on message m. Otherwise, it returns 0.

Security of a Digital Signature Scheme Given a fixed public key vk generated by a *signer*, a *forgery* consists of a message m^* along with a valid signature σ^*, where m^* has not been previously signed by the signer. Now, security intuitively means that it should be infeasible for anybody not possessing the corresponding secret key sk, to produce a forgery. We refer the reader to [62] for a formal security definition in the Universal Composability framework.

4.1.3 Distributed Digital Signature Scheme

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 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 verification key.

A bit more formally, a distributed digital signature scheme consists of three protocols ($\mathsf{SAccountCombine}$, SSign, Verify) defined as follows:

$\mathsf{SAccountCombine}(\mathsf{vk}_1, \ldots, \mathsf{vk}_n) \rightarrow \mathsf{vk}_s$**:** The distributed key generation protocol takes as input a set of verification keys $\mathsf{vk}_1, \ldots, \mathsf{vk}_n$ and returns the combined public key vk_s.

$\mathsf{SSign}(\mathsf{sk}_1, \ldots, \mathsf{sk}_n, m) \rightarrow \sigma$**:** The distributed signing protocol takes as input a set of signing keys $\mathsf{sk}_1, \ldots, \mathsf{sk}_n$ and returns a signature σ.

$\mathsf{Verify}(\mathsf{vk}_s, m, \sigma) \rightarrow \{0,1\}$**:** The verification algorithm is defined as for the digital signature scheme.

We refer the reader to [61] for a detailed description of $\mathsf{SAccountCombine}$ and SSign as well as its security discussion.

4.1.4 Shared Wallet

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 the distributed signature scheme described in the previous section to achieve this functionality. In a bit more detail, a shared wallet is created as follows. First, each user locally creates a fresh Ripple wallet $(\mathsf{vk}_i^*, \mathsf{sk}_i^*)$, using $\mathsf{AccountGen}$, that constitutes her share for the shared wallet vk_s. The shared wallet can be then calculated as $\mathsf{vk}_s := \mathsf{SAccountCombine}(\{\mathsf{vk}_i^*\})$, where $\{\mathsf{vk}_i^*\}$ denotes the set containing one verification key share vk_i^* 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, the users jointly execute $\sigma := \mathsf{SSign}(\{\mathsf{sk}_i^*\}, m)$, where $\{\mathsf{sk}_i^*\}$ is the set containing one signing key share sk_i^* for each user, and m is the message to be signed. This protocol returns a signature σ. This signature can be then verified locally by every user by invoking $\mathsf{Verify}(\mathsf{vk}_s, m, \sigma)$.

4.2 Preliminaries

In this section, we first present *path mixing* (Section 4.2.1), our approach to improve anonymity in credit networks. We then describe the communication model (Section 4.2.2), the security and privacy goals (Section 4.2.3), and the threat model (Section 4.2.4)).

4.2.1 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.

In this setting, a path mixing protocol aims to transfer 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*.

Towards achieving that, the path mixing protocol must *only* require functionality already available in credit networks.

4.2.2 Setup and Communication Model

We assume that users communicate to each other through a bulletin board, e.g., a server that receives a message from a user and broadcasts it to the rest of users. We require that the protocol achieves anonymity even if the bulletin board is malicious and colludes with the attacker. We thus consider the bulletin board only as an efficient mean of communication. 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 [63] for details). The bulletin board can be alternatively implemented by a reliable broadcast protocol [64, 65] at an increased communication cost.

We assume that users participating in the 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 protocol.

Finally, we assume that there is a bootstrapping mechanism in place for users to know other users willing to carry out the 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 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.

4.2.3 Security and Privacy Goals

In this chapter, we consider a subset of the security and privacy goals that we have established in Chapter 3. They are clearly a restricted version but still capture fundamental security and privacy properties of credit networks. Moreover, they open the possibility to have a protocol fully compatible with the current Ripple network.

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

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

These security and privacy goals are in tune with those presented in Chapter 3 but with the limitations inherent to a path mixing approach. In particular, unlinkability is a weaker privacy notion than sender/receiver privacy: The set of possible sender and receivers for a given transaction is limited to the set of participants in one instance

of the path mixing protocol. The integrity property remains the same as presented in Chapter 3: As integrity deals with the total balance of honest users, a weaker notion directly leads to unacceptable credit loss by honest users. Finally, we do not consider serializability here as it is directly provided by the consensus protocol available in the underlying credit network. We do not consider value privacy either as it is at odds with an underlying public ledger that logs all the transactions and their values.

4.2.4 Threat Model

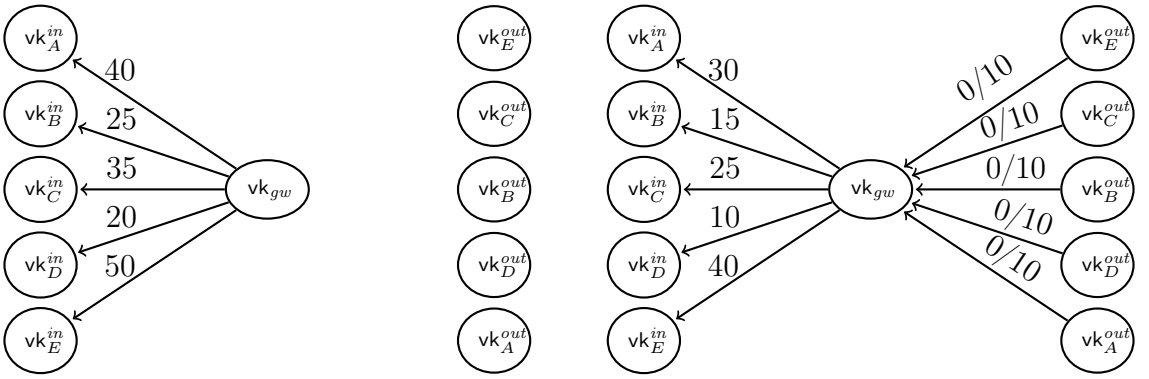
We assume that the attacker controls an arbitrary number f of users participating in the path mixing protocol. We further assume that the attacker also controls the bulletin board (and thus the network). The *anonymity set* of an honest user is the set of all honest users. Thus, in order to achieve any meaningful anonymity guarantee, we need that $f < n - 1$. In other words, we do not consider the $n - 1$ attack [66] in this work.

4.3 Solution Overview

In this section, we first show a straw man approach for path mixing (Section 4.3.1) to illustrate the challenges we have to overcome. Then, we overview the two building blocks of our approach: atomic transactions in Ripple (Section 4.3.2) and the creation of a set of wallets anonymously (Section 4.3.3).

4.3.1 A Straw Man Path Mixing Approach

Path mixing can be achieved following a straw man approach as shown in Figure 4.1. 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.



(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. 4.1. 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 settlement transaction for up to 10 IOU using vk_A^{out} and vk_{gw} as the first hops in the transaction path.

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 Figure 4.1(a). 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 Figure 4.1(b)).

For every user i, the gateway must create a credit link from the output wallet vk_i^{out} to its own wallet of the form $\mathsf{vk}_{gw} \leftarrow \mathsf{vk}_i^{out}$ (i.e., vk_i^{out} 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 $\mathsf{vk}_{gw} \rightarrow \mathsf{vk}_i^{out}$. Such operation must be confirmed with a signature by the user

i (see Section 4.1.1). 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 vk_i^{out} belongs to user i. As the attacker also knows the input wallet belonging to user i, he trivially breaks the unlinkability property.

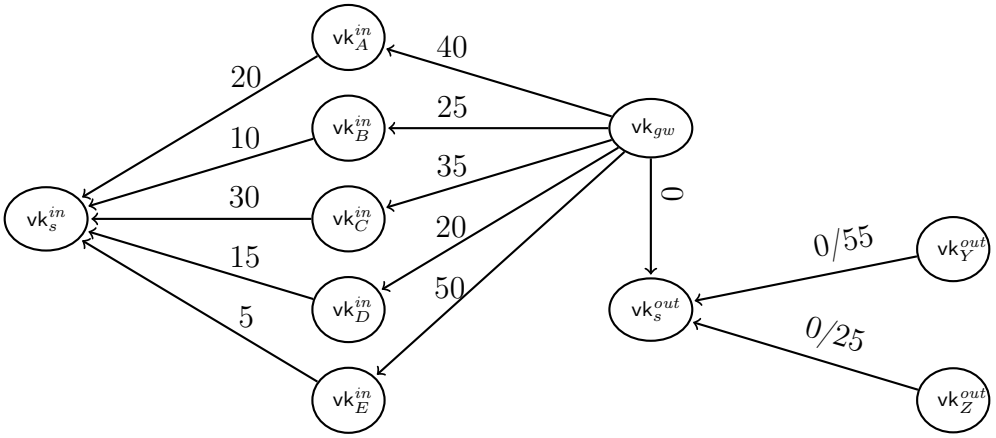
In this straw man approach, the server is trusted for unlinkability and integrity properties. First, the server must be trusted not to reveal the pair of input and output wallets belonging to a user. Second, after receiving the credit from the users' input wallets, the server is trusted not to steal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

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: *Atomic transactions* and *creating a set of output wallets in an anonymous manner*.

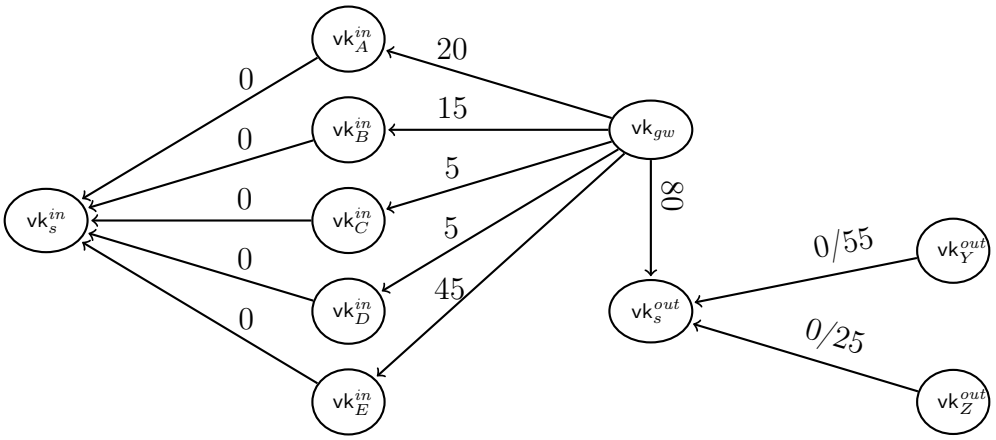
4.3.2 Atomic Transactions in Ripple

Assume a generic setting with a set of n input wallets $\{\mathsf{vk}_i^{in}\}$ and a set of m output wallets $\{\mathsf{vk}_j^{out}\}$. Moreover, assume that instead of a fixed amount of credit β, each input wallet must transfer β_i^{in} IOU and each output wallet must receive β_j^{out} 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., $\sum_i \beta_i^{in} = \sum_j \beta_j^{out}$). In such setting, PathShuffle, our novel protocol to enforce *atomic* transactions *fully compatible* with Ripple, must ensure that either all the $\sum_i \beta_i^{in}$ 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



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



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

Fig. 4.2. An illustrative example of an atomic transaction. The input wallets vk_A^{in}, vk_B^{in}, vk_C^{in}, vk_D^{in} and vk_E^{in} transfer 20, 10, 30, 15 and 5 IOU correspondingly. The output wallets vk_Y^{out} and vk_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 PathShuffle, it is possible to perform a settlement transaction from the output wallets (e.g., from vk_Y^{out} for up to 55 IOU using vk_s^{out}, and vk_{gw} as the first hops in the transaction path).

performed. This effectively allows to add one synchronization round: Each user i first transfers β_i^{in} IOU to a shared wallet and only when $\sum_i \beta_i^{in}$ 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 Figure 4.2: Five users with input wallets $\{\mathsf{vk}_A^{in}, \mathsf{vk}_B^{in}, \mathsf{vk}_C^{in}, \mathsf{vk}_D^{in}, \mathsf{vk}_E^{in}\}$ would like to transfer $\{20, 10, 30, 15, 5\}$ IOU into two output wallets $\{\mathsf{vk}_Y^{out}, \mathsf{vk}_Z^{out}\}$. These two output wallets must receive $\{55, 25\}$ IOU. To achieve that, in the first round users jointly create a credit link from each input wallet (vk_i^{in}) to vk_s^{in} with $\beta_{in}[i]$ IOU on them. Moreover, users jointly create a credit link from each of the output wallets (vk_j^{out}) to vk_s^{out} with no IOU on them but an upper limit of β_j^{out}. At this point, credit at each vk_j^{out} cannot be issued as part of a settlement transaction because vk_s^{out} does not have incoming credit yet (see Figure 4.2(a)). 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 $\sum_i \beta_i^{in}$ IOU. Then, vk_s^{out} gets enough credit that can be used by each of the output wallets vk_j^{out} (see Figure 4.2(b)).

4.3.3 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.

4.4 Protocol Details

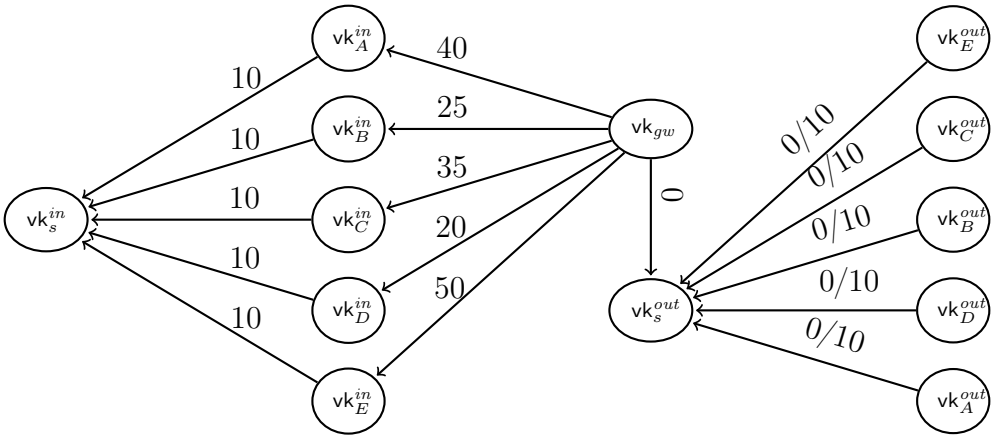
In this section, we describe the details of PathShuffle, our novel protocol for atomic, anonymous transactions in credit networks (Section 4.4.1). Additionally, we describe possible extensions and applications (Section 4.4.2). We analyze the security and privacy guarantees of PathShuffle (Section 4.4.3). Finally, we evaluate the performance of PathShuffle (Section 4.4.4).

4.4.1 Protocol Description

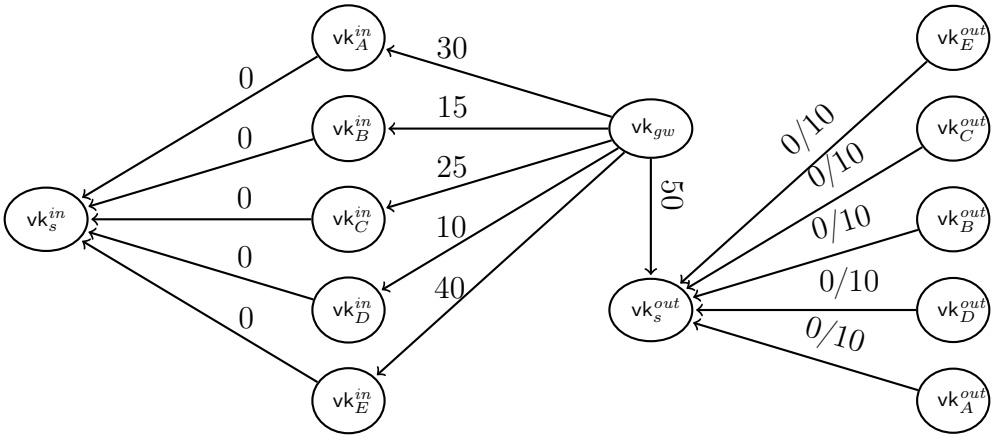
The PathShuffle protocol works as described below. We first describe our assumptions and then we detail each phase of the protocol. We depict a sample execution for PathShuffle in Figure 4.3. Finally, a detailed pseudocode for the protocol is presented in Algorithm 1.

Assumptions We assume that the users have agreed on β, the amount of IOU (in some currency) to be mixed in the path mixing (i.e., $\forall_i \beta_i^{in} = \beta$). We further assume that the users have agreed on a common wallet (i.e., vk_{gw}) and that each user has a credit link $\mathsf{vk}_i^{in} \leftarrow \mathsf{vk}_{gw}$ with at least β IOU. 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.

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.



(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. 4.3. 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 settlement transaction for up to 10 IOU using vk_s^{out} and vk_{gw} as first hops in the transaction path.

Algorithm 1 PathShuffle protocol.

Input :
- $(\mathsf{sk}_i^{in}, \mathsf{vk}_i^{in})$: User i input wallet's keys
- $(\mathsf{sk}_i^{out}, \mathsf{vk}_i^{out})$: User i output wallet's keys
- β, sid: Agreed amount of IOU to mix and session identifier
- vk_{gw}: Agreed gateway wallet

1: {Exchange output wallets anonymously}
2: $\{\mathsf{vk}_{\Pi(i)}^{out}\} := \mathsf{Shuffle}(\{\mathsf{vk}_i^{out}\})$
3: {Create shares for shared wallets and broadcast them}
4: **for** $i \in \{1, \ldots, n\}$ **do**
5: $\quad(\mathsf{vk}_i^{*in}, sk_i^{*in}) := \mathsf{AccountGen}()$
6: $\quad(\mathsf{vk}_i^{*out}, sk_i^{*out}) := \mathsf{AccountGen}()$
7: $\quad$**broadcast** $(\mathsf{vk}_i^{*in}, \mathsf{vk}_i^{*out}, \mathsf{Sign}(\mathsf{sk}_i^{in}, (\mathsf{vk}_i^{*in}, \mathsf{vk}_i^{*out}, \mathsf{sid})))$
8: **end for**
9: {Create shared wallets }
10: $\mathsf{vk}_s^{in} := \mathsf{SAccountCombine}(\{\mathsf{vk}_i^{*in}\})$
11: $\mathsf{vk}_s^{out} := \mathsf{SAccountCombine}(\{\mathsf{vk}_i^{*out}\})$
12: {Create credit links $\mathsf{vk}_i^{in} \rightarrow \mathsf{vk}_s^{in}$ and fund them}
13: **for** $i \in \{1, \ldots, n\}$ **do**
14: $\quad\textsc{link}_i^{in} := \mathsf{CreateLink}(\mathsf{vk}_i^{in}, \mathsf{vk}_s^{in}, \infty)$
15: $\quad\sigma_i^{in} := \mathsf{SSign}(\{sk_i^{*in}\}, \textsc{link}_i^{in})$
16: $\quad\mathsf{Apply}(\textsc{link}_i^{in}, \sigma_i^{in})$
17: $\quad\textsc{link}_i^{'in} := \mathsf{ChangeLink}(\mathsf{vk}_i^{in}, \mathsf{vk}_s^{in}, \beta)$
18: $\quad\sigma_i^{'in} := \mathsf{Sign}(\mathsf{sk}_i^{in}, \textsc{link}_i^{'in})$
19: $\quad\mathsf{Apply}(\textsc{link}_i^{'in}, \sigma_i^{'in})$
20: **end for**
21: {Verify credit link $\mathsf{vk}_i^{in} \rightarrow$ for every user }
22: **for** $i \in \{1, \ldots, n\}$ **do**
23: $\quad v := \mathsf{testLink}(\mathsf{vk}_i^{in}, \mathsf{vk}_s^{in})$
24: $\quad$**if** $v = \perp \vee v < \beta$ **then abort end if**
25: **end for**
26: {Create credit links $\mathsf{vk}_i^{out} \rightarrow \mathsf{vk}_s^{out}$}
27: **for** $i \in \{1, \ldots, n\}$ **do**
28: $\quad\textsc{link}_i^{out} := \mathsf{CreateLink}(\mathsf{vk}_i^{out}, \mathsf{vk}_s^{out}, \beta)$
29: $\quad\sigma_i^{out} := \mathsf{SSign}(\{sk_i^{*out}\}, \textsc{link}_i^{out})$
30: $\quad\mathsf{Apply}(\textsc{link}_i^{out}, \sigma_i^{out})$
31: **end for**
32: {Create credit link $\mathsf{vk}_{gw} \rightarrow \mathsf{vk}_s^{out}$}
33: $\textsc{link}_{gw} := \mathsf{CreateLink}(\mathsf{vk}_{gw}, \mathsf{vk}_s^{out}, \infty)$
34: $\sigma_{gw} := \mathsf{SSign}(\{sk_i^{*out}\}, \textsc{link}_{gw})$
35: $\mathsf{Apply}(\textsc{link}_{gw}, \sigma_{gw})$
36: {Final settlement transaction}
37: $\mathsf{tx} := \mathsf{CreateTx}(\mathsf{vk}_s^{in}, \mathsf{vk}_s^{out}, \beta \cdot n)$
38: $\sigma_{\mathsf{tx}} := \mathsf{SSign}(\{sk_i^{*in}\}, \mathsf{tx})$
39: $\mathsf{Apply}(\mathsf{tx}, \sigma_{\mathsf{tx}})$

Phase 0: Exchange Output Wallets Anonymously Several P2P mixing protocols proposed in the literature implement a permutation that ensures that output messages (i.e., wallets in this case) cannot be linked to their owners, as required in our decentralized path mixing protocol. Among them, we decide to use DiceMix [63] due to its efficiency, but in principle we could have used any P2P mixing protocol. Given that PathShuffle is parametric to the P2P mixing protocol, we omit its details here and refer the reader to [63] for a detailed description. In the rest, we denote a P2P mixing protocol by $\mathsf{Shuffle}$.

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 2.2) must be disabled at each credit link with vk_s^{in} wallet.

Then, users jointly create a credit link from each input wallet vk_i^{in} 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 i locally creates and signs a transaction that issues β_i^{in} credit to the recently created link $\mathsf{vk}_i^{in} \rightarrow \mathsf{vk}_s^{in}$. 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 vk_j^{out} to vk_s^{out} with an

upper limit of β_j^{out}. Moreover, the users jointly create a link $\mathsf{vk}_{gw} \rightarrow \mathsf{vk}_s^{out}$ with no IOU on it. These links will later allow to transfer up to β_j^{out} IOU from the wallet vk_j^{out}.

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 settlement transaction transferring $\sum_j \beta_j^{out}$ IOU from vk_s^{in} to vk_s^{out}. This settlement 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 settlement 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 $\sum_j \beta_j^{out}$ IOU. This fact enables now settlement transactions from each output wallet to the rest of the credit network.

4.4.2 Extensions and Applications

Other Credit Networks We have focused the description of PathShuffle 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 PathShuffle. For instance, PathShuffle 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 [67]. Moreover, Stellar implements a mechanism to enable and disable the rippling option as in Ripple [68]. Finally, Stellar supports the same digital signature schemes as Ripple and thus shared wallets can also be implemented in Stellar.

Crowdfunding Application 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 Figure 4.2 is indeed a crowdfunding transaction where the five input wallets are used to fund the two output wallets. PathShuffle ensures that either every user participating in the crowdfunding transfers the expected amount of IOU into the crowdfunding wallets (e.g., vk_Y^{out} and vk_Z^{out}) or none of the users transfers any IOU.

4.4.3 Security and Privacy Analysis

In this section, we argue why PathShuffle achieves the security and privacy goals of interest. We refer the reader to [61] for a more detailed security and privacy analysis.

Correctness The final transaction ensures that exactly β are transferred through the input wallet of the user i (i.e., vk_i^{in}). Moreover, the upper limit on the links from each output wallet to vk_s^{out} ensures that wallet vk_j^{out} has only access to vk_j^{out} IOU. This demonstrates the correctness of PathShuffle.

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 PathShuffle 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 settlement 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 settlement transaction with path: $\mathsf{vk}_{malicious} - \mathsf{vk}_s^{in} - \mathsf{vk}_{honest} - \mathsf{vk}_{gw} - \mathsf{vk}_{malicious}$. (Circular transactions are accepted and used in the Ripple network. For example, a transaction of the form vk_i –

vk_{gw_1}– . . . – vk_{gw_2}– vk_i, where . . . denotes an arbitrary set of wallets, can be used by user i to exchange IOU from gateway 1 to gateway 2.)

Third, the settlement 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 settlement 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 $\{\mathsf{vk}_i^{in}\}$ and vk_{gw} is not used and can be reused in another invocation of PathShuffle. Finally, the links between $\{\mathsf{vk}_i^{in}\}$ 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.

Unlinkability PathShuffle relies on a secure P2P mixing protocol to construct the list of output wallets and 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 for the rest of the PathShuffle protocol shows that operations on PathShuffle 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, PathShuffle does not leak the owner of any output wallet. This shows that PathShuffle achieves unlinkability.

Integrity The underlying P2P mixing protocol does not perform any operation involving the credit of the users. Moreover, the underlying P2P mixing protocol ensures for each user that the rest of the P2P mixing protocol is only called if the list of output wallets contains her own output wallet. Thus if the PathShuffle succeeds, the same amount β of IOU that is taken from her input wallet is transferred to her

output wallet. If PathShuffle fails, no IOU is transferred at all. This shows that PathShuffle achieves integrity.

4.4.4 Performance Analysis

In PathShuffle, we use the DiceMix protocol as defined in the original paper [63] as the underlying P2P mixing protocol. Thus, in this section we restrict our analysis to the additional operations required by PathShuffle and the performance analysis for the core of DiceMix described in [63] carries over in our work.

Implementation We have implemented PathShuffle in JavaScript by modifying the current Ripple code [69]. 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 [70] to implement the submission of transactions to the Ripple network. Our source code is publicly available [71] 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 PathShuffle 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 $\mathsf{SSign}(\dots)$ 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 $\mathsf{SSign}(\dots)$ procedure requires two broadcast rounds (see Section 4.1.3): 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 $\mathsf{SSign}(\dots)$ costs only one extra communication round.

Communication Overhead A protocol based on DiceMix needs $(c+3)+(c+1)f$ communications rounds, where c is number of communication rounds required by $\mathsf{Confirm}(\dots)$ 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 PathShuffle is invoked. Then, one communication round is required for each of the two times $\mathsf{SSign}(\dots)$ is invoked: First to jointly sign the creation of the links $\mathsf{vk}_i^{in} \to \mathsf{vk}_s^{in}$, $\mathsf{vk}_{gw} \to \mathsf{vk}_s^{out}$, and $\mathsf{vk}_j^{out} \to \mathsf{vk}_s^{out}$; 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 PathShuffle are deterministically defined from the input of the protocol, the signatures on the funding transactions for the links $\mathrm{VK}_{in}[i] \to \mathsf{vk}_s^{in}$ can be broadcast the first time $\mathsf{SSign}(\dots)$ is invoked.

Computation Overhead In this evaluation, 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 [36]. 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 PathShuffle.

Running 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 [63].

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) [63].

In PathShuffle, the execution time is dominated by the $\mathsf{Apply}(\ldots)$ operations. Although PathShuffle 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 PathShuffle, we expect that PathShuffle provides 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 Figure 4.3. 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 the details in the original paper [61].

4.5 Anonymity Protocols for Bitcoin

In the realm of Bitcoin and other cryptocurrencies, several solutions have been proposed to achieve anonymous payments [46,63,72–84]. These solutions are tailored to the specifics of blockchain-based cryptocurrencies. Given the fundamental differences between credit networks and cryptocurrencies, it remains an interesting future work to study whether it is feasible to adapt the underlying ideas of these solutions to credit networks.

For example, it is conceivable that simple centralized mixing protocols such as Mixcoin [75] and Blindcoin [77], 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. All existing theft-resistant mixing protocols for cryptocurrencies either rely on multi-input-multi-output transactions or on script-based smart contracts, 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.

5 DECENTRALIZED CREDIT NETWORKS

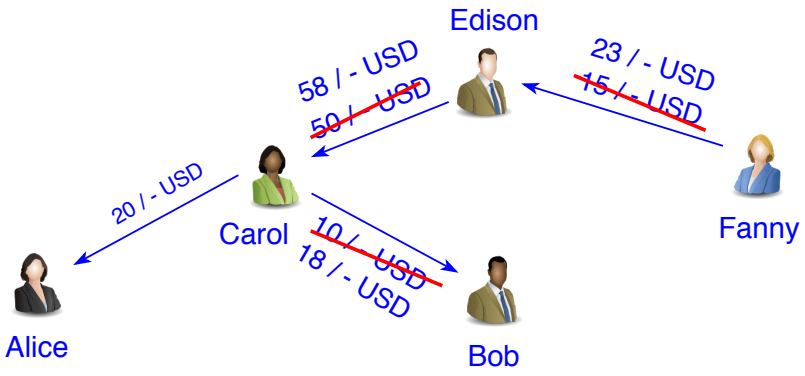
As we described in Section 2.3, the Ripple ledger maintains a log of every credit link between the wallets in the credit network and all the transactions settled between the wallets, what becomes the key source of several privacy issues. In this state of affairs, the following question naturally arises: *Is it possible to build a credit network without a ledger?*

We observe that the net credit balance of a user in a credit network (i.e., difference between the credit that the user is owed and the credit the user owes to others) is fully determined by her own credit links. Therefore, a user can ensure that she does not incur in credit loss solely checking that her net balance does not change without her being a sender or receiver of a transaction.

The illustrative example depicted in Figure 5.1 shows the net balances for each user before and after the transaction from Fabi to Bob for a credit value of 8 IOU (with no fees for clarity of explanation). Although inflow and outflow values change for all transaction participants, intermediate users Carol and Edward maintain their net balance intact. The net balance only changes for the sender Fabi (it is reduced by 8 IOU) and the receiver Bob (it is increased by 8 IOU).

We build upon the aforementioned observation to create a decentralized credit network where users locally store their own credit links. In the example depicted in Figure 5.1, Alice locally stores the credit value on her link with Carol. Similarly, Carol locally stores the credit on her links with Alice, Bob and Edward. The rest of users locally store their credit links in a similar manner.

In such setting, where a complete ledger is no longer available, several challenges arise. *How to calculate the path between sender and receiver of a transaction? How*



		Al	Bo	Ca	Ed	Fa
Prior	Inflow	20	10	50	15	0
	Outflow	0	0	30	50	15
	Net	20	10	20	-35	-15
Posterior	Inflow	20	18	58	23	0
	Outflow	0	0	38	58	23
	Net	20	18	20	-35	-23

Fig. 5.1. Illustrative example of net balance conservation. Fanny performs a settlement transaction with Bob for a credit value of 8 IOU. Although inflow and outflow change, net balance is maintained for intermediate users (e.g., Carol and Edison). Values crossed in red denoted stale values due to the settlement transaction.

to calculate the credit available between two users to perform a transaction? How to perform a transaction involving several users? And finally, *how to carry out all the previous operations while preserving privacy?*

We answer these questions in SilentWhispers [59], an architecture to enforce security and privacy in decentralized credit networks. In such settings, a transaction is jointly executed by all the users involved in such transaction, who contribute their locally stored credit links in a privacy-preserving manner. In this manner, assuming that participating users are online, SilentWhispers ensures that privacy properties of interest, such as transaction value privacy, sender and receiver privacy as well as link privacy, are preserved.

5.1 Cryptographic Background and Notation

We start with an intuitive description of the cryptographic primitives that we have deployed in SilentWhispers. We describe here only the additional primitives that were not shown in Section 4.1.

5.1.1 Secret Sharing

A secret sharing scheme [85] allows a dealer to distribute shares of a secret among different parties such that any arbitrary subset of shares above the threshold allows a receiver to fully reconstruct the secret. We refer to such a sharing mechanism as a (t, N)-threshold secret sharing scheme. A bit more formally, a secret sharing scheme $\mathcal{T}$ consists of two algorithms $(\mathsf{Share}, \mathsf{Reconstruct})$ defined as follows:

$\mathsf{Share}(s) \rightarrow [\![s_1, \ldots, s_N]\!]$**:** The share creation algorithm takes as input the secret s and returns a set of N shares $[\![s_1, \ldots, s_N]\!]$.

$\mathsf{Reconstruct}([\![s_1, \ldots, s_t]\!]) \rightarrow s$**:** The secret reconstruction algorithm takes as input t shares of a secret s and returns the secret s itself.

Security of a Secret Sharing Scheme Given a fixed number of parties N, it should be infeasible for anybody in possession of any subset of less than t shares of a secret s, to reconstruct the secret s itself.

5.1.2 Distributed Minimum Computation

A secret sharing scheme as presented above can be leveraged to perform multiparty computations. In a multiparty computation protocol, each user i has an input x_i and a function to be computed $f()$. The goal of a multiparty computation protocol is to let a set of N users to compute $y := f(x_1, \ldots, x_N)$ without them learning anything about the input from any other user other than what is trivially revealed by y.

In SilentWhispers, we require a multiparty computation where the function f is defined as the minimum among all the input values. On input secret shares of values $x_1, \ldots, x_n$ shared using scheme $\mathcal{T}$ among a set of computing parties, we use a multi-party computation protocol $min()$ that results in each party having a share of the minimum of those values. We employ a distributed integer comparison protocol [86] for this distributed computation.

5.1.3 Notation for Protocol Description

We use the following termonology to describe our protocols.

$\mathsf{p}(\mathsf{u}, i)$	Parent of node u in the path_i
$\mathsf{c}(\mathsf{u}, i)$	Child of node u in the path_i
$v_{\mathsf{u}_1,\mathsf{u}_2}$	Credit value on link $\mathsf{u}_1 \rightarrow \mathsf{u}_2$
$\mathsf{st}_{\mathsf{u}_1,\mathsf{u}_2}$	Last value on $\mathsf{u}_1 \rightarrow \mathsf{u}_2$ agreed by $\mathsf{u}_1, \mathsf{u}_2$
$\mathsf{m}[i]$	Element at position i in array m
$\mathsf{vk}_{\mathsf{u}}^{i}$	Fresh verification key of user u in path_i
max	Maximum path length (system parameter)
ts	Current timestamp

5.2 Preliminaries

In this section, we describe how to adapt the Ripple Network to fit our description of credit network. Moreover, we review the setup and communication model. Here, we consider the threat model as described in Chapter 3 and aim to achieve the security and privacy goals described in Chapter 3.

5.2.1 Adapting the Ripple Network

In this chapter, we consider a credit network as described in Chapter 3. In particular, for ease of exposition, we consider a transformation of the credit network to denote how much IOU can be transferred between wallets instead of how much IOU one wallet owes to its counterparty, as described by Dandekar et al. [87]. For example, a credit link of the form $\mathsf{u}_1 \rightarrow \mathsf{u}_2$ with balance α and limit β, is now represented as two credit links: one credit link $\mathsf{u}_1 \rightarrow \mathsf{u}_2$ with weight $\beta - \alpha$, and a second link $\mathsf{u}_1 \leftarrow \mathsf{u}_2$ with weight α.

In this alternative way of representing the credit network, a payment operation works slightly different. A payment for a value v requires to reduce v in each credit link from the sender to the receiver and to increase v in each link from the receiver to the sender. One advantage of this representation is to calculate the credit available in a path: It simply consists on calculating the minimum weight in the credit links from the sender to the receiver.

5.2.2 Setup and Communication Model

Throughout this chapter, we assume that the set of landmarks is fixed at the beginning of each epoch and that it is known to all users. Any changes to the set become effective in the next epoch. This is crucial as this allows users to know the root of all Breadth-First Search trees in advance (and therefore the number of possible paths) during the routing operation, and to securely communicate with them. In practice, one can maintain the set of landmarks in a public and authenticated log (e.g., as Tor directory authorities listing).

We assume that the communication between two honest users is not observable by the attacker. This is a stronger requirement than the presence of a secure channel, since, in addition to hiding the messages exchanged by the two clients, we want to hide the fact that communication happened in the first place. If the adversary observes whether two honest users communicate, it is not possible to enforce any

meaningful notion of sender/receiver privacy. We note that, in practice, this condition can be enforced by having the two users deploying some anonymous communication channel (e.g., Tor [57]). Moreover, we require all of the involved users to be online during a given transaction for the execution of the algorithms. We discuss later in the extensions of SilentWhispers how to relax this condition.

5.3 Solution Overview

As a warm up, we propose a naïve solution to build a secure, privacy-preserving decentralized credit network and discuss its flows. We then overview our approach towards our constructions in SilentWhispers. We divide our exposition in the two main functionality blocks that compose a credit network: *Routing* and *graph management*.

5.3.1 Routing

A common, prominent challenge in a credit network is to determine credit routes between the senderand the receiver. Ghosh et al. [1] have shown that the problem of maximizing the possible transactions (which they term as social welfare) in a credit network is NP-hard. Existing credit networks instead consider one transaction at a time and employ the maximum flow approach [88] to check the available credit among all possible paths between sender and receiver. However, the most efficient known max-flow algorithms run in $O(V^3)$ [89] or $O(V^2 log(E))$ [90] time. For this reason, recent work explored the possibility to efficiently calculate only a subset of all possible paths between senderand receiver, thereby underestimating the available credit. The idea of this algorithm, called landmark routing [91], is to calculate a path between senderand receiverthrough an intermediary node called a *landmark*. As demonstrated by Viswanath et al. in the Canal credit network [92], landmark routing performs much better in large networks than the max-flow approach, with an accuracy loss of only 5%. Canal is split into two processes:

1. **Universe creator:** This process has access to the plain network graph along with all links' weights. It randomly selects a small subset of nodes denoted as landmarks. For every landmark, it calculates the shortest path from the landmark to every other node in the graph using a breadth-first search (BFS) algorithm, resulting in a BFS tree. The resulting set of BFS trees is stored in the so-called *landmark universe*.

2. **Path stitcher:** For a request to pay β credits from a sendernode to a receivernode, the path stitcher reads the landmark universe looking for paths with available credit between senderand receiver. When the process finds a set of paths with a total of at least β available credits, it carries out the transaction by decreasing the credit of the corresponding links and returning a successful result. If the process instead reaches the end of the landmark universe without success, the graph is kept unchanged and it returns an unsuccessful result.

Routing information must be repeatedly recalculated to account for the dynamic nature of credit networks: credit links among users are continuously updated, created, and deleted as a result of carrying out the transactions. Under the assumption that users are loosely synchronized, we divide the time in well-known epochs: BFS arborescences and anti-arborescences are created at the beginning of each epoch and users utilize that routing information throughout the duration of the epoch.

We assume that the set of landmarks is fixed and known to all users and that the credit network is a connected graph. Then, the correctness of BFS ascertains that each user receives routing information from all her neighbors for each landmark. This ensures that no honest user is alienated by a malicious neighbor; the absence of BFS related communication from a neighbor for any landmark serves as a detection mechanism of misbehavior so that further actions (e.g., removing the link with the misbehaving neighbor) can be adopted.

5.3.2 Graph Management

The central technical challenge in the design of a credit network is the computation of the credit available in a certain path, which is necessary for performing a transaction. A first, trivial solution would be to let every user in the path privately communicate her own link's value to the corresponding landmark so that the landmark can thereby compute the minimum value over the path and notify the intended recipients. It is easy to see, however, that this approach fails to guarantee privacy against an honest-but-curious landmark as the landmark would learn the credit associated with each link.

A local approach, where the credit on the path gets computed step-by-step by each user in the path, does not solve the privacy problem either. For instance, suppose that each user sends to the next user in the path the lower value between the one of its own link and the one received from the previous user: It is easy to see that such a protocol leaks all the intermediate values.

The idea underlying our approach is to design a secure Multi-Party Computation (SMPC) protocol to compute the credit available on a path. In order to boost the efficiency of our construction, we let landmarks play the role of *computation parties*, each receiving a share of the credit on each link from the sender to the receiver. Landmarks can jointly compute the credit on the whole path, intuitively by computing a series of minimum functions, but without learning anything about the result of the computation, nor of course the credit on the links.

An illustrative example is shown in Figure 5.2. First, every user in the payment path from the sender (Alice) to the receiver (Dave), creates a share of the link's value for each of the landmarks. After receiving all shares, landmarks locally compute the "minimum" function over the shares, thereby obtaining a share of the result that is then sent to the sender. Finally, the sender reconstructs the result and carries out the payment.

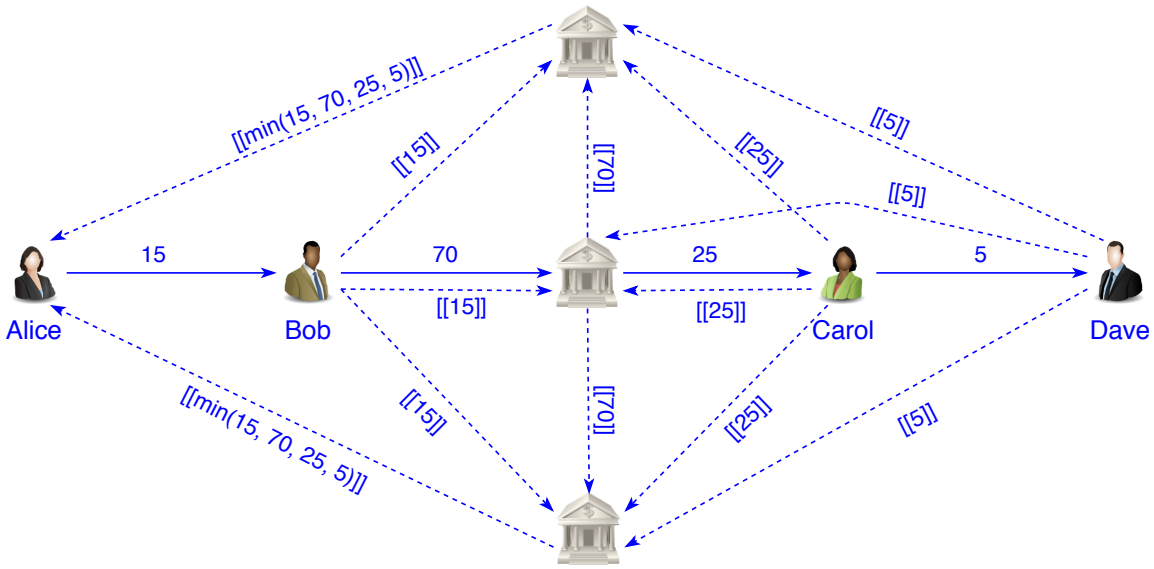


Fig. 5.2. An illustrative example of the use of SMPC in SilentWhispers: Dashed lines show communication between parties and solid arrows represent credit links, while notation $[a]$ indicates a (secret sharing) share of value a. We consider a payment from Alice to Dave. First, every user in the path sends a share of her link value to each landmark. Then, landmarks locally compute the share of the minimum credit on the path and send it to the sender. Transfer of the share from the landmark in the middle to the sender has been omitted for readability.

This approach, however, leaves two important concerns unanswered. First, how to assure that the shares come from users forming a path from the sender to the receiver without compromising their privacy (e.g., revealing the links); and second, how to enforce the correctness of the updates of links caused by the transaction without using a public ledger.

We ensure that all shares come from users in a path from the sender to the receiver by resorting to a chain of signatures. Naïvely, we could assume that every user uses a long-term key pair to sign the verification key from her predecessor and her successor in a given path. This would result in a unique signature chain serving as a valid proof of the existence of a path from sender to receiver.

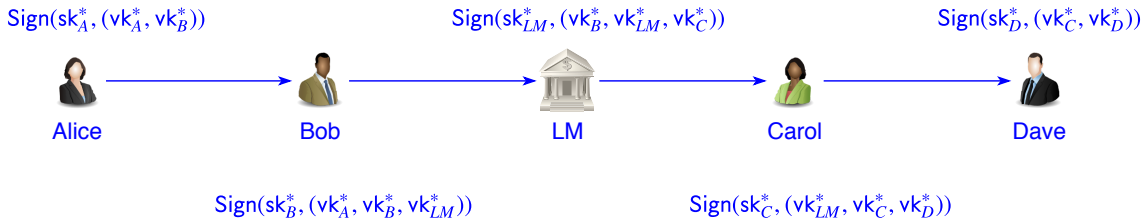


Fig. 5.3. Illustrative example of path construction in SilentWhispers. Every user i has a pair $(\mathsf{sk}_i^*, \mathsf{vk}_i^*)$ of signing and verification keys. Every user in the path privately exchanges the fresh verification key to both neighbors. Then, each user publishes a signed tuple containing the fresh verification keys of the neighbors and his/her own. A path is correct if contiguous verification keys in the path are equal.

However, the exposure of the same long term keys across different transactions would allow for correlation attacks and ultimately compromise user privacy. Using fresh keys per transaction to overcome this issue does not entirely solve the problem either: since fresh keys are not bound to a user, an adversary can always impersonate an honest user with her own keys.

Our idea, instead, is to combine long term and fresh keys. First, a user signs a fresh verification key with her long term signing key so that they are bound together. The (sensitive) long term verification key is revealed only to the counterparty in a credit link so that the relation between a fresh verification key and a user is verifiable to the counterparty but remains hidden for the rest of users in the credit network. Second, a user can use her fresh signing key to sign the fresh verification key of the predecessor and successor in any given path, thereby creating a signature chain. A pictorial description of the approach is reported in Figure 5.3.

5.4 Protocol Specifications

In this section, we first describe the SilentWhispers protocol and then discuss on possible extensions to cope with offline users or malicious landmarks among others aspects. Finally, we analyze the security, privacy and performance of our proposed protocol.

5.4.1 Protocol Description

In the following we describe the routines of SilentWhispers.

Routing Users have access to a synchronous network through $\mathcal{F}_{\text{NET}}$. Every pair of users sharing a credit link communicate through a secure and authenticated channel, described by $\mathcal{F}_{\text{SMT}}$. Secure realizations of $\mathcal{F}_{\text{NET}}$ and $\mathcal{F}_{\text{SMT}}$ have been proposed in [60]. Finally, users have access to the routing protocol described in $\mathcal{F}_{\text{ROUT}}$: this functionality is executed periodically at epochs (e.g., according to some system parameter) so that frequent changes in the inherently dynamic topology of credit networks are taken into account for subsequent transactions.

Setup Link The credit link updates are handled as defined in the *link setup* protocol (Algorithm 2). This protocol allows two users sharing a credit link to agree on the link's value at the beginning of each routing *epoch*. This is later used as a reference for subsequent updates within the epoch. For that, each user signs the other's long-term verification key and the current credit with her own long-term signing key.

Algorithm 2 Link setup protocol.

Input :

$\mathsf{u_1}$, $\mathsf{u_2}$:	Nodes creating a shared link
v:	Value of the link $\mathsf{u}_1 \rightarrow \mathsf{u}_2$
$(\mathsf{sk}^*_{\mathsf{u}_i}, \mathsf{vk}^*_{\mathsf{u}_i})$:	User i long term keys
epoch:	Current epoch

1: u_1 sends $\sigma_1 := \mathsf{Sign}(\mathsf{sk}^*_1, (\mathsf{settled}||\mathsf{vk}^*_1||\mathsf{vk}^*_2||v||epoch))$ to u_2
2: u_2 sends $\sigma_2 := \mathsf{Sign}(\mathsf{sk}^*_2, (\mathsf{settled}||\mathsf{vk}^*_1||\mathsf{vk}^*_2||v||epoch))$ to u_1
3: if $\mathsf{Verify}(\mathsf{vk}^*_2, (\mathsf{settled}||\mathsf{vk}^*_1||\mathsf{vk}^*_2||v||epoch), \sigma_2)$ then u_1 stores $(\sigma_1, \sigma_2, \mathsf{st}_{\mathsf{vk}^*_1,\mathsf{vk}^*_2} := (\mathsf{settled}||\mathsf{vk}^*_1||\mathsf{vk}^*_2||v||epoch))$
4: if $\mathsf{Verify}(\mathsf{vk}^*_1, (\mathsf{settled}||\mathsf{vk}^*_1||\mathsf{vk}^*_2||v||epoch), \sigma_1)$ then u_2 stores $(\sigma_1, \sigma_2, \mathsf{st}_{\mathsf{vk}^*_1,\mathsf{vk}^*_2} := (\mathsf{settled}||\mathsf{vk}^*_1||\mathsf{vk}^*_2||v||epoch))$

Payment For easing the presentation, we have made two simplifications. First, we assume the set of paths $\{\mathsf{path}_1, \ldots, \mathsf{path}_{\mathsf{LM}}\}$ as input of the transaction protocol, although in reality every user notifies her parent on the path that she is part of a transaction path and she needs to carry out the corresponding operations. Second, at certain steps of the protocol we write that users submit messages directly to the corresponding landmark (e.g., line 10) to mean that such messages are sent to the landmark by forwarding it among neighbors in the path. The creator of such message encrypts it under the public key of the landmark and signs it with her fresh signing key to avoid tampering from other users.

Phase 1: path construction and shares submission (lines 1-15): In this phase, users on each path create a signature chain and submit the shares of their link values to the landmarks. In detail, starting from the sender, each user signs her fresh verification key with her long term signing key and sends the signature to both the successor and the predecessor in the path (lines 3-4). This signature binds a fresh verification key to a user and thus avoids illegitimate impersonations. Neighbors can then exchange the shares of their shared link's value and check that they reconstruct to the same value

(i.e., the two end-points agree on the credit between them) (lines 5-6). Finally, each user on the path signs all this information along with a timestamp (to avoid replay attacks) and sends it to the landmarks (lines 8-10). The signature is created with the user's fresh signing key so that the user's identity is concealed from the landmarks. Finally, the sender must create additional messages for each path in order to pad it into a length predefined by the system (i.e., max) in order to avoid inference attacks based on the path length (line 14). The same procedure is symmetrically carried out on the paths from the receiver to each landmark.

Concerning the integrity of paths, we observe that a malicious user could divert the signature chain using fresh keys of her choice. However, she cannot get an honest user into the fake chain continuation, since that user would refuse to sign the attacker's fresh key, making the attack ineffective.

Phase 2: computation of credit on a path (lines 16-24): In this phase, landmarks verify the correctness of the signature chain and calculate the credit available in each path. In particular, after the landmarks receive messages from up to max users for each path, they verify that neighboring keys in a path are consistent and calculate the minimum value of each path using a secure multi-party computation (lines 17-24). This results into each landmark having a share of the minimum value for each path which is then sent to the sender.

In a nutshell, the use of fresh keys hides users identities and the multiparty computation over shared values does not reveal the actual link values to the landmarks. Additionally, due to the use of fresh keys for each path, landmarks cannot detect whether a given link is shared in more than one path. This could result in landmarks calculating a path value greater than the available one. Nevertheless, this over-approximation is detected in the next phase when a link cannot be updated due to insufficient credit and this path is then ignored for the transaction without incurring any credit loss for the users involved in the transaction.

Algorithm 3 Transaction protocol.

	$\mathsf{Sdr}, \mathsf{Rcv}$:	Transaction sender and receiver
Input:	$\{\mathsf{path_1}, \ldots, \mathsf{path_{LM}}\}$	Set of paths Sdr to Rcv
	$(\mathsf{sk}^*_{\mathsf{u}_i}, \mathsf{vk}^*_{\mathsf{u}_i})$:	user u_i long term keys

1: {Phase 1: signature chain }
2: **for** $i \in |\mathsf{LM}|$ **do**
3: **for** $\mathsf{u} \in \mathsf{path}_i$ **do**
4: u creates fresh keys $(\mathsf{sk_u}, \mathsf{vk_u})$, $\sigma_\mathsf{u} := \mathsf{Sign}(\mathsf{sk}^*_\mathsf{u}, \mathsf{vk_u})$ and sends $(\sigma_\mathsf{u}, \mathsf{vk_u})$ to $\mathsf{p}(\mathsf{u}, i)$ and to $\mathsf{c}(\mathsf{u}, i)$
5: u receives $(\sigma_{\mathsf{c}(\mathsf{u},i)}, \mathsf{vk}_{\mathsf{c}(\mathsf{u},i)})$ from $\mathsf{c}(\mathsf{u}, i)$ and $(\sigma_{\mathsf{p}(\mathsf{u},i)}, \mathsf{vk}_{\mathsf{p}(\mathsf{u},i)})$ from $\mathsf{p}(\mathsf{u}, i)$
6: u receives from $\mathsf{c}(\mathsf{u}, i)$ shares $[\![s'_1, \ldots, s'_{|\mathsf{LM}|}]\!]$, u reconstructs v' from $[\![s'_1, \ldots, s'_{|\mathsf{LM}|}]\!]$ and checks whether $v' = v_{\mathsf{c}(\mathsf{u},i),\mathsf{u}}$
7: u creates $[\![s_1, \ldots, s_{|\mathsf{LM}|}]\!]$ for the value $v_{\mathsf{u},\mathsf{p}(\mathsf{u},i)}$ and sends them to $\mathsf{p}(\mathsf{u}, i)$
8: **if** $\mathsf{Verify}(\mathsf{vk}^*_{\mathsf{c}(\mathsf{u},i)}, \mathsf{vk}_{\mathsf{c}(\mathsf{u},i)}, \sigma_{\mathsf{c}(\mathsf{u},i)}) \wedge \mathsf{Verify}(\mathsf{vk}^*_{\mathsf{p}(\mathsf{u},i)}, \mathsf{vk}_{\mathsf{p}(\mathsf{u},i)}, \sigma_{\mathsf{p}(\mathsf{u},i)})$ **then**
9: **for** $j \in |\mathsf{LM}|$ **do**
10: u creates $\mathsf{m} := (\mathsf{vk}_{\mathsf{c}(\mathsf{u},i)}||[\![s'_j]\!]||\mathsf{vk_u}||\mathsf{vk}_{\mathsf{p}(\mathsf{u},i)}||[\![s_j]\!]||\mathsf{Txid}||ts)$, u creates $\sigma_{\mathsf{LM}_j} := \mathsf{Sign}(\mathsf{sk_u}, \mathsf{m})$ and finally sends $(\sigma_{\mathsf{LM}_j}, \mathsf{m})$ to LM_j
11: **end for**
12: **end if**
13: **end for**
14: Sdr creates $k := (\mathsf{max} - |\mathsf{path}_i|)$ more tuples $(\mathsf{m}, \sigma_{\mathsf{LM}_i})$, where all shares reconstruct to the maximum possible credit in a link, and sends them to LM_i
15: **end for**
16: {Phase 2: Minimum computation}
17: **for** $i \in |\mathsf{LM}|$ **do**
18: Each LM checks whether $|\mathsf{path}_i| = \mathsf{max} \wedge$
19: **for all** $j \in \{1, \ldots, |\mathsf{path}_i|\}$ **do**
20: $\mathsf{Verify}(\mathsf{m}_j[3], \mathsf{m}_j, \sigma_j) \wedge \mathsf{m}_j[1] = \mathsf{m}_{j-1}[3] \wedge \mathsf{m}_j[4] = \mathsf{m}_{j+1}[3] \wedge \mathsf{m}_{j-1}[6] = \mathsf{m}_j[6] = \mathsf{m}_{j+1}[6]$
21: **end for**
22: Each LM computes the share s_{min_i} as result for function $min(\cdot)$ over the shares $[\![s_1, \ldots, s_{\mathsf{max}}]\!]$ belonging to path_i.
23: Each LM sends the resulting tuples $(i, s_{min_i}, \mathsf{vk}^i_1, \mathsf{vk}^i_{\mathsf{max}})$ to Sdr
24: **end for**

Algorithm 4 Transaction protocol (continued)

1: {Phase 3: Carrying out transaction}
2: Sdr reconstructs the tuples (i, min_i) and verifies that vk_1^i and $\mathsf{vk}_{\mathsf{max}}^i$ are the first and last keys of path_i as she expects
3: **for** $i \in |\mathsf{LM}|$ **do**
4: Sdr chooses the transaction value x_i, generates $\mathsf{tx}_i := (ts||x_i||\mathsf{Txid}||\mathsf{vk}_{\mathsf{Sdr}}||\mathsf{vk}_{\mathsf{Rcv}})$ and $\sigma_{\mathsf{Sdr}}^i := \mathsf{Sign}(\mathsf{sk}_{\mathsf{Sdr}}, \mathsf{tx}_i)$, and sends $(\mathsf{tx}_i, \sigma_{\mathsf{Sdr}}^i)$ to the nodes in path_i
5: **for** $\mathsf{u} \in \mathsf{path}_i$ **do**
6: u checks $\mathsf{Verify}(\mathsf{vk}_{\mathsf{Sdr}}, \mathsf{tx}_i, \sigma_{i_{\mathsf{Sdr}}})$, x_i is smaller than the value $v_{\mathsf{u},\mathsf{p}(\mathsf{u},i)}$, and previous link $\mathsf{c}(\mathsf{u}, i) \rightarrow \mathsf{u}$ has been reduced by x_i u decreases link value on path_i by x_i resulting in x_i'
7: u creates $\mathsf{m} := (\mathsf{on\ hold}||\mathsf{vk}_{\mathsf{u}}^*||\mathsf{vk}_{\mathsf{p}(\mathsf{u},i)}^*||x_i'||\mathsf{tx}_i)$, $\sigma_{\mathsf{u}} := \mathsf{Sign}(\mathsf{sk}_{\mathsf{u}}^*, \mathsf{m})$ and sends $(\sigma_{\mathsf{u}}, \mathsf{m})$ to $\mathsf{p}(\mathsf{u}, i)$
8: u receives $\sigma_{\mathsf{p}(\mathsf{u},i)} := \mathsf{Sign}(\mathsf{sk}_{\mathsf{p}(\mathsf{u},i)}^*, \mathsf{m})$ from $\mathsf{p}(\mathsf{u}, i)$
9: u and $\mathsf{p}(\mathsf{u}, i)$ locally store $(\mathsf{st}_{\mathsf{vk}_{\mathsf{u}}^*, \mathsf{vk}_{\mathsf{p}(\mathsf{u},i)}^*} := \mathsf{m})$ and $(\sigma_{\mathsf{p}(\mathsf{u},i)}, \sigma_{\mathsf{u}})$
10: **end for**
11: Rcv $\sigma_{\mathsf{Rcv}}^i := \mathsf{Sign}(\mathsf{sk}_{\mathsf{Rcv}}, \mathsf{tx}_i)$ and sends $(\mathsf{tx}_i, \sigma_{\mathsf{Rcv}}^i)$ to Sdr
12: **end for**
13: **for** $i \in |\mathsf{LM}|$ **do**
14: Rcv sends $(\mathsf{tx}_i, \sigma_{\mathsf{Sdr}}^i, \sigma_{\mathsf{Rcv}}^i)$ to every node in path_i
15: **for** $\mathsf{u} \in \mathsf{path}_i$ **do**
16: u creates $\mathsf{m} := (\mathsf{settled}||\mathsf{vk}_{\mathsf{u}}^*||\mathsf{vk}_{\mathsf{c}(\mathsf{u},i)}^*||x_i'||ts)$, $\sigma_{\mathsf{u}} := \mathsf{Sign}(\mathsf{sk}_{\mathsf{u}}^*, \mathsf{m})$ and sends $(\sigma_{\mathsf{u}}, \mathsf{m})$ to $\mathsf{c}(\mathsf{u}, i)$
17: u receives $\sigma_{\mathsf{c}(\mathsf{u},i)} := \mathsf{Sign}(\mathsf{sk}_{\mathsf{c}(\mathsf{u},i)}^*, \mathsf{m})$ from $\mathsf{c}(\mathsf{u}, i)$
18: u and $\mathsf{c}(\mathsf{u}, i)$ locally store $(\mathsf{st}_{\mathsf{vk}_{\mathsf{u}}^*, \mathsf{vk}_{\mathsf{c}(\mathsf{u},i)}^*} := \mathsf{m})$ and $(\sigma_{\mathsf{c}(\mathsf{u},i)}, \sigma_{\mathsf{u}})$
19: **end for**
20: **end for**

Phase 3: Updating link values (lines 1-20, continuation): Link values on each path are updated so that the expected credit reaches the receiver. This process is performed in two steps. First, the transaction value for each path is decreased (i.e., *on hold*) on each link from the sender to the receiver (lines 3-10, continuation). This ensures that a user puts on hold credit on her outgoing link only after assuring the credit in the incoming link has been held, and thus a honest user in the path cannot incur in credit loss. This escrow serves as a commitment to accept the new link value if the receiver eventually accepts the transaction.

Second, after receiving the confirmation from the receiver (i.e., the receiver signature on the transaction's value for a given path), the held value is adopted as the new credit value (i.e., *settled*) on each link, starting from the receiver to the sender (lines 13-20, continuation). This reverse order ensures that each user in the path has an incentive to settle the final value: a user first settles the outgoing link (i.e., giving out credit), and thus is in the user interest to settle the incoming link (i.e., receiving credit) to recover the credit. In this manner, credit values on transaction paths can be consistently updated. Interestingly, if any user does not cooperate with her neighbor during this phase (e.g., a faulty user), the credit involved in the dispute is bounded and eventually would be resolved by either continuing the execution or aborting the payment in the complete path.

Test Credit The test operation works similar to the transaction protocol. It only differs in the fact that the sender will not carry out the transaction, as the test operation only requires the sender to learn the available credit.

Test Link and Change Link The testLink and chgLink can be easily performed by exchanging a message between the two end-points of the credit link through their authenticated private channel.

5.4.2 Security and Privacy Analysis

We hereby state the security and privacy results for SilentWhispers. We prove our result in the $\mathcal{F}_{\text{NET}}$, $\mathcal{F}_{\text{SMT}}$-hybrid model; i.e., the theorem holds for any UC-secure realization of $\mathcal{F}_{\text{NET}}$ and $\mathcal{F}_{\text{SMT}}$.

Theorem 5.4.1 (UC-Security) *Let $\mathcal{T}$ be a secure secret sharing scheme and Π be an existentially unforgeable digital signature scheme, then SilentWhispers UC-realizes the ideal functionality $\mathcal{F}_{CN}$ in the $\mathcal{F}_{NET}$, $\mathcal{F}_{SMT}$-hybrid model.*

Proof [Sketch] The proof proceeds by describing the simulator $\mathcal{S}$ that interacts with the ideal functionality $\mathcal{F}_{\text{CN}}$ in the ideal world and must provide to a corrupted environment $\mathcal{E}$ inputs that are computationally indistinguishable to the ones that $\mathcal{A}$ would output in the real-wold protocol. Thus the core of the proof is the simulation of the inputs that $\mathcal{A}$ is expecting from the protocol while interacting in the ideal world. We identify two main technical highlights in the proof: (i) any honestly computed signature in the protocol is simulated via the simulator offered by the security definition of digital signatures and (ii) shares of unknown secrets are simulated with values sampled uniformly at random from the appropriate domain. In the former case the indistinguishability from an honest execution is provided directly by the security definition of the signature scheme, while in the latter case it follows by the information theoretic hiding of the shared secret (throughout the experiment we always simulate a number of shares below the threshold value). The rest of the simulation focuses on adapting the adversarial behavior to the ideal functionality, aborting when appropriate. ■

The full proof is elaborated in the full version of the paper [59].

5.4.3 Performance Analysis

We have developed a C++ implementation to demonstrate the feasibility of SilentWhispers. We focus in particular on the payment protocol (Protocol 3), which dominates by far the computational complexity, simulating the main functionality of both landmarks and users in the credit network. Our realization relies on the MPC Shared Library [93], on the Shamir's information theoretic construction [85] for secret sharing, and on Schnorr's signatures [94, 95] due to their efficiency.

Implementation-Level Optimizations There exist several independent operations in a transaction that can be parallelized. Intuitively, in the first phase, users can prepare fresh keys, signatures and shares of the link values for each path in parallel.

They can then be processed and verified by landmarks in parallel as well during the second phase of the transaction protocol. Finally, users can carry out the third phase by updating links for different paths independently of each other.

Since the *min* function is associative, we can parallelize independent min operations to improve the efficiency of calculating the minimum value in a path. For instance, $x := min(a, b)$ and $y := min(c, d)$ can be done in parallel and then compute $min(x, y)$ to obtain the minimum among a, b, c, d. Finally, the sender can reconstruct the min_i values for each path_i and transmit it to the users in path_i in parallel.

Performance We conduct our experiments in machines with 3.3 GHz processor and 8 GB RAM to carry out decentralized operations involving landmarks (e.g., multiparty computation of the minimum value of a path). We simulate each landmark in a different machine. For our experiments, we have implemented the cryptographic schemes used in the transaction protocol. Based on their execution time, we calculated the total time for the transaction operations taking into account the implementation optimizations.

Transaction Time The chgLink and testLink operations are performed directly between the users sharing a credit link and are extremely efficient. Among the other transactions, we have studied the pay transaction, since it is clearly more expensive than test. In particular, we first study the communication cost and then the computation time required for the pay operation.

In the pay operation, each user in the path must forward messages to the neighbors. The longest message to be sent as defined in Algorithm 3 contains 340 bytes: 4 verifications keys (i.e., 64 bytes each in the elliptic curve setting), 5 integers of 4 bytes each and a signature (i.e., extra 64 bytes). In the worst case, a user must forward one such messages for each of the max neighbors and thus the communication cost is $\max \cdot$ 340 bytes. In practice, max is a small constant and forwarding such message can be done efficiently even with commodity communication links.

Table 5.1.
Times in seconds to compute $\text{Min}(a, b)$. We use 32 bits to represent a and b. In a setup (n, t), n denotes the total number of landmarks out of which t are compromised.

Setup	$(5,1)$	$(5,2)$	$(7,1)$	$(7,2)$	$(7,3)$
Time	0.304	0.314	0.357	0.346	0.349

Regarding computation time, we observe that operations performed by users in phases 1 and 3 consist of the creation and verification of signatures, which are extremely efficient. Therefore, we focus on the computation of the credit value of a path (i.e., the minimum among the credit values of the links composing the path), since it is the most expensive operation.

The time to compute the minimum between two values among a set of landmarks is shown in Table 5.1. The actual number of such min computations required to calculate the credit in a path depends on the length of the path (i.e., max). Using the implementation level optimizations, landmarks need to perform only $\lceil \log(\text{max}) \rceil$ min operations sequentially. In Table 5.2 we show the time to compute the credit in a path for different path lengths. In particular, computing the minimum credit in a path takes roughly 1.7 seconds for max = 20.

Routing Time We consider the remaining protocols in SilentWhispers: the link setup and the routing protocol. The link setup is extremely efficient and can be done even offline. The routing protocol requires a decentralized BFS algorithm. The decentralized BFS is well studied in the literature and it has been shown to be practical [96]. In particular, the proposed algorithm has a communication complexity of $O(E)$ and a time complexity of $O(l^2)$, where E denotes the number of links and l denotes the height of the BFS tree. Moreover, BFS does not involve cryptographic

Table 5.2.
Times in seconds to compute the credit on a path. We use a setup (7, 3): 7 landmarks, 3 compromised.

Path Length (max)	5	10	20
Time	1.047	1.349	1.745

operations and it can be run as a background process, thus it does not hinder the performance of the rest of system operations.

Establishing System Parameters Running SilentWhispers requires setting up two system parameters: the maximum path length and the number of landmarks. To do that, we have extracted transactions carried out in Ripple [37]. Based on this information, we set up the system parameters such that SilentWhispers can process the transactions already performed in Ripple.

First, for processing a transaction, the sender has to pad the number of links in the path to maintain the privacy properties. In order to find a meaningful value for the maximum path length, we have collected all transactions from the start of the Ripple network until December 2015, resulting in a set of 17,645,343 transactions. The maximum path length that we have observed is 10 links. Thus, we set up the maximum path length to 10 in our evaluation.

Second, processing a transaction requires more than one path. The actual number of paths used in a transaction will determine the number of landmarks required in our system. In order to find this value, we have extracted the distribution on the number of paths that have been used for the Ripple transactions. We have observed that the maximum number of paths used in a transaction is 7 and thus we use 7 landmarks in our evaluation. We note that using the landmark routing algorithm in the current Ripple network might imply a variation in the number of required landmarks. However, choosing adequate users as landmarks will ensure that the maximum number of paths

is maintained within a small factor, as most of the transactions are routed through the landmarks.

In practice, selecting those users with higher number of credit links as the landmarks facilitates finding suitable transaction paths between users for a transaction. For instance, banks are the natural candidate to serve as landmarks in a transaction network. Furthermore, we have extracted the Ripple network and observed that most nodes have links to a few highly connected nodes, which correspond to gateways. They are already well known to all users as most of them also contribute to validate the Ripple network, and they thus become the ideal landmark candidates when applying SilentWhispers in Ripple.

In conclusion, SilentWhispers can simulate the Ripple network using 7 landmarks and a path length of 10. Given these system parameters, each user has to forward, in the worst case, a message of $10 \cdot 340 = 3400$ bytes, which can be done efficiently even with commodity communication links. Moreover, computing the minimum credit in a path takes roughly 1.3 seconds (see Table 5.2). A transaction in the currently deployed transaction network Ripple, takes approximately 5 seconds. Thus, our evaluation shows that SilentWhispers does not introduce any significant overhead to the transaction time.

6 SUMMARY

In this dissertation we thoroughly study the security and privacy in credit networks and we expect that it motivates the use of secure and privacy-preserving transaction protocols in the current and emerging systems based on credit networks. In particular, this dissertation describes the following contributions.

Security and Privacy Study of the Ripple Network As the most representative instance of credit network in practice, we have thoroughly studied the Ripple network to characterize its security and privacy issues. As a result, we describe the effect of unexpected redistribution of credit, the effect of faulty gateways and the effect of stale offers on the credit held by users. Our results show that the Ripple community must be educated about these issues to prevent them from credit losses. Moreover, we shed light on the gap –due to certain patterns of use and interaction between parties in the network— between the (supposedly) provided privacy available in the Ripple network and the actual privacy achieved by the current Ripple users and, most importantly, their transactions. Our analysis thus motivates the imperative need for better privacy-preserving transactions mechanisms for Ripple and any other emerging transaction network based on the same design principles.

Security and Privacy Definitions for Credit Networks The lack of formal definitions of security and privacy notions of interest in a credit network hinders the design of systems aiming to provide secure and privacy-preserving transactions. In this state of affairs, we provide the first formalization for the notions of integrity, transaction value privacy and transaction sender/receiver privacy. These definitions

serve as the basis to formally assess the security and privacy guarantees achieved by new proposals of credit network systems.

Anonymous Transactions in Ripple Currently deployed credit networks such as Ripple rely on a public available ledger that inherently leaks sensitive financial information such as credit links and transactions. Given that, we require a solution that provides anonymous transactions fully compatible with the current Ripple network. In order to achieve that, we have built PathShuffle, our novel protocol to perform atomic and anonymous transactions in the Ripple network. The atomicity provided by PathShuffle is of special interest not only for path mixing protocols, but also for other applications such as crowdfunding.

Secure and Privacy-Preserving Decentralized Credit Networks Tailored privacy-enhancing protocols such as PathShuffle help to raise the bar, but do not provide a generic solution with strong privacy guarantees. In this dissertation we have presented SilentWhispers, a novel decentralized architecture for credit networks that achieves strong security and privacy properties. The distinguishing feature of our architecture is the avoidance of a global, publicly available ledger and still provides the functionality required by users in a credit network.

REFERENCES

REFERENCES

[1] A. Ghosh, M. Mahdian, D. M. Reeves, D. M. Pennock, and R. Fugger, "Mechanism design on trust networks," in *Internet and Network Economics*, ser. WINE, 2007, pp. 257–268.

[2] R. Fugger, "Money as IOUs in Social Trust Networks and A Proposal for a Decentralized Currency Network Protocol," 2004, http://library.uniteddiversity.coop/Money_and_Economics/decentralizedcurrency.pdf (Accessed May 2018).

[3] D. do B. DeFigueiredo and E. T. Barr, "TrustDavis: A non-exploitable online reputation system," in *Conference on E-Commerce Technology*, ser. CEC, 2005, pp. 274–283.

[4] A. Post, V. Shah, and A. Mislove, "Bazaar: Strengthening user reputations in online marketplaces," in *Symposium on Networked Systems Design and Implementation*, ser. NSDI, 2011, pp. 183–196.

[5] A. Mislove, A. Post, P. Druschel, and K. P. Gummadi, "Ostra: Leveraging trust to thwart unwanted communication," in *Symposium on Networked Systems Design and Implementation*, ser. NSDI, 2008, pp. 15–30.

[6] "Ripple website," https://ripple.com/ (Accessed May 2018).

[7] "Stellar website," https://www.stellar.org/ (Accessed May 2018).

[8] A. Liu, "Ripple Labs signs first two US banks," Ripple blog, 2014, https://ripple.com/blog/ripple-labs-signs-first-two-us-banks/ (Accessed May 2018).

[9] T. Yablonskaya, "Royal Bank of Canada teams up with Ripple for blockchain remittance system," Coinspeaker blog, 2016, http://www.coinspeaker.com/2016/02/25/royal-bank-of-canada-teams-up-with-ripple-for-blockchain-remittance-system/ (Accessed May 2018).

[10] A. Liu, "Santander: Distributed ledger tech could save banks $20 billion a year," Ripple blog, 2015, https://ripple.com/blog/santander-distributed-ledger-tech-could-save-banks-20-billion-a-year/ (Accessed May 2018).

[11] A. Goel, "Bank-wise analysis of blockchain activity," Let's Talk Payments blog, 2015, http://letstalkpayments.com/bank-wise-analysis-of-blockchain-activity (Accessed May 2018).

[12] P. Rizzo, "Japan's SBI holdings teams with Ripple to launch new company," CoinDesk blog, 2016, http://www.coindesk.com/sbi-holdings-ripple-new-company/ (Accessed May 2018).

[13] J. Southurst, "Australia's Commonwealth Bank latest to experiment with Ripple," CoinDesk blog, 2015, http://www.coindesk.com/australia-commonwealth-bank-ripple-experiment/ (Accessed May 2018).

[14] M. Long, "Santander becomes the first U.K. bank to use Ripple for cross-border payments," Ripple blog, 2016, https://ripple.com/insights/santander-becomes-first-uk-bank-use-ripple-cross-border-payments/ (Accessed May 2018).

[15] P. (Pseudonym), "How EarthPort and Ripple are teaming up to make cross-border payments instant," PYMNTS.com blog, 2015, http://www.pymnts.com/in-depth/2015/how-earthport-and-ripple-are-teaming-up-to-make-cross-border-payments-instant/ (Accessed May 2018).

[16] F. F. (Pseudonym), "Earthport launches distributed ledger hub via Ripple," Banking Technology blog, 2016, http://www.bankingtech.com/420912/earthport-launches-distributed-ledger-hub-via-ripple/ (Accessed May 2018).

[17] C. Hunt, "How Marco Montes is empowering migrant workers," Ripple blog, 2015, https://ripple.com/blog/how-marco-montes-is-empowering-migrant-workers/ (Accessed May 2018).

[18] P. Rizzo, "Microsoft explores adding Ripple tech to blockchain toolkit," CoinDesk blog, 2015, http://www.coindesk.com/microsoft-hints-future-ripple-blockchain-toolkit/ (Accessed May 2018).

[19] P. Moreno-Sanchez, M. B. Zafar, and A. Kate, "Listening to whispers of Ripple: Linking wallets and deanonymizing transactions in the Ripple Network," in *Privacy Enhancing Technologies Symposium*, ser. PETS, 2016, pp. 436–453.

[20] A. Narayanan and V. Shmatikov, "Robust de-anonymization of large sparse datasets," in *Symposium on Security and Privacy*, ser. S&P, 2008, pp. 111–125.

[21] K. Sharad and G. Danezis, "An automated social graph de-anonymization technique," in *Workshop on Privacy in the Electronic Society*, ser. WPES, 2014, pp. 47–58.

[22] A. Sala, X. Zhao, C. Wilson, H. Zheng, and B. Y. Zhao, "Sharing graphs using differentially private graph models," in *Internet Measurement Conference*, ser. IMC, 2011, pp. 81–98.

[23] P. Mittal, C. Papamanthou, and D. X. Song, "Preserving link privacy in social network based systems," in *Network and Distributed System Security*, ser. NDSS, 2013.

[24] N. Tran, B. Min, J. Li, and L. Subramanian, "Sybil-resilient online content voting," in *Symposium on Networked Systems Design and Implementation*, ser. NSDI, 2009, pp. 15–28.

[25] A. Mohaisen, H. Tran, A. Chandra, and Y. Kim, "Trustworthy distributed computing on social networks," in *Proceedings of the 8th ACM SIGSAC Symposium on Information, Computer and Communications Security*, 2013, pp. 155–160.

[26] A. Molavi Kakhki, C. Kliman-Silver, and A. Mislove, "Iolaus: Securing online content rating systems," in *Proceedings of the 22Nd International Conference on World Wide Web*, ser. WWW '13. New York, NY, USA: ACM, 2013, pp. 919–930. [Online]. Available: http://doi.acm.org/10.1145/2488388.2488468

[27] A. Mohaisen, N. Hopper, and Y. Kim, "Keep your friends close: Incorporating trust into social network-based Sybil defenses," in *INFOCOM, 2011 Proceedings IEEE*, 2011, pp. 1943–1951.

[28] T. Bergin and J. Finkle, "Exclusive: SWIFT confirms new cyber thefts, hacking tactics," Reuters press, 2016, http://www.reuters.com/article/us-usa-cyber-swift-exclusive/exclusive-swift-confirms-new-cyber-thefts-hacking-tactics-idUSKBN1412NT (Accessed May 2018).

[29] M. J. Schwartz, "Another SWIFT hack stole $12 million," Blog entry, 2016, https://www.bankinfosecurity.com/another-swift-hack-stole-12-million-a-9121 (Accessed May 2018).

[30] "Earthport blockchain offering secures fintech finance award," Blog entry, 2017, https://www.earthport.com/news-insights/news/earthport-blockchain-offering-secures-fintech-finance-award (Accessed May 2018).

[31] D. Patterson, "MIT running a Ripple validator," Ripple blog, 2016, https://ripple.com/insights/mitvalidator/ (Accessed May 2018).

[32] S. Marquer, "XRP ledger decentralizes further with expansion to 55 validator nodes," Ripple blog, 2017, https://ripple.com/insights/xrp-ledger-decentralizes-expansion-55-validator-nodes/ (Accessed May 2018).

[33] "Ripple transactions overview," https://ripple.com/build/transactions/ (Accessed May 2018).

[34] "Understanding the NoRipple flag," https://ripple.com/build/understanding-the-noripple-flag/ (Accessed May 2018).

[35] M. Castro and B. Liskov, "Practical byzantine fault tolerance," in *Symposium on Operating Systems Design and Implementation*, ser. OSDI, 1999, pp. 173–186.

[36] D. Schwartz, N. Youngs, and A. Britto, "The Ripple protocol consensus algorithm," Whitepaper, 2014, https://ripple.com/files/ripple_consensus_whitepaper.pdf (Accessed May 2018).

[37] F. Armknecht, G. Karame, A. Mandal, F. Youssef, and E. Zenner, "Ripple: Overview and outlook," in *Trust and Trustworthy Computing*, ser. TRUST, 2015, pp. 163–180.

[38] A. D. Luzio, A. Mei, and J. Stefa, "Consensus robustness and transaction de-anonymization in the Ripple currency exchange system," in *Distributed Computing Systems*, ser. ICDCS, 2017, pp. 140–150.

[39] C. Cachin and M. Vukolic, "Blockchain consensus protocols in the wild," *CoRR*, vol. abs/1707.01873, 2017.

[40] B. Chase and E. MacBrough, "Analysis of the XRP ledger consensus protocol," *CoRR*, vol. abs/1802.07242, 2018, http://arxiv.org/abs/1802.07242.

[41] P. Moreno-Sanchez, N. Modi, R. Songhela, A. Kate, and S. Fahmy, "Mind your credit: Assessing the health of the Ripple credit network," in *World Wide Web Conference*, ser. WWW, 2018, pp. 329–338.

[42] "Technical report on Ripple Flag," Whitepaper, 2015, https://ripple.com/files/GB-2015-04.pdf (Accessed May 2018).

[43] J. Young, "Ripple directs Bitstamp to freeze funds of former co-founder Jed McCaleb," Cointelegraph blog, 2015, https://cointelegraph.com/news/ripple-directs-bitstamp-to-freeze-funds-of-former-co-founder-jed-mccaleb (Accessed May 2018).

[44] B. R. (pseudonym), "The Ripple story," BitMex blog, 2018, https://blog.bitmex.com/the-ripple-story/ (Accessed May 2018).

[45] A. (pseudonym), "Ripple explodes monday morning setting all-time best market cap - glitch or gain?" XRPChat forum, 2016, https://www.xrpchat.com/topic/2187-ripple-explodes-monday-morning-setting-all-time-best-market-cap-glitch-or-gain (Accessed May 2018).

[46] S. Barber, X. Boyen, E. Shi, and E. Uzun, "Bitter to Better — how to make Bitcoin a better currency," in *Financial Cryptography and Data Security*, ser. FC, 2012, pp. 399–414.

[47] S. Meiklejohn, M. Pomarole, G. Jordan, K. Levchenko, D. McCoy, G. M. Voelker, and S. Savage, "A fistful of bitcoins: characterizing payments among men with no names," in *Internet Measurement Conference*, ser. IMC, 2013, pp. 127–140.

[48] M. Spagnuolo, F. Maggi, and S. Zanero, "Bitiodine: Extracting intelligence from the Bitcoin network," in *Financial Cryptography and Data Security*, ser. FC, 2014, pp. 457–468.

[49] F. Reid and M. Harrigan, "An analysis of anonymity in the Bitcoin system," in *Privacy, Security, Risk and Trust*, ser. PASSAT, 2011, pp. 1318–1326.

[50] D. Ron and A. Shamir, "Quantitative analysis of the full Bitcoin transaction graph," in *Financial Cryptography and Data Security*, ser. FC, 2013, pp. 6–24.

[51] P. Koshy, D. Koshy, and P. McDaniel, "An analysis of anonymity in Bitcoin using P2P network traffic," in *Financial Cryptography and Data Security*, ser. FC, 2014, pp. 469–485.

[52] A. Biryukov, D. Khovratovich, and I. Pustogarov, "Deanonymisation of clients in Bitcoin P2P network," in *Computer and Communications Security*, ser. CCS, 2014, pp. 15–29.

[53] "Becoming a Ripple gateway: Hot and cold wallets," Ripple blog, https://ripple.com/build/gateway-guide/#hot-and-cold-wallets (Accessed May 2018).

[54] "Issuing and operational addresses," Ripple blog, https://ripple.com/build/issuing-operational-addresses/ (Accessed May 2018).

[55] S. Kullback and R. A. Leibler, "On information and sufficiency," *The Annals of Mathematical Statistics*, vol. 22, no. 1, pp. 79–86, 1951. [Online]. Available: http://dx.doi.org/10.2307/2236703

[56] D. Kaminsky, "Black ops of TCP/IP," Black Hat slides, 2011, http://www.slideshare.net/dakami/black-ops-of-tcpip-2011-black-hat-usa-2011 (Accessed May 2018).

[57] R. Dingledine, N. Mathewson, and P. Syverson, "Tor: The second-generation onion router," in *USENIX Security Symposium*, ser. USENIX Security, 2014, pp. 303–320.

[58] A. Biryukov and I. Pustogarov, "Bitcoin over Tor isn't a good idea," in *Symposium on Security and Privacy*, ser. S&P, 2015, pp. 122–134.

[59] G. Malavolta, P. Moreno-Sanchez, A. Kate, and M. Maffei, "SilentWhispers: Enforcing security and privacy in decentralized credit networks: Not every permissionless payment network requires a blockchain," in *Network and Distributed System Security*, ser. NDSS, 2017.

[60] R. Canetti, "Universally composable security: A new paradigm for cryptographic protocols," in *Foundations of Computer Science*, ser. FOCS, 2001, pp. 136–145.

[61] P. Moreno-Sanchez, T. Ruffing, and A. Kate, "PathShuffle: Credit mixing and anonymous payments for Ripple," in *Privacy Enhancing Technologies Symposium*, ser. PETS, 2017, pp. 436–453.

[62] R. Canetti, "Universally composable signature, certification, and authentication," in *Computer Security Foundations Workshop*, ser. CSFW, 2004, p. 219.

[63] T. Ruffing, P. Moreno-Sanchez, and A. Kate, "P2P mixing and unlinkable Bitcoin transactions," in *Network and Distributed System Security Symposium*, ser. NDSS, 2017.

[64] T. K. Srikanth and S. Toueg, "Simulating authenticated broadcasts to derive simple fault-tolerant algorithms," *Distributed Computing*, vol. 2, no. 2, pp. 80–94, 1987.

[65] D. Dolev, R. Reischuk, and H. R. Strong, "Early stopping in byzantine agreement," *Journal of the ACM*, vol. 37, no. 4, pp. 720–741, 1990.

[66] A. Serjantov, R. Dingledine, and P. F. Syverson, "From a trickle to a flood: Active attacks on several mix types," in *Information Hiding*, ser. IH, 2002, pp. 36–52.

[67] "List of operations," In: Stellar Documentation. Stellar Development Foundation, https://www.stellar.org/developers/guides/concepts/list-of-operations.html (Accessed May 2018).

[68] jedmccaleb (pseudonym), "Drop the concept of rippling in favor of long lived offers," GitHub issue, 2014, https://github.com/stellar/stellar-protocol/issues/6 (Accessed May 2018).

[69] "Ripple implementation," Github repository, https://github.com/ripple (Accessed May 2018).

[70] "ripple-lib: A JavaScript API for interacting with Ripple in Node.js," Github repository, https://github.com/ripple/ripple-lib (Accessed May 2018).

[71] P. Moreno-Sanchez, "PathJoin implementation," Github repository, 2017, https://github.com/pedrorechez/PathJoin (Accessed May 2018).

[72] I. Miers, C. Garman, M. Green, and A. D. Rubin, "Zerocoin: Anonymous distributed e-cash from bitcoin," in *Symposium on Security and Privacy*, ser. S&P, 2013, pp. 397–411.

[73] E. B. Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, and M. Virza, "Zerocash: Decentralized anonymous payments from bitcoin," in *Symposium on Security and Privacy*, ser. S&P, 2014, pp. 459–474.

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

[75] J. Bonneau, A. Narayanan, A. Miller, J. Clark, J. A. Kroll, and E. W. Felten, "Mixcoin: Anonymity for Bitcoin with accountable mixes," in *Financial Cryptography and Data Security*, ser. FC, 2014, pp. 486–504.

[76] G. Maxwell, "CoinJoin: Bitcoin privacy for the real world," Post on Bitcoin Forum, Aug. 2013, https://bitcointalk.org/index.php?topic=279249.

[77] L. Valenta and B. Rowan, "Blindcoin: Blinded, accountable mixes for Bitcoin," in *BITCOIN Workshops*, ser. BITCOIN, 2015, pp. 112–126.

[78] E. Heilman, F. Baldimtsi, and S. Goldberg, "Blindly signed contracts: Anonymous on-blockchain and off-blockchain bitcoin transactions," in *BITCOIN Workshop*, ser. BITCOIN, 2016, pp. 43–60.

[79] E. Heilman, L. Alshenibr, F. Baldimtsi, A. Scafuro, and S. Goldberg, "Tumblebit: An untrusted bitcoin-compatible anonymous payment hub," in *Network and Distributed System Security Symposium*, ser. NDSS, 2017.

[80] J. H. Ziegeldorf, F. Grossmann, M. Henze, N. Inden, and K. Wehrle, "CoinParty: Secure multi-party mixing of bitcoins," in *Conference on Data and Application Security and Privacy*, ser. CODASPY, 2015, pp. 75–86.

[81] G. Bissias, A. P. Ozisik, B. N. Levine, and M. Liberatore, "Sybil-resistant mixing for bitcoin," in *Proceedings of the 13th Workshop on Privacy in the Electronic Society*, ser. WPES, 2014, pp. 149–158.

[82] T. Ruffing, P. Moreno-Sanchez, and A. Kate, "Coinshuffle: Practical decentralized coin mixing for bitcoin," in *European Symposium on Research in Computer Security*, ser. ESORICS, 2014, pp. 345–364.

[83] A. E. Kosba, A. Miller, E. Shi, Z. Wen, and C. Papamanthou, "Hawk: The blockchain model of cryptography and privacy-preserving smart contracts," in *Symposium on Security and Privacy*, ser. S&P, 2016, pp. 839–858.

[84] G. Zyskind, O. Nathan, and A. Pentland, "Enigma: Decentralized computation platform with guaranteed privacy," *CoRR*, vol. abs/1506.03471, 2015, http://arxiv.org/abs/1506.03471.

[85] A. Shamir, "How to share a secret," *Communications of the ACM*, vol. 22, no. 11, pp. 612–613, 1979.

[86] O. Catrina and S. de Hoogh, "Improved primitives for secure multiparty integer computation," in *Security and Cryptography for Networks*, ser. SCN, 2010, pp. 182–199.

[87] P. Dandekar, A. Goel, R. Govindan, and I. Post, "Liquidity in credit networks: A little trust goes a long way," in *Conference on Electronic Commerce*, ser. EC, 2011, pp. 147–156.

[88] L. R. Ford and D. R. Fulkerson, "Maximal flow through a network," *Canadian Journal of Mathematics*, vol. 8, pp. 399–404, 1954.

[89] A. V. Goldberg and R. E. Tarjan, "A new approach to the maximum-flow problem," *J. ACM*, vol. 35, no. 4, pp. 921–940, 1988.

[90] Y. Dinitz, "Dinitz' algorithm: The original version and even's version," in *Theoretical Computer Science*, 2006, pp. 218–240.

[91] P. F. Tsuchiya, "The landmark hierarchy: A new hierarchy for routing in very large networks," in *Communications Architectures and Protocols*, ser. SIGCOMM, 1988, pp. 35–42.

[92] B. Viswanath, M. Mondal, K. P. Gummadi, A. Mislove, and A. Post, "Canal: Scaling social network-based sybil tolerance schemes," in *European Conference on Computer Systems*, ser. EuroSys, 2012, pp. 309–322.

[93] I. Pryvalov, "MPC shared library," http://smpc.ml/, 2015.

[94] C.-P. Schnorr, "Efficient signature generation by smart cards," *Journal of Cryptology*, 1991.

[95] D. R. Stinson and R. Strobl, "Provably secure distributed Schnorr signatures and a (t, n) threshold scheme for implicit certificates," in *Australasian Conference Information Security and Privacy*, ser. ACISP, 2001, pp. 417–434.

[96] S. Makki, "Efficient distributed breadth-first search algorithm," *Computer Communications*, vol. 19, no. 8, pp. 628 – 636, 1996.

VITA

VITA

Pedro Moreno Sanchez received his Ph.D. in the department of computer science at Purdue University in August 2018. He worked as a research assistant under the supervision of Aniket Kate. His doctoral research focused on the study and design of secure, privacy-preserving and decentralized credit networks. Pedro received a B.S. degree and M.S. degree in computer science from Universidad de Murcia in 2011 and 2013 respectively.

From June to December 2012, he worked as research intern at Philips Research Europe, The Netherlands under the supervision of Oscar Garcia-Morchon and Rafael Marin-Lopez. In Summer 2016, he worked as research intern at Ripple Labs, USA under the supervision of Evan Schwartz and Stefan Thomas. After this, in Summer 2017, he worked as research intern at IBM Research-Zurich, Switzerland under the supervision of Christian Cachin.

At Purdue, on the way to the Ph.D., he got the CERIAS/Intel Research Scholarship from January to May 2017 and the Emil Stefanov Fellowship in computer science in April 2017.