SilentWhispers: Enforcing Security and Privacy in Decentralized Credit Networks

Not Every Permissionless Payment Network Requires a Blockchain

Giulio Malavolta*
CISPA, Saarland University

Pedro Moreno-Sanchez*
Purdue University

Aniket Kate
Purdue University

Matteo Maffei
CISPA, Saarland University
TU Vienna

***Abstract*—Credit networks model transitive trust (or credit) between users in a distributed environment and have recently seen a rapid increase of popularity due to their flexible design and robustness against intrusion. They serve today as a backbone of real-world *IOweYou* transaction settlement networks such as Ripple and Stellar, which are deployed by various banks worldwide, as well as several other systems, such as spam-resistant communication protocols and Sybil-tolerant social networks. Current solutions, however, raise serious privacy concerns, as the network topology as well as the credit value of the links are made public for apparent transparency purposes and any changes are logged. In payment scenarios, for instance, this means that all transactions have to be public and everybody knows who paid what to whom.**

In this work, we question the necessity of a privacy-invasive transaction ledger. In particular, we present SilentWhispers, the first distributed, privacy-preserving credit network that does not require any ledger to protect the integrity of transactions. Yet, SilentWhispers guarantees integrity and privacy of link values and transactions even in the presence of distrustful users and malicious neighbors, whose misbehavior in changing link values is detected and such users can be held accountable. We formalize these properties as ideal functionalities in the universal composability framework and present a secure realization based on a novel combination of secret-sharing-based multi-party computation and digital signature chains. SilentWhispers can handle network churn, and it is efficient as demonstrated with a prototype implementation evaluated using payments data extracted from the currently deployed Ripple payment system.

I. Introduction

Motivation. Credit networks [22], [27], [29] model trust among users in a network through a directed, weighted graph, where the value of each edge shows the amount of credit that a user is willing to extend to another. Credit networks constitute the core of a variety of applications, such as trustworthy online marketplaces [51], spam filtering [44], rating systems [33], cloud computing [46], and social networks [45]. Moreover, a few emerging payment systems, such as Ripple [6] and Stellar [4], rely on credit networks to represent and process the credit between peers: this enables multi-currency transactions (in fiat currencies, cryptocurrencies and user-defined currencies) across the globe in a matter of seconds, which are also significantly cheaper than traditional banking solutions [38]. Several major banks worldwide [31], [52], [53], [57] have now started to use Ripple as a backbone for online transactions.

Credit networks and cryptocurrencies. Ripple and Stellar are the first payment settlement networks deployed in practice using the concept of credit network, and it is interesting to compare them with the other payment systems such as Bitcoin. Despite its unquestionable utility, Bitcoin (as any other currency) is limited to transactions where both transacting parties agree on a common currency. Credit networks, instead, smoothly enable cross-currency transactions in any user specified currency (including Bitcoin) and this is one of the reasons of their increasing deployment in banking systems [38].

Similarly to Bitcoin, Ripple and Stellar networks opted for a ledger-based consensus to demonstrate consistency of transactions through transparency. A crucial, and arguably surprising, insight of this work is that while replicated ledgers (or blockchains) are crucial in cryptocurrencies, credit networks have inherent tolerance against transactional inconsistencies and thus they do not require such replicated ledgers nor a global consensus process, paving thereby the way for lightweight transactions.

Privacy in credit networks. Most current credit network designs [44], [51] are centralized, i.e., the credit network is maintained entirely in a server environment. The others, such as Ripple [6] and Stellar [4], make their entire sets of transactions as well as the network topology publicly available to establish credibility through transparency. As a result, credit networks today cannot provide any meaningful privacy guarantee: simple anonymization methods based on pseudonyms, like those employed in Ripple, are not effective, as transactions still remain linkable to each other and they are susceptible to deanonymization attacks such as [43].

In the context of payment settlement networks, for instance, everybody knows who paid what to whom [48]. This state of affairs clearly conflicts with the desire of users, who instead strive for hiding their credit links and their operations: businesses and customers are interested in hiding their credit information and transactions from competitors and even service providers [37], while regular users aim at protecting their transactions as they might reveal personal information (e.g., medical bills or salary).

To tackle this problem, Moreno-Sanchez et al. [47] recently developed a centralized privacy-preserving architecture for credit networks, based on trusted hardware and oblivious

This is a draft (revision 2017–01–12); the most recent revision is available at https://crypsys.mmci.uni-saarland.de/projects/DecentralizedPrivPay

*Both authors contributed equally and are considered to be co-first authors.

computations. Although this solution provides formal privacy guarantees and is efficient enough to support Ripple transactions, the usage of trusted hardware makes its employment in real life quite problematic. The first challenge is who should maintain the trusted hardware and why should the other parties, distributed over the globe and ranging from individuals to banks, trust such an entity for the setup and maintenance of the hardware [50]. The trusted hardware also becomes a critical bottleneck in terms of scalability and reliability, for instance, in case of failure or simple software upgrades. On the other hand, it is hard to imagine a *centralized* cryptographic solution that does not rely on trusted hardware, since it is unclear how one could possibly read the network information required to perform transactions without breaking the privacy of the other users.

Our work moves from the observation that the user's credit links alone determine her available credit in the network and the amount of credit loss she can incur due to misbehaving users. Hence, unlike Bitcoin and other cryptocurrencies, credit networks are an ideal target for a distributed architecture where each user maintains her own credit links locally and checks that her inflow and outflow of credit do not change without explicit consent. We hereby explore this approach, designing a distributed architecture that provides strong privacy guarantees, offers better scalability and reliability, enforces the correctness of transactions, and holds updates of credit links accountable.

Challenges. Designing a privacy preserving, distributed credit network is technically challenging. A first problem is to correctly compute the available credit (max-flow) between two users without leaking the individual link values. A distributed max-flow computation [7] is a natural fit, but generic, off-the-shelf secure multiparty computations among all involved users would be too slow and not scalable enough for real-world networks. A second problem is that some of the users might be dishonest and try to deviate from the protocol in order to selectively deny some legitimate transactions or learn sensitive information. Perhaps most interestingly, ensuring the correctness of the transactions without relying on a privacy invasive ledger requires a fresh architectural design.

Our contribution. In this work, we present SilentWhispers, the first privacy-preserving, distributed credit network. In particular,

- SilentWhispers adapts the traditional landmark routing-based credit network design [60], [62] to a distributed setting, extending it with cryptographic constructions that enable transactions among any two users while preserving strong privacy guarantees.

Technically, SilentWhispers features a novel distributed payment protocol to calculate the available credit between a sender and a receiver without revealing information about the credit in the involved links, nor the transacting users.

Additionally, SilentWhispers allows for holding users accountable for the correctness of credit updates. Interestingly enough, Whispers is the first payment system that dispenses with the deployment of any centrally maintained or replicated, consensus-based ledger to ensure the integrity of the transactions.

- We formalize for the first time the desired privacy properties of a credit network (i.e., value privacy, link privacy, sender privacy, and receiver privacy) as well as security properties (i.e., integrity and accountability) following the universal composability (UC) framework [13] and prove that SilentWhispers constitutes a secure realization.

- We present two extensions of SilentWhispers to enhance its robustness, by supporting offline nodes, and security, by considering fully malicious landmarks. We discuss in detail the tradeoffs in terms of availability, performance, and privacy induced by these extensions.

- We have implemented SilentWhispers as a C++ library and evaluated its performance to demonstrate the practicality of our approach. Specifically, we have extracted transactions from Ripple activity in the period October 2013–January 2015 to set up the parameters of SilentWhispers and thereby simulate a realistic scenario. Our experiments show that in SilentWhispers it is possible to perform a transaction in about 1.3 seconds. The precision of SilentWhispers is optimal (1.0), which implies the absence of false positives and that users do not incur any money loss. The performance evaluation shows that SilentWhispers can be effectively deployed as a real world transaction network, since it is efficient and scales to a growing number of users and payment transactions.

Organization. The rest of the paper is organized as follows: Section II gives a background on credit networks; Section III presents the key ideas underlying our approach; Section IV formally defines the privacy guarantees of a credit network; Section V presents the cryptographic construction; Section VI illustrates SilentWhispers extensions to handle offline nodes and malicious landmarks; Section VII discusses our implementation and performance analysis; Section VIII discusses the related work; and Section IX concludes this work.

II. Background

A. Credit Networks—CN

A credit network (CN) [22], [27], [29] is a weighted, directed graph $\mathbb{G} = (\mathbb{V}, \mathbb{E})$, where vertices $\mathbb{V}$ represent the users and weighted edges (links) $\mathbb{E}$ represent the credit links. The weight on an edge $(u_1, u_2) \in \mathbb{E}$ indicates the *unconsumed* credit that a user u_2 has extended to u_1. For convenience, we denote by $\mathsf{val}_{(u_1,u_2)}$ the non-negative credit of the directed link between u_1 and u_2. A credit network is equipped with four operations (pay, chgLink, test, testLink): pay allows to decrease the credit between two users *only* along credit paths connecting those users; test checks the available credit along credit paths connecting two users; testLink and chgLink enable to test and increase the credit in a direct credit link between two users. We refer to [47] for their formal definitions.

The loss of credit incurred by the users in a credit network is *bounded* [22] by the credit they have extended to a misbehaving user. Moreover, credit loss is *localized* [21], as only honest users who have extended credit to a misbehaving one can incur credit loss. These interesting and useful properties have motivated the use of credit networks in several Sybil and malicious behavior tolerant applications [4], [6], [33], [44]–[46], [51].

Centralized vs. distributed CN designs. In a centralized credit network design, a service provider maintains the complete credit network and updates it according to users transactions. This approach is used, for instance, to avoid fraud in marketplaces [51] and to mitigate spam in e-mail systems [44] or social networks [45]. In a distributed design, every user maintains her own credit links. This approach is adopted for example in a Local Exchange Trading System (LETS) [1], where credit is used to monetize different goods such as childcare or transport.

We aim at a distributed privacy-preserving credit network design, as it better fits the nature of current banking, where each user is responsible for her own credit while each bank is responsible for credit with its customers. There are several challenges that should be addressed in a distributed design: how the routing information is spread along the credit network, how to reconstruct credit paths to perform transactions (e.g., payments), how to ensure the correctness of these transactions without a privacy-invasive ledger, and what privacy properties are desirable.

B. Routing in Distributed CN

A routing protocol computes the discovery of the credit paths among different users in the network. Using max-flow [24], [26], [30] to perform routing in a centralized credit network does not scale to large networks, which is only accentuated in the case of distributed credit networks. Thus, recent works adopt an approximated routing algorithm, called *landmark routing* [60], where only a subset of all possible paths between sender and receiver are calculated. It has been shown in the context of centralized credit networks that landmark routing outperforms the max-flow approach [62].

The idea of the landmark routing protocol is to calculate a path between sender and receiver through an intermediary node called a landmark. In more detail, landmarks are selected nodes which are well known to every other node in the network. Each landmark starts two instances of the Breadth-First-Search (BFS) algorithm rooted at itself. In the first instance only edges in the forward direction are considered, calculating thus shortest paths from the landmark to each node. The second instance only considers edges in the reverse direction, thereby obtaining the shortest paths from each node to the landmark. This results in every user learning her parent in the path to and from each landmark. In the following, we denote by *arborescence* the BFS trees containing shortest paths from each landmark to all the nodes. Correspondingly, we denote by *anti-arborescence* the BFS trees containing shortest paths from all nodes to each of the landmarks.

With such information, when a path between two users (e.g., sender and receiver) needs to be calculated, the shortest path from sender to landmark and the shortest path from landmark to receiver are stitched together to generate a uni-directional path of the form sender $\rightarrow \ldots \rightarrow$ landmark $\rightarrow \ldots \rightarrow$ receiver. Such a path can be calculated for each of the landmarks. We refer the reader to [47] for a detailed description of landmark routing on directed graphs.

Applying landmark routing in a distributed network requires to perform the BFS algorithm in a distributed manner [7], [39]. Throughout the rest of the paper, we assume that the execution of the landmark-based BFS algorithm, which we address by rout, results in every user knowing its parent on the path to the BFS root node. We formalize the rout functionality in Section IV.

C. Transactions in Distributed CN

Here we discuss how credit network operations are executed in a distributed fashion. First, chgLink and testLink operations are locally performed by the two users sharing the corresponding link.

The pay operation is divided into three main steps. First, the sender reconstructs the transaction paths with the receiver through the different landmarks. These transaction paths can be reconstructed from the arborescence and anti-arborescence generated after performing the routing protocol. Second, the credit available in every path is calculated as the minimum credit among the credits available in each of the links in the path. The exact approach to calculate the available credit in a distributed setting and in a privacy-preserving manner is one of the open challenges that we address in this work. Finally, the sender decreases the credit available in the paths by a total amount corresponding to the requested transaction value.

The test operation works as the pay algorithm except that the last step (where the sender decreases the credit available in the paths by the transaction value) is omitted.

III. Problem Definition & Key Ideas

In the following, we describe the security properties of interest in a credit network. We defer formal treatment to the subsequent sections.

Integrity. A credit network achieves integrity if for all pay operations the following holds. Let $\mathsf{path}_1, \ldots, \mathsf{path}_{|\mathsf{LM}|}$ be the paths in a pay operation, $v_1, \ldots, v_{|\mathsf{LM}|}$ be the credit available in such paths and x be the transaction value. Then, after performing a successful pay, every credit link in each path_i is decreased by $x_i \leq v_i$, where $x_1 + \ldots + x_{|LM|} = x$. If pay is unsuccessful, no credit link must get modified.

Serializability. Transactions in a credit network are serializable if, for all sets of pay and 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).

Accountability. A credit network achieves accountability if it is not possible for the adversary to claim a credit value in a link other than the current value (i.e., last credit value agreed between both users sharing the link), without being detected by honest parties.

We now informally describe the privacy properties of interest in a credit network. Value privacy and sender privacy were already formalized in [47], while the others are presented in this work for the first time.

Value privacy. A credit network achieves value privacy if it is not possible for any adversary to determine the total value of a transaction between non-compromised users: The adversary may see some transactions happening over certain links, but it is not possible for any adversary to determine the total transaction value.

Link privacy. A credit network achieves link privacy if it is not possible for any adversary to determine the credit available in a link between non-compromised users.

Sender privacy. A credit network achieves sender privacy if it is not possible for any adversary to determine the sender in a transaction between non-compromised users. The definition of receiver privacy follows along the same lines.

A. Attacker model

We consider a fully distributed 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 initially 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). Furthermore, operations from the landmarks in the protocol are confined to the computation of the minimum value of each path during the transaction protocol. The landmarks could report a value smaller or greater than the actual minimum value of a path. A smaller value would reduce the functionality of the system while no credit would be lost by the honest user. A greater value would be detected by the honest user who does not have enough credit in the transaction path. Therefore, in both cases landmarks would lose customers and (possibly) go out of business. We thus argue that it is in the interest of the landmarks to follow the protocol in order to maintain the availability of their network.

Nevertheless, it is theoretically interesting to realize a system that is resilient against the active corruption of a subset of landmarks. Later, in Section VI, we extend SilentWhispers in order to provide security and privacy guarantees even in presence of actively malicious landmarks.

B. Strawman Approach and Key Ideas

Routing: finding paths in epochs. 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. We leave the design and implementation of a fault-tolerant BFS as an interesting future work.

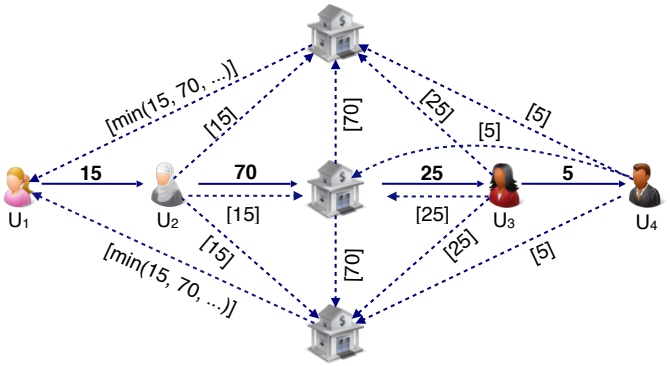

Fig. 1: An illustrative example of the use of SMPC in SilentWhispers: Dashed lines show communication between parties and solid arrows represent credit links, while notation $[a]$ indicates a (secret sharing) share of value a. We consider a payment from user U_1 to U_4. First, every user in the path 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.

Credit on a path: SMPC. 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 Fig. 1. First, every user in the payment path from the sender (U_1) to the receiver (U_4), 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.

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.

Path construction: chained digital signatures. 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.

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 Fig. 2.

Accountability: dispute resolution. In a distributed credit network, the two end-points of a link are responsible for setting its value. We provide an accountability mechanism to establish the real value of a link, in case the two end-points disagree on that. An illustrative example of this idea is depicted in Fig. 3.

In a nutshell, the two end-points establish the current value of the link by signing it with their long-term signing keys. Then, if a transaction is routed through such link, both users log the transaction and sign the new link value. All signatures in our accountability mechanism contain a timestamp to avoid

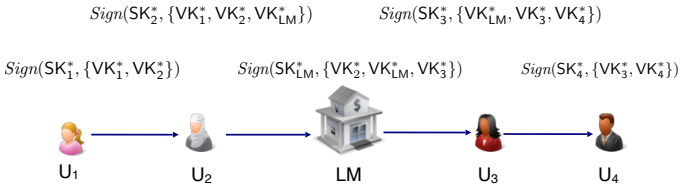

Fig. 2: Illustrative example of path construction in SilentWhispers. Every user i has a pair $(\mathsf{SK}_i^*, \mathsf{VK}_i^*)$ of signing and verification keys. Every user in the path privately exchanges the fresh verification key to both neighbors. Then, each user publishes a signed tuple containing the fresh verification keys of the neighbors and his/her own. A path is correct if contiguous verification keys in the path are equal.

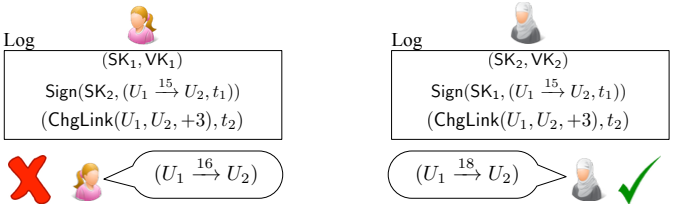

Fig. 3: Illustrative example of the dispute resolution in SilentWhispers. Users sharing a link exchange a signature of the current link's value. When an operation on the link occurs (e.g., increase the link value by 3 credits) and users do not agree on the new link's value, the logs from users allows to solve the dispute (e.g., the new value must be set to 18).

rollback attacks. By inspecting these signatures, a judge can determine the correct value of the link. We assume that long-term keys are associated to the real user's identities (e.g., in an offline contract or using a PKI), such that users are held accountable for their actions.

IV. Security Definition

We define the security and privacy goals of our design using the ideal/real world paradigm from the Universal Composability (UC) framework [13]. We describe in the following the ideal functionality $\mathcal{F}_{\mathsf{CN}}$, which models the intended behavior of the system, in terms of functionality, security, and privacy.

Ideal world. 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 model the synchronous network as an ideal functionality $\mathcal{F}_{\mathsf{NET}}$ as well as the secure and authenticated channels that connect each pair of neighboring nodes, $\mathcal{F}_{\mathsf{SMT}}$, as proposed in [13]. In our abstraction, messages between honest nodes are directly delivered through $\mathcal{F}_{\mathsf{SMT}}$, i.e., the adversary cannot identify whether there is a communication between two honest users. The attacker can corrupt any instance by a message corrupt sent to the respective party ID. The functionality $\mathcal{F}_{\mathsf{NET}}$ 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 $\mathcal{A}$ (passive corruption).

Our ideal functionality for a credit network, $\mathcal{F}_{\mathsf{CN}}$, maintains locally the static information about nodes, links, and their credit using a matrix. Additionally, $\mathcal{F}_{\mathsf{CN}}$ logs all of the changes to the credits between nodes that result from successful transactions and we denote by $\mathsf{val}_{u,u'}^t$ the credit between some u and u' at time t. $\mathcal{F}_{\mathsf{CN}}$ is composed by a set of functionalities ($\mathcal{F}_{\mathsf{ROUT}}$, $\mathcal{F}_{\mathsf{PAY}}$, $\mathcal{F}_{\mathsf{TEST}}$, $\mathcal{F}_{\mathsf{CHGLINK}}$, $\mathcal{F}_{\mathsf{TESTLINK}}$, $\mathcal{F}_{\mathsf{ACC}}$) that interact as follows: $\mathcal{F}_{\mathsf{CN}}$ periodically executes a functionality to update the routing information of the nodes in the network ($\mathcal{F}_{\mathsf{ROUT}}$) using $\mathcal{F}_{\mathsf{NET}}$ as a mean of synchronization. Nodes can contact the ideal functionality to perform transactions ($\mathcal{F}_{\mathsf{PAY}}$), test the available credit ($\mathcal{F}_{\mathsf{TEST}}$), update the credit on a link ($\mathcal{F}_{\mathsf{CHGLINK}}$), test the credit available in a link ($\mathcal{F}_{\mathsf{TESTLINK}}$) or to solve disputes relative to the credit on some link ($\mathcal{F}_{\mathsf{ACC}}$). Under these assumptions, we describe the routines executed by $\mathcal{F}_{\mathsf{CN}}$ in the following.

$\mathcal{F}_{\text{ROUT}}$: The routing algorithm (Fig. 4) allows the functionality to construct the BFS trees required to form transaction paths between a pair of nodes. The landmark fixes the set of children nodes for the computation of the 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.

$\mathcal{F}_{\mathsf{PAY}}$: The algorithm in Fig. 5 provides an ideal functionality of the pay operation in a distributed credit network. The protocol is initiated by the sender Sdr that communicates the two ends of the transaction to the ideal functionality $\mathcal{F}_{\mathsf{PAY}}$ (step 1). For each landmark, $\mathcal{F}_{\mathsf{PAY}}$ 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 $\mathcal{F}_{\mathsf{PAY}}$, 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. $\mathcal{F}_{\mathsf{PAY}}$ 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 $\mathcal{F}_{\mathsf{PAY}}$ of the values to transfer through each path (step 4). $\mathcal{F}_{\mathsf{PAY}}$ 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, $\mathcal{F}_{\mathsf{PAY}}$ updates the credit information of each node involved consistently with the transacted amount. Then $\mathcal{F}_{\mathsf{PAY}}$ 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.

The $\mathcal{F}_{\mathsf{TEST}}$ functionality computes the credit available on the paths connecting any two nodes in the network, and it works analogously to the steps 1-3 in $\mathcal{F}_{\mathsf{PAY}}$. At any point in the execution each node can query $\mathcal{F}_{\mathsf{TESTLINK}}$ to obtain information about her adjacent links and each pair of neighboring nodes can jointly query $\mathcal{F}_{\mathsf{CHGLINK}}$ to update their link or generate a new one.

$\mathcal{F}_{\mathsf{ACC}}$: The accountability algorithm depicted in Fig. 6 solves eventual disputes among pairs of nodes for the value of the link between them. Any two nodes can contact the $\mathcal{F}_{\mathsf{ACC}}$ functionality with their claim for the value of that link (step 1). If the two values are equal or the two nodes are not arguing about the same link, $\mathcal{F}_{\mathsf{ACC}}$ informs the nodes and interrupts the execution (step 2). Otherwise $\mathcal{F}_{\mathsf{ACC}}$ retrieves the current value of the link from $\mathcal{F}_{\mathsf{CN}}$ and reports to each node the index of the instance that queried the correct value, if any. The functionality $\mathcal{F}_{\mathsf{ACC}}$ iterates this procedure with older versions of the link value (step 3) until the initial state of the system is reached. If still no value provided by any clients matches the one recorded by $\mathcal{F}_{\mathsf{CN}}$, $\mathcal{F}_{\mathsf{ACC}}$ returns $\perp$ (step 4).

Discussion. 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.

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

1) LM sends to $\mathcal{F}_{\text{ROUT}}$ two tuples of the form $(u_1, \ldots, u_m)$, indicating the sets of neighbors of LM in the arborescence and anti-arborescence, respectively.
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 $(\mathsf{sid}, \mathsf{ID}_{\mathsf{LM}}, h, u_{\mathsf{p}})$ via $\mathcal{F}_{\mathsf{SMT}}$, 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 (u', 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. 4: Ideal functionality for the rout operation

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

1) For each LM, a sender Sdr sends the tuple $(\mathsf{Sdr}, \mathsf{Rvr}, \mathsf{Txid}, \mathsf{ID}_{\mathsf{LM}})$ to $\mathcal{F}_{\mathsf{PAY}}$, where Rvr, Txid, and $\mathsf{ID}_{\mathsf{LM}}$ denote the receiver, the transaction identifier, and the landmark identifier of the transaction.
2) For each LM, $\mathcal{F}_{\mathsf{PAY}}$ derives the path from Sdr to Rvr, by concatenating the respective paths to LM, as follows: starting from Sdr and Rvr, $\mathcal{F}_{\mathsf{PAY}}$ sends $(\mathsf{Txid}, \mathsf{ID}_{\mathsf{LM}}, u)$ via $\mathcal{F}_{\mathsf{SMT}}$, 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 $\mathcal{F}_{\mathsf{PAY}}$ 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 $(u', \mathsf{Txid}, \mathsf{ID}_{\mathsf{LM}})$ to indicate the next user on the path to LM. $\mathcal{F}_{\mathsf{PAY}}$ 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, $\mathcal{F}_{\mathsf{PAY}}$ 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 Rvr through LM ($\mathsf{path}_{\mathsf{LM}}$). $\mathcal{F}_{\mathsf{PAY}}$ sends then (P, Txid) to the Sdr via $\mathcal{F}_{\mathsf{SMT}}$.
4) Sdr can either abort by sending $(\perp, \mathsf{Txid})$ to $\mathcal{F}_{\mathsf{PAY}}$ or send a set of tuples $(\mathsf{ID}_{\mathsf{LM}}, x_{\mathsf{LM}}, \mathsf{Txid})$ to $\mathcal{F}_{\mathsf{PAY}}$ via $\mathcal{F}_{\mathsf{SMT}}$.
5) For each LM, $\mathcal{F}_{\mathsf{PAY}}$ 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 $\mathcal{F}_{\mathsf{SMT}}$. 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 $\mathcal{F}_{\mathsf{PAY}}$ checks whether for the corresponding edge $\mathsf{e} : \mathsf{val}_{\mathsf{e}} \geq x_{\mathsf{LM}}$, and if yes $\mathcal{F}_{\mathsf{PAY}}$ subtracts x_{LM} from $\mathsf{val}_{\mathsf{e}}$. If one of the conditions is not met or there is at least one $(\perp, \mathsf{ID}_{\mathsf{LM}}, \mathsf{Txid})$ message, then $\mathcal{F}_{\mathsf{PAY}}$ aborts the transaction and restores the credits on the corresponding links of $\mathsf{path}_{\mathsf{LM}}$.
6) $\mathcal{F}_{\mathsf{PAY}}$ sends to Rvr the tuple $(\mathsf{Sdr}, \mathsf{Rvr}, v, \mathsf{Txid})$ via $\mathcal{F}_{\mathsf{SMT}}$, where v is the total amount transacted to Rvr. Rvr can either abort the transaction by sending $(\perp, \mathsf{Txid})$ or allow it by sending $(\mathsf{success}, \mathsf{Txid})$.
7) For each LM, $\mathcal{F}_{\mathsf{PAY}}$ 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 Rvr to the Sdr, starting from the Rvr. Such a user can either reply with $(\perp, \mathsf{Txid})$ to conclude the functionality or with $(\mathsf{accept}, \mathsf{Txid})$ to have $\mathcal{F}_{\mathsf{PAY}}$ passing the message $(\mathsf{success}, \mathsf{Txid})$ (or $(\perp, \mathsf{Txid})$) to the next user until Sdr is reached.

Fig. 5: Ideal functionality for the pay operation

- *Integrity*: In the ideal world, integrity is guaranteed by the ideal functionality, who maintains a database of the link values

Functionality $\mathcal{F}_{\mathsf{ACC}}$

1) Two nodes u_0 and u_1 contact $\mathcal{F}_{\mathsf{ACC}}$ by sending the tuple $(\mathsf{val}_0, u_0, u_0')$, $(\mathsf{val}_1, u_1', u_1)$ respectively, via $\mathcal{F}_{\mathsf{SMT}}$.
2) The functionality $\mathcal{F}_{\mathsf{ACC}}$ checks whether $u_0' = u_1$ and $u_1' = u_0$, if this is not the case then $\mathcal{F}_{\mathsf{ACC}}$ sends the distinguished symbol $\perp$ to both of the instances and aborts the execution. If $\mathsf{val}_0 = \mathsf{val}_1$ the functionality replies with the distinguished symbol $\top$ and interrupts the protocol.
3) Set t to be the current time and iterate until $t = 0$
   a) $\mathcal{F}_{\mathsf{ACC}}$ queries $\mathcal{F}_{\mathsf{CN}}$ to retrieve $\mathsf{val}^t_{(u_0,u_1)}$.
   b) If $\mathsf{val}^t_{(u_0,u_1)} = \mathsf{val}_0$, then $\mathcal{F}_{\mathsf{ACC}}$ sends the tuple $(0, \mathsf{val}_0, \mathsf{val}_1)$ to u_0 and u_1 via $\mathcal{F}_{\mathsf{SMT}}$. Else, if $\mathsf{val}^t_{(u_0,u_1)} = \mathsf{val}_1$, then the functionality sends the tuple $(1, \mathsf{val}_0, \mathsf{val}_1)$ to u_0 and u_1 via $\mathcal{F}_{\mathsf{SMT}}$. Otherwise, $\mathcal{F}_{\mathsf{ACC}}$ sets $t = t - 1$.
4) $\mathcal{F}_{\mathsf{ACC}}$ returns $(\perp, \mathsf{val}_0, \mathsf{val}_1)$.

Fig. 6: Ideal functionality for the accountability mechanism

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.

- *Accountability*: We consider attacks aiming to alter the credit on the network. Hence we can assume without loss of generality that any malicious behavior necessarily results in at least one pair of neighboring nodes not agreeing on the value of one of their shared links. We further note that the ideal functionality updates the values of the links involved in a transaction only if all the corresponding nodes accept it. It follows that, for each link, the functionality always retrieves a value which the two end-points agreed upon and is correctly updated upon each successful transaction. Therefore the accountability algorithm can successfully determine which of the two nodes is claiming the correct value of the link, if any.

- *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.

- *Link privacy*: The reasoning is similar, since we model each link as a value shared between two users and concealed from the others.

- *Sender/Receiver privacy:* For sender and receiver privacy, we note 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.

For a detailed discussion of a few interesting design choices underlying the ideal world, we refer to Appendix A.

UC-Security. We define the concept of UC-security, which intuitively captures under which conditions a cryptographic system in the real world constitutes a secure realization of the ideal world. 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 random coins of all the involved machines).

Definition 1 (UC-Security): A protocol τ UC-realizes an ideal functionality $\mathcal{F}$ 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 $\mathsf{EXEC}_{\mathcal{F},\mathcal{S},\mathcal{E}}$ are computationally indistinguishable.

V. Cryptographic construction

A. Building Blocks

In the following we provide the intuitive description of the cryptographic primitives that we deploy in our system.

Secret Sharing. A Secret Sharing Scheme ($\mathcal{T}$) [56] allows a dealer to distribute shares of a secret among different parties such that any number of shares below a certain threshold reveals no information about the secret itself in the information-theoretic sense, while an arbitrary subset of shares above the threshold allows a receiver to fully reconstruct the secret. In the following, we denote the shares of a secret value by $[\![s_1, \ldots, s_m]\!]$, where m is the number of landmarks. We set the threshold $t < m/2$ so that multiplication of shares can be handled by m computing parties.

Distributed Minimum Computation. On input secret shares of values $x_1, \ldots, x_n$ shared using scheme $\mathcal{T}$ among a set of computing parties, a multi-party computation protocol *min*() results in each party having a share of the minimum of those values. We employ a distributed integer comparison protocol [16] for this distributed computation.

Digital Signatures. A signature scheme Π allows one to compute a tag (σ) on a given message m that proves the authenticity of it. We denote this operation as $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}, \mathsf{m})$, where sk is a secret key. In particular it should be infeasible for anybody not possessing the secret key to produce a valid tag on any arbitrary message. In addition, the validity of the tag-message pair can be publicly verified via an associated verification key (vk). We denote this operation as $\mathsf{Verify}(\mathsf{vk}, \mathsf{m}, \sigma)$. We refer to [14] for the security definition in the UC framework.

B. Protocol Description

System assumptions. 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 routing epoch as users perform a new instance of the link setup protocol. This is crucial as this allows users to know the root of all BFS 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.

Protocol 1: Link setup protocol.

Input:
- $\boldsymbol{u_1}$, $\boldsymbol{u_2}$: Nodes creating a shared link
- val: Value of the link $u_1 \rightarrow u_2$
- $(\mathsf{sk}^*_{u_i}, \mathsf{vk}^*_{u_i})$: User i long term keys
- $epoch$: Current epoch

1 u_1 sends $\sigma_1 \leftarrow \mathsf{Sign}(\mathsf{sk}^*_1, (\mathsf{settled}||\mathsf{vk}^*_1||\mathsf{vk}^*_2||\mathsf{val}||epoch))$ to u_2

2 u_2 sends $\sigma_2 \leftarrow \mathsf{Sign}(\mathsf{sk}^*_2, (\mathsf{settled}||\mathsf{vk}^*_1||\mathsf{vk}^*_2||\mathsf{val}||epoch))$ to u_1

3 if $\mathsf{Verify}(\mathsf{vk}^*_2, (\mathsf{settled}||\mathsf{vk}^*_1||\mathsf{vk}^*_2||\mathsf{val}||epoch), \sigma_2)$ then u_1 stores $(\sigma_1, \sigma_2, \mathsf{st}_{\mathsf{vk}^*_1,\mathsf{vk}^*_2} := (\mathsf{settled}||\mathsf{vk}^*_1||\mathsf{vk}^*_2||\mathsf{val}||epoch))$

4 if $\mathsf{Verify}(\mathsf{vk}^*_1, (\mathsf{settled}||\mathsf{vk}^*_1||\mathsf{vk}^*_2||\mathsf{val}||epoch), \sigma_1)$ then u_2 stores $(\sigma_1, \sigma_2, \mathsf{st}_{\mathsf{vk}^*_1,\mathsf{vk}^*_2} := (\mathsf{settled}||\mathsf{vk}^*_1||\mathsf{vk}^*_2||\mathsf{val}||epoch))$

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 [23]). Moreover, we require all the involved users to be online during a given transaction or the presence of a synchronizer (among the others [7], [59]) for the execution of the algorithms. We discuss later in Section VI-A on how to relax this condition.

Notation. We use bold terms to denote the input fields added only for readability. The rest of inputs are locally held by involved users. Moreover, We use the following notation to describe our protocols.

$\mathsf{p}(u, i)$	Parent of node u in the path_i
$\mathsf{c}(u, i)$	Child of node u in the path_i
val_{u_1,u_2}	Credit value on link $u_1 \rightarrow u_2$
st_{u_1,u_2}	Last value on $u_1 \rightarrow u_2$ agreed by u_1, u_2
$\mathsf{m}[i]$	Element at position i in array m
vk^i_u	Fresh verification key of user u in path_i
max	Maximum path length (system parameter)
ts	Current timestamp

Setup. Users have access to a synchronous network through $\mathcal{F}_{\mathsf{NET}}$. Every pair of users sharing a credit link communicate through a secure and authenticated channel, described by $\mathcal{F}_{\mathsf{SMT}}$. Secure realizations of $\mathcal{F}_{\mathsf{NET}}$ and $\mathcal{F}_{\mathsf{SMT}}$ have been proposed in [13]. Finally, users have access to the routing protocol described in $\mathcal{F}_{\mathrm{ROUT}}$: 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. We show in Appendix B that the landmark routing algorithm UC-realizes $\mathcal{F}_{\mathrm{ROUT}}$.

Link setup. 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.

Transaction. 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., step 8) 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. 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 (line 8). 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 9). 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. 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 11-12). This results into each landmark having a share of the minimum value for each path which is then sent to the sender (line 13).

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.

Phase 3: Updating link values. 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 18-22). 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 25-29). 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 (see Section II-A) and the dispute can be resolved later following the accountability protocol.

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 (steps 15-29), as the test operation only requires the sender to learn the available credit. 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.

Accountability. Our accountability protocol requires some entity (e.g., a judge) that can enforce decisions externally to the system. Given two users u_0, u_1 disputing the current value on their shared link, this protocol allows each user to provide the judge with her view of the link $(\mathsf{st}_0, \mathsf{st}_1)$, the corresponding confirmations $((\sigma_0^0, \sigma_0^1), (\sigma_1^0, \sigma_1^1))$ and the signatures of the sender and the receiver of the last transaction $((\sigma_0^{\mathsf{Sdr}}, \sigma_0^{\mathsf{Rvr}}), (\sigma_1^{\mathsf{Sdr}}, \sigma_1^{\mathsf{Rvr}}))$. Given that, the judge can decide the valid link state following the steps we describe in Protocol 3.

First, the judge checks the validity of signatures (σ_i^0, σ_i^1) on the link states presented by each user u_i (lines 2-5). If only one user provides valid signatures, her state is taken as the valid one. If no user provides valid signatures, the judge cannot decide which state is correct. We denote this by $\perp$. Finally, if all signatures are correct, the judge continues to check the content of both states st_0 and st_1.

The judge must determine the currently valid settled value on the link. However, it is possible that a user's view consists on an on hold state and a valid transaction proving that held value has been successfully transmitted from sender to receiver. Thus, the judge first attempts to upgrade such views to a settled state (lines 6-10).

Finally, if both views are in settled state, the judge resolves the dispute by declaring valid the view with the more recent timestamp ($\mathsf{st}[5]$) (lines 11-15). In a simpler case where only one view is in settled state, such view is declared as the valid one. Otherwise, if any user's view is finally settled, the judge outputs $\perp$.

C. System Discussion

Handling faulty users. During the first phase of the transaction protocol, a malicious user could send inconsistent shares

Protocol 2: Transaction protocol.

Input: $\mathsf{Sdr}, \mathsf{Rvr}$: Transaction sender and receiver; $\{\mathsf{path}_1, \ldots, \mathsf{path}_{\mathsf{LM}}\}$ Set of paths Sdr to Rvr; $(\mathsf{sk}^*_{u_i}, \mathsf{vk}^*_{u_i})$: user u_i long term keys

/* Phase 1: signature chain */

1 **for** $i \in |\mathsf{LM}|$ **do**

2 **for** $u \in \mathsf{path}_i$ **do**

3 u creates fresh keys $(\mathsf{sk}_u, \mathsf{vk}_u)$, $\sigma_u := \mathsf{Sign}(\mathsf{sk}^*_u, \mathsf{vk}_u)$ and sends $(\sigma_u, \mathsf{vk}_u)$ to $\mathsf{p}(u,i)$ and to $\mathsf{c}(u,i)$

4 u receives $(\sigma_{\mathsf{c}(u,i)}, \mathsf{vk}_{\mathsf{c}(u,i)})$ from $\mathsf{c}(u,i)$ and $(\sigma_{\mathsf{p}(u,i)}, \mathsf{vk}_{\mathsf{p}(u,i)})$ from $\mathsf{p}(u,i)$

5 u receives from $\mathsf{c}(u,i)$ shares $[\![s'_1, \ldots, s'_{|\mathsf{LM}|}]\!]$, u reconstructs v' from $[\![s'_1, \ldots, s'_{|\mathsf{LM}|}]\!]$ and checks whether $v' = \mathsf{val}_{\mathsf{c}(u,i),u}$

6 u creates $[\![s_1, \ldots, s_{|\mathsf{LM}|}]\!]$ for the value $\mathsf{val}_{u,\mathsf{p}(u,i)}$ and sends them to $\mathsf{p}(u,i)$

7 **if** $\mathsf{Verify}(\mathsf{vk}^*_{\mathsf{c}(u,i)}, \mathsf{vk}_{\mathsf{c}(u,i)}, \sigma_{\mathsf{c}(u,i)}) \wedge \mathsf{Verify}(\mathsf{vk}^*_{\mathsf{p}(u,i)}, \mathsf{vk}_{\mathsf{p}(u,i)}, \sigma_{\mathsf{p}(u,i)})$ **then**

8 **for** $j \in |\mathsf{LM}|$ **do**

9 u creates $\mathsf{m} := (\mathsf{vk}_{\mathsf{c}(u,i)}||[\![s'_j]\!]||\mathsf{vk}_u||\mathsf{vk}_{\mathsf{p}(u,i)}||[\![s_j]\!]||\mathsf{Txid}||ts)$, u creates $\sigma_{\mathsf{LM}_j} \leftarrow \mathsf{Sign}(\mathsf{sk}_u, \mathsf{m})$ and finally sends $(\sigma_{\mathsf{LM}_j}, \mathsf{m})$ to LM_j

10 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

/* Phase 2: Minimum computation */

11 **for** $i \in |\mathsf{LM}|$ **do**

12 Each LM checks whether $|\mathsf{path}_i| = \mathsf{max} \wedge \forall j \in \{1, \ldots, |\mathsf{path}_i|\} : \mathsf{Verify}(\mathsf{m}_j[3], \mathsf{m}_j, \sigma_j) \wedge \mathsf{m}_j[1] = \mathsf{m}_{j-1}[3] \wedge \mathsf{m}_j[4] = \mathsf{m}_{j+1}[3] \wedge \mathsf{m}_{j-1}[6] = \mathsf{m}_j[6] = \mathsf{m}_{j+1}[6]$

13 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.

14 Each LM sends the resulting tuples $(i, s_{min_i}, \mathsf{vk}^i_1, \mathsf{vk}^i_{\mathsf{max}})$ to Sdr

/* Phase 3: Carrying out transaction */

15 Sdr reconstructs the tuples (i, min_i) and verifies that vk^i_1 and $\mathsf{vk}^i_{\mathsf{max}}$ are the first and last keys of path_i as she expects

16 **for** $i \in |\mathsf{LM}|$ **do**

17 Sdr chooses the transaction value x_i, generates $tx_i := (ts||x_i||\mathsf{Txid}||\mathsf{vk}_{\mathsf{Sdr}}||\mathsf{vk}_{\mathsf{Rvr}})$ and $\sigma^i_{\mathsf{Sdr}} := \mathsf{Sign}(\mathsf{sk}_{\mathsf{Sdr}}, tx_i)$, and sends $(tx_i, \sigma^i_{\mathsf{Sdr}})$ to the nodes in path_i

18 **for** $u \in \mathsf{path}_i$ **do**

19 u checks $\mathsf{Verify}(\mathsf{vk}_{\mathsf{Sdr}}, tx_i, \sigma_{i_{\mathsf{Sdr}}})$, x_i is smaller than the value $\mathsf{val}_{u,\mathsf{p}(u,i)}$, and previous link $\mathsf{c}(u,i) \rightarrow u$ has been reduced by x_i

20 u decreases link value on path_i by x_i resulting in x'_i

21 u creates $\mathsf{m} := (\mathsf{on\ hold}||\mathsf{vk}^*_u||\mathsf{vk}^*_{\mathsf{p}(u,i)}||x'_i||tx_i)$, $\sigma_u := \mathsf{Sign}(\mathsf{sk}^*_u, \mathsf{m})$ and sends (σ_u, m) to $\mathsf{p}(u,i)$

22 u receives $\sigma_{\mathsf{p}(u,i)} := \mathsf{Sign}(\mathsf{sk}^*_{\mathsf{p}(u,i)}, \mathsf{m})$ from $\mathsf{p}(u,i)$

23 u and $\mathsf{p}(u,i)$ locally store $(\mathsf{st}_{\mathsf{vk}^*_u, \mathsf{vk}^*_{\mathsf{p}(u,i)}} := \mathsf{m})$ and $(\sigma_{\mathsf{p}(u,i)}, \sigma_u)$

24 Rvr $\sigma^i_{\mathsf{Rvr}} := \mathsf{Sign}(\mathsf{sk}_{\mathsf{Rvr}}, tx_i)$ and sends $(tx_i, \sigma^i_{\mathsf{Rvr}})$ to Sdr

25 **for** $i \in |\mathsf{LM}|$ **do**

26 Rvr sends $(tx_i, \sigma^i_{\mathsf{Sdr}}, \sigma^i_{\mathsf{Rvr}})$ to every node in path_i

27 **for** $u \in \mathsf{path}_i$ **do**

28 u creates $\mathsf{m} := (\mathsf{settled}||\mathsf{vk}^*_u||\mathsf{vk}^*_{\mathsf{c}(u,i)}||x'_i||ts)$, $\sigma_u := \mathsf{Sign}(\mathsf{sk}^*_u, \mathsf{m})$ and sends (σ_u, m) to $\mathsf{c}(u,i)$

29 u receives $\sigma_{\mathsf{c}(u,i)} := \mathsf{Sign}(\mathsf{sk}^*_{\mathsf{c}(u,i)}, \mathsf{m})$ from $\mathsf{c}(u,i)$

30 u and $\mathsf{c}(u,i)$ locally store $(\mathsf{st}_{\mathsf{vk}^*_u, \mathsf{vk}^*_{\mathsf{c}(u,i)}} := \mathsf{m})$ and $(\sigma_{\mathsf{c}(u,i)}, \sigma_u)$

Protocol 3: Accountability protocol.

Input: $(\mathsf{vk}_0^*), (\mathsf{vk}_1^*)$: Keys for u_0, u_1
$(\mathsf{st}_0, \sigma_0^0, \sigma_0^1), (\mathsf{st}_1, \sigma_1^1, \sigma_1^0)$: Link state for u_0, u_1
$(\sigma_0^{\mathsf{Sdr}}, \sigma_0^{\mathsf{Rvr}}), (\sigma_1^{\mathsf{Sdr}}, \sigma_1^{\mathsf{Rvr}})$: Signatures of tx for u_0, u_1

```
/* Check signatures on link states                                   */
for i ∈ {0,1} do
    if ¬Verify(vk*_i, st_i, σ^i_i) ∨ ¬Verify(vk*_{1−i}, st_i, σ^{1−i}_i) then
        if
        Verify(vk*_{1−i}, st_{1−i}, σ^{1−i}_{1−i}) ∧ Verify(vk*_i, st_{1−i}, σ^i_{1−i})
        then
            return st_{1−i}
        else return ⊥
/* Upgrade states on hold                                            */
for i ∈ {0,1} do
    if st_i[1] = on hold then
        tx := (st_i[5]||st_i[6]||st_i[7]||st_i[8]||st_i[9])
        if Verify(st_i[8], tx, σ_i^Sdr) ∧ Verify(st_i[9], tx, σ_i^Rvr)
        then
            st_i[1] = settled
/* Settled link views                                                */
for i ∈ {0,1} do
    if st_i[1] = settled ∧ st_{1−i}[1] = settled then
        if st_i[5] > st_{1−i}[5] then return st_i
    if st_i[1] = settled then return st_i
return ⊥
```

of her link's value to a landmark. The landmark notices this in the second phase since inconsistent shares are not signed by honest users. Then, the landmark directly assigns zero to that path's capacity and continues processing other paths for the current transaction.

During the third phase, a user could refuse to update a link value during a transaction. On the one hand, if this happens while credit is being set on hold from sender to receiver, the receiver does not receive the expected credit and thus he does not sign the transaction. Consequently, after a certain timeout, users in the path can safely release the held value for such transaction. On the other hand, if this occurs when credit is being settled from receiver to sender, the accountability mechanism provided in SilentWhispers allows the honest counterparty of the link to show the transaction signed by both sender and receiver as a valid proof to settle the new value in the link.

We note that two neighboring corrupted nodes may apply arbitrary modifications to their shared links without necessarily following the procedure specified above. This does not affect the credit balance for honest users: Our mechanism for link updates ensures that any honest user puts on hold the credit on any outgoing link only after the same amount of credit has been held on an incoming edge. That is, the total balance of the intermediate users of a given transaction is maintained throughout the execution of the transaction. Given that, it follows that no malicious behavior can cause loss of credit for honest users.

Best-effort concurrent transactions. Transactions over disjoints sets of links can be easily carried out concurrently. If, however, two or more transactions require more credit than available at a *shared* link, the user of such link notices that when required to decrease her link. This user can handle this situation in an optimistic manner: she can put on hold the value for one of the transactions and abort the others. As previously mentioned, aborted transactions do not affect credit on the network since corresponding receivers do not sign the aborted transactions. Then, each sender of an aborted transaction randomly chooses a waiting period after which she reissues the transaction. This mechanism closely resembles the behavior of users in currently deployed credit networks such as Ripple [2], where better liquidity decreases the odds for a deadlock in a credit network.

Network churn. When a credit link is created (e.g., a new user enters the credit network creating a credit link to an existing user), it cannot be immediately used to perform transactions. It becomes usable next time the routing algorithm is performed (i.e., in the next *epoch*). When a credit link is deleted, its value is set to 0 and thus immediately unusable to perform transactions through it. Finally, we describe the case when a user goes offline in Section VI-A.

D. Security Analysis

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

Theorem 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 $\mathcal{F}_{\mathsf{CN}}$ in the $\mathcal{F}_{\mathsf{NET}}$, $\mathcal{F}_{\mathsf{SMT}}$-hybrid model.

E. Application of SilentWhispers to Other Credit Networks

We describe how SilentWhispers can be used to realize a secure and privacy-preserving distributed variant of previous credit network-based systems, thereby demonstrating its general applicability. For that, we observe that current applications based on credit networks differ on two aspects: the meaning of credit values on each link and how a path is updated as a result of a transaction. We refer to [47] for a characterization of these aspects for Ripple, Bazaar and Ostra.

SilentWhispers can simulate the Ostra credit network by setting the value on a link $i \to j$ as the number of remaining emails that j allows from i. Interestingly, chgLink and testLink can be then performed locally between i and j. Moreover, test and pay operations can be used in a distributed fashion, as defined for SilentWhispers, thereby covering all necessary functionality.

The Bazaar credit network can be also realized in a distributed manner by defining credit values as the accumulated value of successful transactions between two users sharing a link. As before, chgLink and testLink operations are locally carried out, while test and pay can be performed as defined for SilentWhispers. Interestingly, a successful transaction in Bazaar additionally requires to restore the values on the links used for a transaction and the addition of the paid credit to the link between sender and receiver. This functionality can be simulated in SilentWhispers as follows: the sender anonymously (e.g., using Tor) publishes the feedback along

with Txid; users involved in the transaction path for Txid can then locally restore their credit links; finally, the sender can locally update her link with the receiver.

VI. Extending SilentWhispers

A. Boosting the Availability of the System

We assumed throughout our discussion that all users are online, an assumption that might not hold in practice. To relax it, we provide a mechanism to allow transactions to take place even when some (or all) intermediate users in a path are offline. Intuitively, we allow each user to *store information on the landmarks* so to let them impersonate it during the execution of the routing algorithm or a transaction but without revealing critical sensitive data. When the user goes back online, she can retrieve her updated information and verify the correctness of the changes. Due to space constraints, we show the detailed protocol in Appendix C and describe it here on a high level.

When a user u is about to go offline, she first notifies her neighbors and then she secret shares the value of each of her links and her long term key sk_u^* among the landmarks. Finally, she sends them together with the long term verification key and the signatures over all link shares (produced by herself and the neighbors) to all the landmarks. Intuitively, shares of the link value allow the landmarks to jointly perform operations on the links values without revealing the value itself. The signatures $(\sigma_{u,i}, \sigma_i)$ are crucial to enable accountability of the link updates. Shares s'_j allow landmarks to jointly sign the link state after each update for accountability. Thus, landmarks can impersonate the user and execute the protocols on her behalf.

In particular, during the landmark routing protocol, each landmark locally impersonates the offline users. The transaction protocol requires instead the following modifications. In the first phase, offline users are impersonated by the landmarks, who interact with online neighbors to create the signature chain. In the second phase, since landmarks already have the shares of link values from offline users, they can compute the credit value on a path and send it to the sender.

Finally, the third phase is performed as follows. The sender generates shares $[\![s_1, \ldots, s_{|\mathsf{LM}|}]\!]$ for the value to be subtracted in the path along with $\sigma_i := \mathsf{Sign}(sk_{\mathsf{Sdr}}, (s_i, \mathsf{Txid}))$, a signature of each share ensuring the authenticity of the request. Then, the sender passes it over to the next user in the path.

The steps for each user in the path depend on whether it is online or not. If the user is online, it gets all the tuples (s_i, σ_i), verifies them, reconstructs the original value and reduces it (i.e., on hold) from the credit available in the link with the next neighbor. Moreover, the user signs the new link state together with the next neighbor or with the landmarks if the neighbor is offline; if offline, the user sends to each landmark a pair (s_i, σ_i), who in turns verifies it, checks through a multi-party computation protocol whether s_i is indeed the share of a valid value (i.e., smaller or equal than the path capacity) and eventually reduces (i.e., on hold) s_i from the share s_j of the path's link. As before, the new link state is signed together with the next neighbor if online or locally by the landmarks. Finally, each landmark logs the tuple (s_i, σ_i) to be later on revealed to the offline user. This mechanism is repeated for each user until the receiver of the transaction is reached. The protocol works similarly for the path from the receiver to the sender.

The test operation is processed analogously except for link updates. The chgLink operation is, in the offline case, issued by the requesting user in the form of signed shares over the value to subtract from (or to add to) the given link. The landmarks verify the respective signatures, collectively compute whether the link has enough capacity and eventually carry out the operation, logging the request. The testLink operation is simply performed locally by the requester and the link's counterparty if online, or the landmarks if offline.

Once a user goes online again, it can simply retrieve the updated share of the link value from the landmarks together with the logged operations performed while it was offline. The user can then reconstruct the values subtracted to her links and identify the users originating the requests through the signatures on the shares.

Analysis. The correctness of the underlying SMPC protocol combined with the integrity of our original construction are enough to prove that our extension achieves integrity. Also, this extension achieves value privacy: loosely speaking, all the information that each landmark learns about the values of the links comes in the form of shares from the secret-sharing scheme, which do not reveal any information about the respective value (unless all the landmarks are colluding, which is excluded by assumption). The accountability of link values is guaranteed by unforgeability of the pair of signatures from the long term keys (or their shares if the user went offline).

These modifications enhance the performance of SilentWhispers, inasmuch landmarks impersonate offline users in both routing and transaction protocols and the corresponding rounds of communications are avoided. However, this reveals the network topology to the landmarks, thus losing the notions of link privacy as well as sender or receiver privacy. We observe that it is hard for any distributed protocol to preserve link privacy as well as sender and receiver privacy when users go offline, unless some party is assumed to be honest. In the extreme case where all of the users go offline except for two, even a semi-honest adversary can easily figure out the sender and receiver of an eventual transaction. Thus, this can be seen as a general issue when handling offline users in a distributed scenario instead of an inherent limitation of our construction.

B. Security against Malicious Landmarks

In our initial security model, we only considered passive corruption of a proper subset of landmarks. In the following we show how to provide security guarantees in presence of malicious landmarks.

In the standard protocol the landmarks are deployed for the computation of the path's capacity. Therefore, we can deploy standard techniques to upgrade our system to a stronger security setting while preserving UC-security, such as the transformation of [19]. We shall note that the previously mentioned transformation allows the system to handle only static corruptions of the nodes. While it is certainly possible to extend the security of our system against adaptive corruption of peers adopting fully UC-secure SMPC constructions like [15],

this however comes with a considerable increase in cost both in terms of computation and cryptographic assumptions.

VII. Performance Analysis

A. Implementation

We have developed a C++ implementation to demonstrate the feasibility of SilentWhispers. We focus in particular on the transaction protocol (Protocol 2), 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 [5], on the Shamir's information theoretic construction [56] for secret sharing, and on Schnorr's signatures [55], [58] 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.

B. Performance

We conduct our experiments in machines with 3.3 GHz processor and 8 GB RAM to carry out distributed 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 (see Section VII-A).

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 is defined in Algorithm 2-line 20 and 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. As we show in Section VII-C, in practice, $\max$ is a small constant and forwarding such message can be done efficiently even with commodity communication links.

Setup	$(5,1)$	$(5,2)$	$(7,1)$	$(7,2)$	$(7,3)$
Time HbC	0.304	0.314	0.357	0.346	0.349
Time Mal	12.630	13.105	18.973	20.408	21.457

TABLE I: Times in seconds to compute Min(a, b). We use 32 bits to represent a and b. We consider two scenarios: landmarks are honest but curious (HbC) and malicious (Mal). In a setup (n, t), n denotes the total number of landmarks out of which t are compromised.

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 I. 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 $\lceil \log(\max) \rceil$ min operations sequentially. In Table II we show the time to compute the credit in a path for different path lengths. In the honest-but-curious case, computing the minimum credit in a path takes roughly 1.7 seconds for $\max = 20$.

Routing time. For completeness, we consider the other two 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 [39]. 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 operations and it can be run as a background process, thus it does not hinder the performance of the rest of system operations.

C. 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 [6], a currently deployed credit network system with a publicly available ledger containing the network graph and transaction history. 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

Path Length (max)	5	10	20
Time HbC	1.047	1.349	1.745
Time Mal	64.371	85.828	107.285

TABLE II: Times in seconds to compute the credit on a path. We use a setup (7, 3): 7 landmarks, 3 compromised. We study two scenarios: landmarks are honest but curious (HbC) and malicious (Mal).

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 II). A transaction in the currently deployed transaction network Ripple, takes approximately 5 seconds. Thus, our evaluation shows that our approach will not introduce any significant overhead to the transaction time.

D. Discussion

Scalability. The running time of the routing protocol increases with the number of users. However, it can be performed as a background process and thus does not directly impact the time to perform transactions. As shown before, the time to perform a transaction is mostly dictated by the length of the path, which is set to a fixed value of 10 and it does not increase with a growing number of users. Therefore, SilentWhispers has the potential to scale to a large number of users.

Precision and recall. Due to the inefficiency of optimal routing protocols in networks, it is unavoidable to use approximate routing protocols as the landmark routing technique [62]. This approach motivates our results in terms of precision and recall.

Precision measures the ratio between true and false positives. SilentWhispers achieves an optimal precision of 1 (i.e., no false positives) ensuring thus users do not lose money using the system. The calculation of the credit available is performed over the most recent snapshot of the credit network. Therefore, no transaction is answered as successful if there is currently not enough credit in the network. Recall measures the ratio between true and false negatives. SilentWhispers achieves a high recall. Previous solutions [47], [62] have shown that the use of landmark routing incurs in answering less than 5% of the transactions wrongly as unsuccessful. Interestingly, these transactions can be reissued when a new iteration of the routing protocol is done. At that point, new transaction paths are calculated and the transaction can be successfully carried out. Given the efficiency of available distributed BFS algorithms and that we use the same routing technique as Canal [62] and Privpay [47], SilentWhispers achieves the same recall value. Moreover, transactions wrongly tagged as unsuccessful can be reissued so that they are eventually successfully performed.

Handling offline users. When a user is offline and goes back online, she must retrieve from landmarks the updates of her credit links. Then the user must verify that all changes on her credit links have been correctly performed by the landmarks (see Section VI-A).

In order to study the impact of this extension, we have extracted the number of updates performed over credit links of Ripple users. We have randomly selected 100 Ripple users and checked how many transactions involve their credit links over a period of a month (i.e., June 2014). We observe that a maximum of 107 updates are performed over the credit links of a user. In principle, the user must check each update as many times as the number of landmarks, but this task can be parallelized. For each update performed by a given landmark, the user must verify the update and its signature, which can be performed in less than 5 ms. Therefore, even if a user is offline for a period of a month, it will take her about 500 ms to restore her information and check its authenticity.

Moreover, as described in Section VI-A, landmarks must check that the share of the transaction value issued by the sender is indeed smaller than the capacity of the path. For that purpose, landmarks carry out the *less than* operation as a multiparty computation protocol using the two shared values as input. This modification implies an overhead of only 0.330 seconds assuming honest-but-curious landmarks and 21.132 seconds in the presence of malicious landmarks.

Finally, landmarks must sign new link states on behalf of offline users using the shares of their signing keys. For efficiency, this task can be performed using distributed Schnorr signatures [58] so that the time to compute a signature is similar to the creation of a single signature by each landmark, and thus can be computed efficiently. Therefore, the overall time for the transaction operation in the presence of offline users is still dominated by the computation of the credit available in the path (i.e., about 1.3 seconds).

Handling malicious landmarks. For handling malicious landmarks, we need to incorporate SMPC protocols secure against malicious adversaries. In the transaction protocol, the landmarks are confined to the computation of the minimum value of the path, whose execution time in the malicious case is shown in Table I. In a setting with 7 landmarks out of which 3 are corrupted, the computation of the minimum of two values takes 21.457 seconds. This implies that computation of the minimum credit in a path takes approximately 86 seconds. Notice that the employed SMPC library does not yet incorporate the recent significant advancements in the SMPC domain [18], [20], [34], [36]. It employs the older SMPC paradigm [10], [28], and with an improved library, we expect

at least an order of magnitude improvement in the performance of our maliciously secure SMPC executions.

We believe that the resulting running time is still practical in most of the scenarios. For example, it still allows one to significantly speed up intercontinental transactions that currently take up to several days. Nevertheless, we believe that the extension to malicious landmarks is interesting from a theoretical point of view, but not worth to be implemented in practice since landmarks have no incentive in misbehaving, as discussed in Section III-A.

VIII. Related Work

The problem of enforcing privacy-preserving transactions in a credit network has recently been studied by Moreno-Sanchez et al. [47]. They leverage the use of trusted hardware to enforce the privacy of the transacting parties and the transacted value by accessing the credit network information by means of an oblivious RAM algorithm. In contrast to this work, they target a centralized approach, where a unique server stores the complete credit network. Such a solution is hardly applicable to real-life payment systems (e.g., Ripple and Stellar), since it would require them to change their hardware infrastructure to incorporate trusted hardware. Also, the centralized design of [47] is in inherent tension with the distributed nature of consensus algorithms used in Ripple and Stellar, and it is unclear who should play the role of the central server running the trusted hardware. Moreover, the centralized architecture of [47] may constitute a severe problem for scalability, since transactions are handled by a single secure processor and thus cannot be easily parallelized.

Mezzour et al. [41] propose a path-discovery technique that computes a hash tree connecting users that share a credit link. To find a path between two users, they compute the private set intersection of the set of hashed values they hold. A path exists between two users if the intersection set is non-empty. This technique does not allow to reconstruct the path connecting sender and receiver in a private manner, and it omits the concept of credit by considering unweighted links, two features that are crucial for credit networks.

Backes et al. [8] present the concept of Anonymous Webs of Trust, which includes a mechanism to prove the existence of a path of trust certificates among the sender and the receiver in zero-knowledge such that intermediate trust relationships remain private for third parties. This approach, however, relies on a server maintaining all the trust certificates, which must be publicly available, thus breaking link privacy.

There is an extensive literature on privacy-preserving online social networks [9], [17], [32], [49]. Intuitively, an object (e.g., tweet) is secured such that only friends of the object's owner can access it while remaining private for the rest of users. In a credit network however, a given credit link is potentially accessed by any user connected through a credit path. Therefore, the owner of a link cannot establish in advance an access policy to a given link other than the most permissive one, i.e., everybody is allowed to access it.

There exist several proposals to construct privacy preserving payments in Bitcoin based on zero-knowledge proofs [11], [42], centralized mixing [12], decentralized mixing [54], [61], SMPC [25], confidential transactions [40] and smart contracts [35], [64]. These proposals, however, are *not* applicable to the inherently different credit network setting: Bitcoin does not use the concept of credit link and thus there is not a credit network among Bitcoin users. SilentWhispers enables credit network operations such as path finding and payment path updates in a privacy preserving manner while ensuring correctness (e.g., no double spending): each intermediate user verifies that her total balance is preserved after each payment.

Wu et al. [63] present a protocol to compute the shortest path in a privacy-preserving manner based on private information retrieval (PIR) in a centralized setting. Employing their techniques in a decentralized network does not seem feasible, if even possible.

IX. Conclusions

In this work, we presented SilentWhispers, the first privacy-preserving distributed credit network. SilentWhispers achieves a number of privacy properties (i.e., sender, receiver, link, and value privacy), preserves the correctness of transactions, and provides an accountability mechanism to enforce the correctness of link updates. Use of highly connected and readily available nodes such as gateways in Ripple is a crucial ingredient to make our system efficient, robust, and scalable.

We implemented the cryptographic schemes employed in SilentWhispers and demonstrated through an experimental evaluation the practicality of our approach for real-life payment systems. In particular, we showed that our solution allows for fast transactions and scales to a growing number of users. Moreover, we discussed how SilentWhispers can be used to instantiate currently available applications based on credit networks on a distributed manner. Finally, SilentWhispers demonstrates that a privacy-invasive public ledger is not necessary for the secure instantiation of a credit network.

Acknowledgments: We would like to thank the anonymous reviewers for their valuable feedback and our shepherd Erman Ayday for his comments and suggestions. This work was partially supported by the German Federal Ministry of Education and Research (BMBF) through the Emmy Noether program and through funding for the Center for IT-Security, Privacy and Accountability (CISPA) and by the German Research Foundation (DFG) via the collaborative research center "Methods and Tools for Understanding and Controlling Privacy" (SFB 1223).

References

[1] "Local Exchange Trading System," http://tinyurl.com/zea59ob.

[2] "Reliable Transaction Submission," http://tinyurl.com/zdgdlkm.

[3] "SilentWhispers site," http://crypsys.mmci.uni-saarland.de/projects/DecentralizedPrivPay/.

[4] "Stellar website," https://www.stellar.org/.

[5] "MPC Shared Library," http://smpc.ml/, 2015.

[6] F. Armknecht, G. Karame, A. Mandal, F. Youssef, and E. Zenner, "Ripple: Overview and outlook," in *Trust and Trustworthy Computing'15*.

[7] B. Awerbuch, "Reducing complexities of the distributed max-flow and breadth-first-search algorithms by means of network synchronization." *Networks*, 1985.

[8] M. Backes, S. Lorenz, M. Maffei, and K. Pecina, "Anonymous webs of trust," in *PETS'10*.

[9] F. Beato, M. Conti, B. Preneel, and D. Vettore, "Virtualfriendship: Hiding interactions on online social networks," in *CNS'14*.

[10] M. Ben-Or, S. Goldwasser, and A. Wigderson, "Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract)," in *STOC'98*.

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

[12] J. Bonneau, A. Narayanan, A. Miller, J. Clark, J. A. Kroll, and E. W. Felten, "Mixcoin: Anonymity for bitcoin with accountable mixes," in *FC'14*.

[13] R. Canetti, "Universally composable security: A new paradigm for cryptographic protocols," in *FOCS'01*.

[14] ——, "Universally composable signature, certification, and authentication," in *CSFW'04*.

[15] R. Canetti, Y. Lindell, R. Ostrovsky, and A. Sahai, "Universally composable two-party and multi-party secure computation," in *STOC '02*.

[16] O. Catrina and S. de Hoogh, "Improved primitives for secure multiparty integer computation," in *SCN'10*.

[17] L. Cutillo, R. Molva, and T. Strufe, "Safebook: A privacy-preserving online social network leveraging on real-life trust," *IEEE Communications Magazine*, 2009.

[18] I. Damgård, M. Keller, E. Larraia, V. Pastro, P. Scholl, and N. P. Smart, "Practical covertly secure MPC for dishonest majority - or: Breaking the SPDZ limits," in *ESORICS'13*.

[19] I. Damgård and C. Orlandi, "Multiparty computation for dishonest majority: From passive to active security at low cost," in *CRYPTO'10*.

[20] I. Damgård, V. Pastro, N. P. Smart, and S. Zakarias, "Multiparty computation from somewhat homomorphic encryption," in *CRYPTO'12*.

[21] P. Dandekar, A. Goel, R. Govindan, and I. Post, "Liquidity in credit networks: a little trust goes a long way." in *ACM Conference on Electronic Commerce*, 2011.

[22] D. DeFigueiredo and E. T. Barr, "TrustDavis: A Non-Exploitable Online Reputation System," in *E-Commerce Technology'05*.

[23] R. Dingledine, N. Mathewson, and P. Syverson, "Tor: The Second-generation Onion Router," in *USENIX'04*.

[24] Y. Dinitz, "Dinitz's Algorithm: The Original Version and Even's Version," in *Theoretical Computer Science*, 2006.

[25] K. El Defrawy and J. Lampkins, "Founding digital currency on secure computation," in *CCS '14*.

[26] L. R. Ford and D. R. Fulkerson, "Maximal Flow through a Network." *Canadian Journal of Mathematics*, vol. 8, 1954.

[27] R. Fugger, "Money as IOUs in Social Trust Networks & A Proposal for a Decentralized Currency Network Protocol," 2004. [Online]. Available: http://archive.ripple-project.org/decentralizedcurrency.pdf

[28] R. Gennaro, M. O. Rabin, and T. Rabin, "Simplified VSS and fact-track multiparty computations with applications to threshold cryptography," in *PODC'98*.

[29] A. Ghosh, M. Mahdian, D. M. Reeves, D. M. Pennock, and R. Fugger, "Mechanism Design on Trust Networks," in *WINE'07*.

[30] 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.

[31] E. Holley, "Earthport launches distributed ledger hub via Ripple," 2016, http://tinyurl.com/hdygnab.

[32] S. Jahid, S. Nilizadeh, P. Mittal, N. Borisov, and A. Kapadia, "Decent: A decentralized architecture for enforcing privacy in online social networks," in *(PERCOM Workshops) 2012*.

[33] A. M. Kakhki, C. Kliman-Silver, and A. Mislove, "Iolaus: securing online content rating systems," in *WWW'13*.

[34] M. Keller, P. Scholl, and N. P. Smart, "An architecture for practical actively secure mpc with dishonest majority," in *CCS'13*.

[35] A. Kosba, A. Miller, E. Shi, Z. Wen, and C. Papamanthou, "Hawk: The blockchain model of cryptography and privacy-preserving smart contracts," Cryptology ePrint Archive, Report 2015/675, 2015.

[36] Y. Lindell, B. Pinkas, N. P. Smart, and A. Yanai, "Efficient constant round multi-party computation combining BMR and SPDZ," in *CRYPTO'15*.

[37] A. Liu, "Implementing the interledger protocol in ripple," 2015, http://tinyurl.com/gtf6dpj.

[38] ——, "Santander: Distributed Ledger Tech Could Save Banks $20 Billion a Year," 2015, http://tinyurl.com/zwhkoln.

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

[40] G. Maxwell, "Confidential Transactions, Content privacy for Bitcoin transactions," Post on Bitcoin Forum, http://tinyurl.com/zvdr6q2.

[41] G. Mezzour, A. Perrig, V. Gligor, and P. Papadimitratos, "Privacy-Preserving Relationship Path Discovery in Social Networks," in *CANS'09*.

[42] I. Miers, C. Garman, M. Green, and A. D. Rubin, "Zerocoin: Anonymous distributed e-cash from bitcoin," in *S&P '13*.

[43] T. Minkus and K. W. Ross, "I Know What You're Buying: Privacy Breaches on eBay," in *PETS'14*.

[44] A. Mislove, A. Post, P. Druschel, and K. P. Gummadi, "Ostra: Leveraging Trust to Thwart Unwanted Communication," in *NSDI'08*.

[45] A. Mohaisen, N. Hopper, and Y. Kim, "Keep your friends close: Incorporating trust into social network-based Sybil defenses," in *INFOCOM'11*.

[46] A. Mohaisen, H. Tran, A. Chandra, and Y. Kim, "Trustworthy distributed computing on social networks," in *ASIACCS'13*.

[47] P. Moreno-Sanchez, A. Kate, M. Maffei, and K. Pecina, "Privacy preserving payments in credit networks: Enabling trust with privacy in online marketplaces," in *NDSS'15*.

[48] P. Moreno-Sanchez, M. B. Zafar, and A. Kate, "Listening to whispers of ripple: Linking wallets and deanonymizing transactions in the ripple network," in *PETS'16*.

[49] S. Nilizadeh, S. Jahid, P. Mittal, N. Borisov, and A. Kapadia, "Cachet: A decentralized architecture for privacy preserving social networking with caching," in *CoNEXT'12*.

[50] B. Parno, "Bootstrapping trust in a "trusted" platform," in *HotSec'08*.

[51] A. Post, V. Shah, and A. Mislove, "Bazaar: Strengthening User Reputations in Online Marketplaces," in *NSDI'11*.

[52] P. Rizzo, "Japan's SBI Holdings Teams With Ripple to Launch New Company," http://tinyurl.com/jaartry.

[53] ——, "Royal Bank of Canada Reveals Blockchain Trial With Ripple," 2016, http://tinyurl.com/zw48e3c.

[54] T. Ruffing, P. Moreno-Sanchez, and A. Kate, "Coinshuffle: Practical decentralized coin mixing for bitcoin," in *ESORICS'14*.

[55] C.-P. Schnorr, "Efficient signature generation by smart cards," *J. Cryptol.*, 1991.

[56] A. Shamir, "How to share a secret," *Commun. ACM*, 1979.

[57] J. Southurst, "Australia's Commonwealth Bank Latest to Experiment With Ripple," 2015, http://tinyurl.com/pt9gpnv.

[58] D. R. Stinson and R. Strobl, "Provably secure distributed schnorr signatures and a (t, n) threshold scheme for implicit certificates." in *ACISP'01*.

[59] C. Tryfonopoulos, P. Raftopoulou, V. Setty, and A. Xiros, "Towards content-based publish/subscribe for distributed social networks," in *DEBS'15*.

[60] P. F. Tsuchiya, "The Landmark Hierarchy: A New Hierarchy for Routing in Very Large Networks," *SIGCOMM'88*.

[61] L. Valenta and B. Rowan, "Blindcoin: Blinded, accountable mixes for bitcoin," in *FC 2015*.

[62] B. Viswanath, M. Mondal, K. P. Gummadi, A. Mislove, and A. Post, "Canal: Scaling Social Network-based Sybil Tolerance Schemes," in *EuroSys '12*.

[63] D. J. Wu, J. Zimmerman, J. Planul, and J. C. Mitchell, "Privacy-preserving shortest path computation," in *NDSS'16*.

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

APPENDIX

A. Motivations for the Design of the Ideal Functionality

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 $\mathcal{F}_{\mathsf{PAY}}$ 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: $\mathcal{F}_{\mathsf{PAY}}$ 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.

B. Security Analysis

In the following we show that SilentWhispers UC-realizes the ideal functionality $\mathcal{F}_{\mathsf{CN}}$ as described in Section IV, thereby proving Theorem 1. We include a pictorial description of the interactions of the various functionalities in Fig. 7.

Proof: In order to prove that SilentWhispers UC-realizes $\mathcal{F}_{\mathsf{CN}}$ we shall describe an efficient simulator $\mathcal{S}$ such that for any PPT adversary $\mathcal{A}$ and for any environment $\mathcal{E}$, $\mathcal{E}$ cannot distinguish the interaction with $\mathcal{A}$ and the real protocol from the interaction with $\mathcal{S}$ and $\mathcal{F}_{\mathsf{CN}}$. Our simulator internally runs

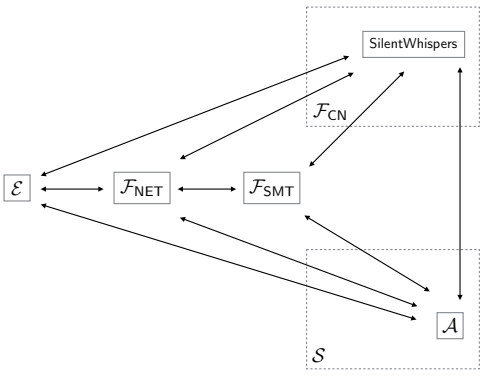

Fig. 7: Overview of the interactions in the ideal world

$\mathcal{A}$ and simulates the inputs that $\mathcal{A}$ is expecting in the real protocol execution. The crucial observation of the proof is that the functionality $\mathcal{F}_{\mathsf{CN}}$ provides $\mathcal{S}$ with all the necessary information to simulate a honest run of the protocol. We prove the indistinguishability by defining a series of neighboring games that are pairwise indistinguishable by any PPT $\mathcal{A}$.

Game 0. Game_0 resembles the original settings in which the environment $\mathcal{E}$ interacts directly with the protocol described in Section V-B and the attacker $\mathcal{A}$.

Game 1. In Game_1 the simulator $\mathcal{S}$ runs internally the attacker and simulates the secure channel functionality $\mathcal{F}_{\mathsf{SMT}}$ and the synchronous network functionality $\mathcal{F}_{\mathsf{NET}}$.

Lemma 1: For all PPT adversaries $\mathcal{A}$ and for all PPT environments $\mathcal{E}$, there exists a simulator $\mathcal{S}$ such that

$$\mathsf{EXEC}_{\mathsf{Game}_0,\mathcal{E}} \approx \mathsf{EXEC}_{\mathsf{Game}_1,\mathcal{E}}.$$

The proof for the lemma follows directly from the realization of the functionalities described above: $\mathcal{S}$ simply executes the simulators provided by the security definitions of $\mathcal{F}_{\mathsf{SMT}}$ and $\mathcal{F}_{\mathsf{NET}}$. For further details, we refer to [13].

Game 2. In Game_2 the simulator $\mathcal{S}$ substitutes the signatures and verification keys of the honest parties with the appropriate outputs of $\mathcal{F}_{\mathsf{SIG}}$.

Lemma 2: For all PPT adversaries $\mathcal{A}$ and for all PPT environments $\mathcal{E}$, there exists a simulator $\mathcal{S}$ such that

$$\mathsf{EXEC}_{\mathsf{Game}_1,\mathcal{E}} \approx \mathsf{EXEC}_{\mathsf{Game}_2,\mathcal{E}}.$$

In order to prove our claim, we describe in the following the modifications that we apply to the simulator $\mathcal{S}$ in the various phases of the protocol. Throughout the discussion we refer to the ideal functionality for digital signatures (described in [14]) as $\mathcal{F}_{\mathsf{SIG}}$ and to the simulator for the specific digital signature scheme as $\mathcal{S}_{\mathsf{SIGN}}$. We shall note that for all digital signatures that UC-realize $\mathcal{F}_{\mathsf{SIG}}$, such an efficient $\mathcal{S}_{\mathsf{SIG}}$ must exist.

System Setup: At the very beginning of the simulation, the system is set up by fixing the landmarks and the initial users of the network. In both cases $\mathcal{S}$ queries the KeyGen interface offered by $\mathcal{F}_{\mathsf{SIG}}$ and simulates a one-time key genereation running $\mathcal{S}_{\mathsf{SIG}}$ for the nodes (including landmarks) that $\mathcal{A}$ is interacting with. Note that nodes can dynamically join the network, $\mathcal{S}$ handles these cases by performing 'on the fly' the same procedure as described before. If a user or a landmark gets corrupted by $\mathcal{A}$, $\mathcal{S}$ corrupts the corresponding identity created in $\mathcal{F}_{\mathsf{SIG}}$.

Link Setup: The adversary $\mathcal{A}$ can either issue requests to create links with some honest user ID_u (from some specific corrupted node $\mathsf{ID}_{\mathcal{A}}$) or receive incoming requests from honest parties. We note that links among corrupted users are handled locally by $\mathcal{A}$ and thus they do not need to be simulated. We assume all the request to be done explicitly, so in the former case the simulator $\mathcal{S}$ simply forwards the request to $\mathcal{F}_{\mathsf{CN}}$ by sending $(\mathsf{ID}_{\mathcal{A}}, \mathsf{ID}_u, \mathsf{val})$, assuming that the request from $\mathcal{A}$ is well-formed. If $\mathcal{F}_{\mathsf{CN}}$ notifies the acceptance then $\mathcal{S}$ simulates Sign via $\mathcal{S}_{\mathsf{SIG}}$ on $(\mathsf{settled}||\mathsf{vk}^*_{\mathsf{ID}_{\mathcal{A}}}||\mathsf{vk}^*_{\mathsf{ID}_u}||\mathsf{val}||\mathsf{epoch})$, while in the ideal world $\mathcal{S}$ queries the ideal functionality $\mathcal{F}_{\mathsf{SIG}}$ on the appropriate interface. Otherwise $\mathcal{S}$ notifies $\mathcal{A}$ of the rejection.

The latter case is analogous except that $\mathcal{S}$ forwards the request to the adversary and, depending on the response, behaves as specified above.

Update Link: The simulation works as the Link Setup (with obvious modifications on the values to sign) except that, in case of a request from $\mathcal{A}$ to modify the value with a node ID_u, $\mathcal{S}$ checks whether there exists already a link between $\mathcal{A}$ and ID_u before forwarding the request to $\mathcal{F}_{\mathsf{CN}}$.

Landmark Routing: If LM is corrupted, $\mathcal{S}$ receives two tuples of the form $(u_1, \ldots, u_m)$. $\mathcal{S}$ then simulates the notification of LM to each of these users that have a direct link with LM in the arborescence (or anti-arborescence, respectively). Every message is signed with the key of LM by running $\mathcal{S}_{\mathsf{SIG}}$ and querying the Sign interface of $\mathcal{F}_{\mathsf{SIG}}$ in the ideal world. The transcript of this simulation is handed over to $\mathcal{A}$. For each corrupted node $u_{\mathcal{A}}$ involved in the routing, $\mathcal{S}$ receives in the ideal world the tuple $(\mathsf{sid}, h, u_{\mathsf{p}})$; $\mathcal{S}$ forwards to $\mathcal{A}$ the message h. If $\mathcal{A}$ does not output any well-formed message to any honest node that he is connected to, $\mathcal{S}$ returns $(\perp, \mathsf{sid})$ to $\mathcal{F}_{\mathsf{CN}}$. On the other hand, if $\mathcal{A}$ communicates a certain h' to an honest node u' from a corrupted node $u'_{\mathcal{A}}$, then the simulator behaves as follows: first it creates a chain of $(h - h' - 1)$ many corrupted nodes $(u_1, \ldots, u_{h-h'-1})$ that connects $u_{\mathcal{A}}$ to $u'_{\mathcal{A}}$ where each link has the dummy value ∞, that indicates an unbounded amount of credit between two nodes. Then $\mathcal{S}$ (interactively) sends $(u_1, \mathsf{sid}), \ldots, (u_{h-h'-1}, \mathsf{sid}), (u', \mathsf{sid})$ to $\mathcal{F}_{\mathsf{CN}}$. The functionality stores locally the derived routing trees.

Transaction - Phase 1: $\mathcal{S}$ gets notified of the beginning of a honest transaction by the message $(\mathsf{Txid}, \mathsf{ID}_{\mathsf{LM}}, u)$ from $\mathcal{F}_{\mathsf{CN}}$, in response it samples a new key pair using $\mathcal{S}_{\mathsf{SIG}}$ and querying $\mathcal{F}_{\mathsf{SIG}}$ in the ideal world. $\mathcal{S}$ simulates a signature on the freshly sampled verification key for u with the long-term signature key of u. In case the adversary sends a correctly-formed message to a certain honest node u', then $\mathcal{S}$ forwards $(u', \mathsf{Txid}, \mathsf{ID}_{\mathsf{LM}})$ to $\mathcal{F}_{\mathsf{CN}}$, or $(\top, \mathsf{Txid}, \mathsf{ID}_{\mathsf{LM}})$ if u' is the correct node defined by the routing tree; otherwise it sends $(\perp, \mathsf{Txid}, \mathsf{ID}_{\mathsf{LM}})$. Since $\mathcal{S}$ knows the number of hops that separate any corrupted node to LM, it is easy to pad the signature chain with the appropriate number freshly generated keys, until the max length is reached. If the transaction starts from $\mathcal{A}$, then $\mathcal{S}$ must notify the ideal functionality $\mathcal{F}_{\mathsf{CN}}$ of the beginning of a transaction. We shall note that $\mathcal{A}$ will provide the appropriate Txid to the landmark (which is impersonated by $\mathcal{S}$) but the Sdr and the Rvr of the transaction are not necessarily known to $\mathcal{S}$. Therefore $\mathcal{S}$ must wait until $\mathcal{A}$ tries to communicate a valid tuple $(\sigma_{\text{child}}, \mathsf{vk}_{\text{child}})$ from a certain corrupted node $u_{\mathcal{A}}$ on the path from Sdr to LM and one on the path from Rvr to LM from some $u'_{\mathcal{A}}$. Note that every transaction must pass through the LM so $\mathcal{S}$ knows the chain of keys (one of which is LM's key). $\mathcal{S}$ notifies $\mathcal{F}_{\mathsf{CN}}$ of the transaction with $(u_{\mathcal{A}}, u'_{\mathcal{A}}, \mathsf{Txid}, \mathsf{ID}_{\mathsf{LM}})$. In order to validate the transaction, $\mathcal{S}$ must also receive from the adversary a signed set of shares $([\![s_1]\!], \ldots, [\![s_{|\mathsf{LM}|}]\!])$ from the corrupted nodes in the path. $\mathcal{S}$ simulates the interaction with the neighbor nodes as specified by the real world protocol (aborting in case of unexpected behavior from $\mathcal{A}$) and reflecting the routing choices of $\mathcal{A}$ in the ideal world. Additionally, $\mathcal{S}$ collects all of the shares sent by the corrupted nodes and verifies their validity. If some LM is corrupted by $\mathcal{A}$, then $\mathcal{S}$ must also provide the adversary with the chain of verification keys (including the honest nodes) and a subset of the shares sent by the honest nodes. As $\mathcal{S}$ knows the value h that separates each corrupted node from LM, it can sample the appropriate number of verification keys of the intermediate honest nodes querying $\mathcal{F}_{\mathsf{SIG}}$ and simulate the signatures on the appropriate values with $\mathcal{S}_{\mathsf{SIG}}$. For the moment we assume the shares to be the ones sent by the honest nodes.

Transaction - Phase 2: Since the shared of the values are generated by the honest users, $\mathcal{S}$ can simply forwards them (together with the transcript of the interactive protocol among the landmarks) to the adversary, in case the landmark is corrupted. In case the transaction was started by $\mathcal{A}$, then $\mathcal{S}$ is notified by $\mathcal{F}_{\mathsf{CN}}$ with v_{LM}, which denotes the capacity of the path between Sdr and Rvr. As the shared values come from the honest nodes, we note that the value v_{LM} is always consistent with the computation provided to the adversary.

Transaction - Phase 3: In case the transaction was initiated by $\mathcal{A}$, he can send out a signed message containing a value x_{LM} for a specific landmark LM. If the signature verifies (according to the interface provided by $\mathcal{F}_{\mathsf{SIG}}$), $\mathcal{S}$ forwards to $\mathcal{F}_{\mathsf{CN}}$ the message $(\mathsf{ID}_{\mathsf{LM}}, x_{\mathsf{LM}}, \mathsf{Txid})$, who informs the nodes of the corresponding path. $\mathcal{S}$ needs to simulate messages from honest nodes to the corrupted nodes along the path: if $\mathcal{F}_{\mathsf{CN}}$ aborts the transaction, then $\mathcal{S}$ reflects the abort in the real world simulation, otherwise it follows the protocol impersonating the nodes that the adversary is connected to. If at any point in time $\mathcal{A}$ times out or sends some invalid message, $\mathcal{S}$ sends to $\mathcal{F}_{\mathsf{CN}}$ the tuple $(\perp, \mathsf{ID}_{\mathsf{LM}}, \mathsf{Txid})$, otherwise he forwards $(\mathsf{accept}, \mathsf{ID}_{\mathsf{LM}}, \mathsf{Txid})$. In case Sdr is not corrupted, then $\mathcal{S}$ gets the message $(\mathsf{ID}_{\mathsf{LM}}, x_{\mathsf{LM}}, \mathsf{Txid})$ from $\mathcal{F}_{\mathsf{CN}}$ and it simulates the inputs for the corrupted nodes along the path as specified above. Note that $\mathcal{S}$ can impersonate Sdr since it initiate a fresh verification key at the beginning of the payment. In the last step of the protocol again $\mathcal{S}$ simulates the intermediate honest nodes in the path by following the real world protocol. If the adversary times out or sends some invalid message to an honest node, $\mathcal{S}$ forwards $(\perp, \mathsf{Txid})$ to $\mathcal{F}_{\mathsf{CN}}$ that concludes the execution of the payment. Otherwise $\mathcal{S}$ sends $(\mathsf{accept}, \mathsf{Txid})$ for every corrupted node. Eventual $\perp$ messages from $\mathcal{F}_{\mathsf{CN}}$ results in $\mathcal{S}$ aborting the simulation of the transaction.

Test Credit: The simulation works exactly as the first two phases of the Transaction protocol.

Accountability: The simulator runs the real world protocol with the exception that the verification of the signatures is done by querying the interface provided by the ideal functionality $\mathcal{F}_{\mathsf{SIG}}$. It is important to observe that, when at least one honest node is involved, the output of $\mathcal{S}$ is always identical to the one of $\mathcal{F}_{\mathsf{ACC}}$: since the adversary cannot forge signatures of honest nodes, any valid link state of a honest node always corresponds to the output of an update link or a transaction protocol. Like $\mathcal{F}_{\mathsf{ACC}}$, $\mathcal{S}$ considers as valid the most updated link state (the one with the latest timestamp).

We first notice that the simulator $\mathcal{S}$ is efficient as it executes only polynomially-bounded algorithms together with a set of basic algebraic operations. We further observe that $\mathcal{S}$ reflects the protocol outcomes and aborts of the ideal functionality in the simulation of the real protocol. Therefore, since $\mathcal{S}_{\mathsf{SIG}}$ produces inputs indistinguishable from the real execution of the digital signature scheme, we have that Game_1 and Game_2

are computationally indistinguishable to the eyes of $\mathcal{A}$. As $\mathcal{A}$ cannot forge messages for non-corrupted node, it is easy to see that any attempt of the adversary to cheat results in an abort both in the ideal world and in the real protocol. It follows that any behavior of the adversary in the real protocol is reflected by the $\mathcal{S}$ in the ideal functionality. Therefore it is infeasible for any environment $\mathcal{E}$ to distinguish the ensembles $\mathsf{EXEC}_{\mathsf{Game}_1,\mathcal{E}}$ and $\mathsf{EXEC}_{\mathsf{Game}_2,\mathcal{E}}$ with probability greater than negligible.

Game 3. In Game_3 the simulator $\mathcal{S}$ does not receive any share from the honest parties but it still has to provide the attacker with simulated values for all of the corrupted landmarks.

Lemma 3: For all PPT adversaries $\mathcal{A}$ and for all PPT environments $\mathcal{E}$, there exists a simulator $\mathcal{S}$ such that

$$\mathsf{EXEC}_{\mathsf{Game}_2,\mathcal{E}} \approx \mathsf{EXEC}_{\mathsf{Game}_3,\mathcal{E}}.$$

In the following we describe the modifications to the simulation of $\mathcal{S}$.

Transaction - Phase 1: The simulator $\mathcal{S}$ samples the shares of the honest nodes as values distributed uniformly at random from the appropriate domain.

Transaction - Phase 2: For each corrupted LM, $\mathcal{S}$ honestly performs the multi-party computation protocol over the shares that are known to the adversary from the previous phase. If the protocol requires some communication of shares from honest landmarks, their messages are simulated with random values in the appropriate domain. $\mathcal{S}$ hands over the transcript of this execution to $\mathcal{A}$. Assuming n ($< |\mathsf{LM}|/2$) many corrupted landmarks, we denote by $[\![s_{i,\min}]\!]$ the resulting shares of the computation for each $i \in \{1,\ldots,n\}$. In case the transaction was started by $\mathcal{A}$, then $\mathcal{S}$ is notified by $\mathcal{F}_{\mathsf{CN}}$ with v_{LM}, which denotes the capacity of the path between Sdr and Rvr. $\mathcal{S}$ computes $j \in \{1,\ldots,|\mathsf{LM}|-n\}$ many shares $[\![s_{j,\min}]\!]$ such that the set $([\![s_{1,\min}]\!],\ldots,[\![s_{|\mathsf{LM}|,\min}]\!])$ reconstructs to v_{LM} and sends each of these value to $\mathcal{A}$ simulating the signature of the appropriate LM.

We argue next about the indistinguishability of the two games. Note that possessing less than $|\mathsf{LM}|/2$ many shares hides the secret value in an information-theoretic sense (by our definition of secret sharing scheme), thus randomly distributed values over the appropriate domain perfectly reproduce the input that the adversary is expecting. We also notice that $\mathcal{S}$, give a set of n ($< |\mathsf{LM}|/2$) shares, must be able to compute the remaining shares such that the whole set reconstructs to some arbitrary value. Using the scheme in [56] this can be efficiently done for any value in the message domain. It follows that the simulation is efficient and that the inputs provided to the adversary in Game_3 are statistically close to the inputs provided in Game_2. Combining the previous lemmas we have that

$$\mathsf{EXEC}_{\mathsf{Game}_0,\mathcal{E}} \approx \mathsf{EXEC}_{\mathsf{Game}_3,\mathcal{E}}$$

where Game_0 reproduces exactly the interaction of $\mathcal{A}$ with the real protocol, whereas in Game_3 the simulator $\mathcal{S}$ only interacts with ideal functionalities. This concludes our proof. ■

Malicious Landmarks. In the following we outline the proof of security when active corruption of landmarks is considered. As discussed in Section VI-B, we need to upgrade our system with a fully UC-secure SMPC construction, as opposed to the simple computation over secret-shared values. Note that by definition any SMPC protocol must provide the description of a simulator $\mathcal{S}_{\mathsf{SMPC}}$, that simulates the honest parties in the computation of an arbitrary function, while interacting with the ideal functionality. It follows that the first two steps of our proof above stay untouched, while in Game_3 the simulator executes $\mathcal{S}_{\mathsf{SMPC}}$ in interaction with $\mathcal{A}$ to simulate the communication rounds of the computation of the function. Note that in our formal definition (see Section IV) $\mathcal{F}_{\mathsf{CN}}$ is in fact an extended version of the ideal functionality for SMPC, as it locally computes the target function revealing only the output of it to the designated party. Since the distribution of the outputs of $\mathcal{S}_{\mathsf{SMPC}}$ is computationally indistinguishable from the distribution of the outputs of the real protocol, the bound on the success probability of the environment $\mathcal{E}$ carries over. The security guarantees of our protocol are parametric in the SMPC scheme: if the construction is secure only against passive corruption of users, then so it is our construction.

C. Handling offline users

In the following, we detail the protocol to handle offline users.

Protocol 4: Offline protocol.

rout: Routing information of neighboring nodes.
Input: $(\mathsf{sk}_u^*, \mathsf{vk}_u^*)$: Long term keys for user u.
e_i: Credit link between u and neighbor i.

/* User goes offline */
for $i \in ngb(u)$ **do**
  Generate shares $[\![s_1,\ldots,s_m]\!]$ of the val of e_i;
  Create $\mathsf{m} := \mathsf{vk}_u^* || \mathsf{vk}_i^* || [\![s_1,\ldots,s_m]\!] || ts || \mathsf{offline}$ and $\sigma_{u,i} := \mathsf{Sign}(\mathsf{sk}_u^*, \mathsf{m})$ and sends $(\sigma_{u,i}, \mathsf{m})$ to i
  Receive σ_i from i and locally store $\mathsf{st}_{\mathsf{vk}_u^*}, \mathsf{vk}_i := (\sigma_i, \mathsf{m})$
  Generate shares $[\![s'_1,\ldots,s_m]\!]$ of the key sk_u^*
  for $j \in |\mathsf{LM}|$ **do**
    Send to LM_j : $(\mathsf{rout}, s_j, \sigma_{u,i}, \sigma_i, s'_j, \mathsf{vk}_u^*)$

/* User goes online */
for $i \in ngb(u)$ **do**
  for $j \in |\mathsf{LM}|$ **do**
    Query to LM_j and verify: $(s_j, \sigma_{u,i}, \sigma_i)$
    Query to LM_j: s''_j for value e_i and $\mathbf{updates}_{s''_j}$, where s''_j is the original share s_j updated by LM_j
    $tmp := s_j$
    for $update \in |\mathbf{updates}_{s'_j}|$ **do**
      $tmp \leftarrow tmp + update$
    Check whether $tmp = s_j$