Privacy Preserving Payments in Credit Networks

Enabling trust with privacy in online marketplaces

Pedro Moreno-Sanchez
CISPA, Saarland University
pedro@mmci.uni-saarland.de

Aniket Kate
CISPA, Saarland University
aniket@mmci.uni-saarland.de

Matteo Maffei
CISPA, Saarland University
maffei@cs.uni-saarland.de

Kim Pecina
CISPA, Saarland University
pecina@cs.uni-saarland.de

***Abstract*—A credit network models trust between agents in a distributed environment and enables payments between arbitrary pairs of agents. With their flexible design and robustness against intrusion, credit networks form the basis of several Sybil-tolerant social networks, spam-resistant communication protocols, and payment systems. Existing systems, however, expose agents' trust links as well as the existence and volumes of payment transactions, which is considered sensitive information in social environments or in the financial world. This raises a challenging privacy concern, which has largely been ignored by the research on credit networks so far.**

This paper presents PrivPay, the first provably secure privacy-preserving payment protocol for credit networks. The distinguishing feature of PrivPay is the obliviousness of transactions, which entails strong privacy guarantees for payments. PrivPay does not require any trusted third party, maintains a high accuracy of the transactions, and provides an economical solution to network service providers. It is also general-purpose trusted hardware-based solution applicable to all credit network-based systems. We implemented PrivPay and demonstrated its practicality by privately emulating transactions performed in the Ripple payment system over a period of four months.

I. Introduction

Credit networks [10], [16], [23] exemplify a flexible yet robust design for distributed trust through pairwise credit allocations, indicating commitments to possible payments. In credit networks, agents (or users) express trust in each other numerically in terms of the credit they are willing to extend each other. By introducing suitable definitions of payments, credit networks may support a variety of applications [22], [27], [30], [32], [35], [37].

Indeed, several systems based on the concept of credit networks have been proposed in the last few years, such as Bazaar [35], Iolaus [22], Ostra [27], Ripple [37], and SocialCloud [32]. Among these, the Google-backed [19] payment system Ripple [37] is emerging as an economical, fast method for performing financial transactions online. Ripple may serve as a complement to decentralized currency systems like Bitcoin [39], and a few banks have started to use Ripple in online payment systems [24], [40].

NDSS '15, 8-11 February 2015, San Diego, CA, USA
 ISBN 1-891562-38-X
http://dx.doi.org/10.14722/ndss.2015.23284

Despite its promising future, the concept of credit networks is still in an early stage and there is room for improvement. System issues such as liquidity [7], network formation [8], [49] and routing scalability [35], [48] of credit networks have been addressed in the recent literature; however, the important issue of credit networks' privacy has not been thoroughly investigated yet. While employing credit network-based payment systems, businesses and customers strive to ensure the privacy of their credit links and transactions from the prying eyes of competitors, authorities, and even service providers; patients want to protect the privacy of their medical bills; in Sybil-tolerant social networks based on credit networks [48], users naturally demand to keep some of their social links and interactions hidden from others. In general, privacy of credit links and payments is crucial for credit network based systems.

Challenges. Designing a privacy-preserving solution for credit networks is technically challenging. Simple anonymization methods such as the pseudonyms employed in Ripple [38] are ineffective, as all transactions remain linkable to each other and they are susceptible to deanonymization attacks. For instance, Minkus et al. [26] were recently able to successfully identify and reveal highly sensitive information about eBay users by accessing their public profiles in the eBay Feedback System and correlating these with social network profiles on Facebook. In decentralized solutions where only the system users are entrusted with their credit links, the system's availability and efficiency is significantly hampered, as users are not online all the time and service providers cannot perform any transaction without the users. Providing the service provider only with the topological network graph while keeping credit values private still leads to a privacy loss. Besides revealing the transaction partners' pseudonyms, a public topological network graph also opens the system up to correlation attacks that ultimately reveal the partners' real identities [33], [43]. Perturbing the links or their credit values by means of differential privacy techniques [28], [41] would yield stronger privacy guarantees, but this is often unacceptable in payment scenarios as it implies unconsented redistribution of credit. Finally, pre-computing the transitive closure of the network and then accessing it through a data-oblivious protocol is infeasible as the credit network is highly dynamic (e.g., credit links typically get modified with every transaction).

Our Contribution. We present PrivPay, a novel architecture for credit networks that preserves the privacy of transactions, maintains a high rate of transaction accuracy, and provides high performance. The distinguishing feature of our architecture is a novel data-oblivious algorithm for computing the maximal credit between two agents, without revealing any information

about the credit network, the transaction, or the agents themselves. This algorithm is implemented by employing a minimal secure and verifiable server-side execution environment, such as the IBM 4765 cryptographic co-processor [21]. PrivPay does not introduce significant computational or financial overhead to either the credit network service provider or the users. In particular, we avoid computationally burdensome cryptography at the user ends, which paves the way for deploying PrivPay on mobile devices such as smartphones.

We formalized for the first time the privacy properties of interest for credit networks and prove that PrivPay achieves them. In particular, we demonstrate that no third party, including the service provider, can identify the transaction values or the parties performing the transaction. Furthermore, the network is concealed from both the users and the server.

We have implemented our system in multithreaded C++ code. For our experiments, we have extracted payment transactions from the real-world Ripple payment ledgers from October 2013 until January 2014, conveying a dataset with more than 14,000 users and more than 8,000 transactions. Our experiments show that a payment operation on average can be done in a privacy-preserving manner within 1.5 seconds, while adding a link between two users into the network on average requires only 0.1 seconds. The execution of our data-oblivious algorithm within the universe creator module takes approximately 22 seconds. Several instances of the algorithm, however, can be run in parallel as a background process to enlarge the set of paths used for checking the maximal credit between the transacting agents. Therefore, PrivPay is suitable to deploy as an online real-time payment system.

Organization. The rest of the paper is organized as follows: In Section II, we present an overview of credit networks . Section III defines the problem of privacy-preserving transactions in credit networks. In Section IV, we give an overview of the PrivPay architecture and we present the detailed construction in Section V. In Section VI, we formally analyze the security and privacy goals of the system and in Section VII, we discuss our implementation and performance analysis. Section VIII considers related work and we conclude in Section IX.

II. Background

Credit networks were introduced to leverage pairwise trust among users in order to build transitive trust models that are robust against intrusions [10], [16], [23]. We start our discussion by explaining the concept of credit networks. We then present an overview of existing systems and discuss the routing algorithms for credit networks.

A. Credit Networks

A credit network is a weighted, directed graph $G = (V, E)$, where the set V of vertices represents the set of users and the set E of weighted edges (links) represents the credit links. A weighted, directed edge $(u, v) \in E$ is labeled with a dynamic scalar value α_{uv} indicating the amount of *unconsumed* credit that a user v has extended to a user u. The success of a transaction between two users in a credit network depends upon the availability of enough credit along credit paths connecting those users. Assume that user u wishes to pay β credits to user v and that u and v are connected by the path $u \to u_1 \to \cdots \to u_n \to v$. For their transaction to be successful, every credit link on the path must have a credit value $\alpha \geq \beta$. At the end of a successful transaction, the credit value on every edge on the path from u to v is decreased by β. For example, in the illustrative credit network described in Fig. 1, assume u_2 wants to buy a service worth ten credits from u_1 The payment can be routed using the path $u_2 \to u_3 \to u_4 \to u_1$. Since every edge in the path holds a credit larger than or equal to ten, the payment can be carried out and edges are decreased by ten credits: edge $u_2 \to u_3$ is deleted while edges $u_3 \to u_4$ and $u_4 \to u_1$ are decreased to five and twenty credits, respectively.

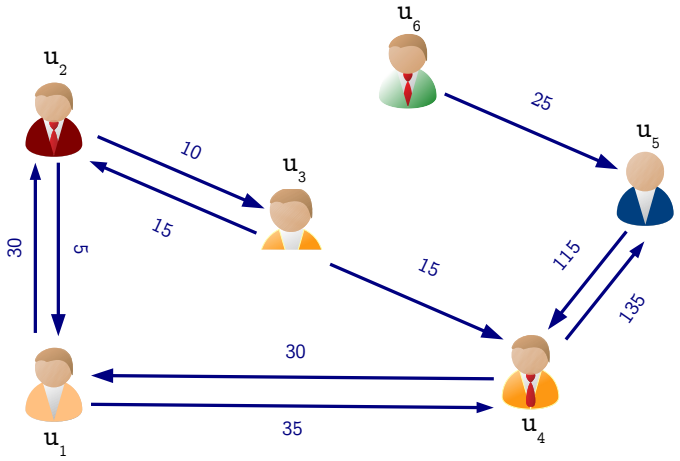

Fig. 1: An illustrative credit network example. Here, u_3 could perform a direct payment for up to 15 credits to u_4 through the credit link between them. However, a payment from u_6 to u_1 should only be routed through the path $u_6 \to u_5 \to u_4 \to u_1$, and it cannot exceed 25 credits.

It is not necessary to find a single path with available credit along each edge; instead, the payment can be split across multiple paths such that the sum of credit available on all paths is larger than or equal to β. For example, in the credit network from Fig. 1, assume now that u_2 wants to pay fifteen credits to u_1. The payment now can be split into two payments with amounts of five and ten credits. The ten-credit payment can be performed as explained earlier, while the five-credit payment is carried out over the direct link shared between u_2 and u_1.

To enable transactions between arbitrary users, the credit network must have good *connectivity* and should maintain *liquidity*, i.e., the network must offer a good choice of paths with sufficient credit along their links. These aspects have been considered in the literature [7], [8], [49].

B. Overview of Existing Systems

Credit networks are designed to tolerate misbehaving (Sybil) agents. In particular, the effect of misbehaving agents in credit networks is bounded by the total credit that they receive from honest agents and localized only to agents that extend such credit [7]. Moreover, these features are independent of the number of agents an adversary may introduce into the system. As a result, several credit networks have been proposed in the last half decade [22], [27], [30], [32], [35], [37].

In these systems, the interpretation of credit associated with links as well as dynamics of the credit flow entirely

depend upon the application scenario. For example, in the Ostra spam-resistant communication system [27], the weight of a link indicates the number of messages allowed from a user to another user. When a message is sent, one unit of weight is removed from the link connecting the sender to the receiver, while it is added to the link in the opposite direction. In the Ripple payment system [37], the weight of a link defines the amount of IOweYou credit offered by a sender to a receiver . To carry out a payment, the weight along the path between sender and receiver is decreased by the payment amount; however, nothing changes in the opposite direction. We discuss state-of-the-art credit network-based systems in Appendix A.

In theory, it is possible to realize a credit network in a decentralized manner; however, as observed in the literature, the decentralized approach is severely restricted in practice in terms of maintaining liveness (or availability) [14], [34].

As a result, none of the proposed or deployed credit networks features a decentralized design, relying instead on a trusted service provider that maintains the credit network by executing valid (i.e., policy-compliant) user requests. A system user generally holds some kind of credential (such as a password or a public-private key pair) for ensuring the authenticity of their requests, and every transaction is usually associated with a small fee to reward the service provider for its service and to mitigate possible DoS attacks. In some systems such as Ripple [37], the complete credit network and transaction ledgers are also published online. In this work, we protect privacy for users transactions on such centralized credit networks.

C. Routing in Credit Networks

A common, prominent task in a credit network is to determine credit routes between the sender and the receiver. Ghosh et al. [16] have shown that the problem of maximizing the possible transactions (which they term as social welfare) in a credit network is NP-hard. Existing credit networks instead consider one transaction at a time and employ the maximum flow approach [15] to check the available credit among all possible paths between sender and receiver. However, the most efficient known max-flow algorithms run in $O(V^3)$ [17] or $O(V^2 log(E))$ [13] time. For this reason, recent work explored the possibility to efficiently calculate only a subset of all possible paths between sender and receiver, thereby underestimating the available credit. The idea of this algorithm, called landmark routing [47], is to calculate a path between sender and receiver through an intermediary node called a *landmark*. As demonstrated by Viswanath et al. in the Canal credit network [48], landmark routing performs much better in large networks than the max-flow approach, with an accuracy loss of only 5%. Canal is split into two processes:

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

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

III. Problem Definition

In this section we formalize the concept of a credit network and the correctness of the underlying operations. We then characterize the expected privacy and system properties.

Let λ represent the security parameter. Let $poly(\cdot)$ and $\nu(\cdot)$ represent respectively a polynomial function and a negligible function parametrized by λ. We assume that the adversary is computationally bounded by the security parameter λ.

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

- setup *$(1^\lambda) \rightarrow params$. On input of a security parameter, output a set of public parameters $params$.*
- pay *$(u_1, u_2, v) \rightarrow \{0, 1\}$. On input of two user identifiers $u_1, u_2 \in V$ and the credit value v, if the payment is approved by the two involved users and if there exists enough credit flow between u_1 and u_2, perform a payment from u_1 to u_2 of value v and return 1. Otherwise, return 0.*
- chgLink *$(u_1, u_2, v) \rightarrow \{0, 1\}$. On input of two user identifiers u_1, u_2 and a credit value v, if u_1 approves the operation, modify the link $u_1 \rightarrow u_2 \in E$ by v and return 1. Otherwise, return 0.*
- test *$(u_1, u_2) \rightarrow v$. On input of two user identifiers u_1, u_2, if both users approve the operation, return the available credit flow between u_1 and u_2.*
- testLink *$(u_1, u_2) \rightarrow v$. On input of two user identifiers u_1, u_2, if one of the users approves the operation, return the credit available in the link $u_1 \rightarrow u_2$.*

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

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

A. Privacy Goals

We characterize two fundamental privacy properties for transactions in a credit network, namely, value privacy and receiver privacy.

Intuitively, we say that a credit network maintains *value privacy* if the adversary cannot determine the value of a

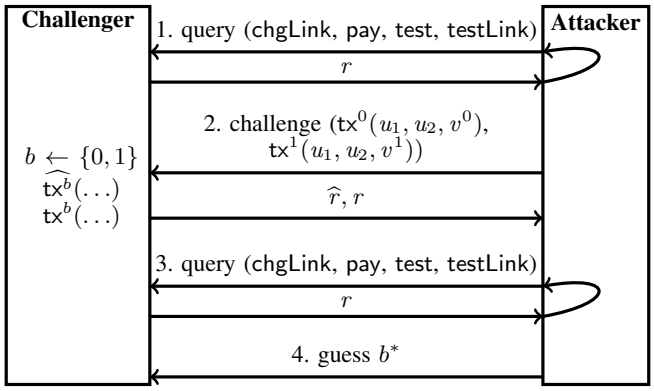

Fig. 2: Cryptographic game for value privacy.

transaction between two non-compromised users. We say that a credit network maintains *receiver privacy* if the adversary cannot determine the receiver of a transaction, as long as this is issued by a non-compromised sender. We formalize these two privacy definitions as cryptographic games.

$\mathsf{Exp}_A^{\mathsf{TxValPriv}}(1^\lambda)$ denotes the cryptographic game for value privacy, which is visualized in Fig. 2. The game consists of an interactive protocol between a challenger and an attacker, where the challenger maintains the credit network in its internal memory and exports the credit network operations from Definition 1 to the adversary in order to give him full control over the network.

This cryptographic game comprises the following phases: *query*, *challenge*, *query*, and *guess*. In the first query phase, the attacker is allowed to create/change links in the network (chgLink), perform payments (pay), test the credit available between users (test) and test the credit available in any link (testLink). The challenger executes these queries and returns the result r to the attacker. In other words, the attacker may setup the network and learn the credit on each edge. This adversarial model is strong but arguably realistic, since the attacker may always extend credit to honest users and, by social engineering attacks, convince an honest user to extend credit to him. At this point, the attacker may perform transactions between two compromised nodes that are connected through honest nodes, thereby indirectly testing the credit on edges connecting honest nodes.

The adversary then enters in the challenge by submitting two transactions of the same type (either pay or chgLink): $\mathsf{tx}^0(u_1, u_2, v^0)$ and $\mathsf{tx}^1(u_1, u_2, v^1)$. Since the attacker knows the network before the challenge and may know the one

challenge: $\mathsf{tx}^0 := \mathsf{pay}(u_1, u_2, v^0)$ - $\mathsf{tx}^1 := \mathsf{pay}(u_1, u_2, v^1)$				
tx^0	tx^1	r	$\widehat{\mathsf{tx}^0}$	$\widehat{\mathsf{tx}^1}$
×	×	×	-	
×	√	×	$\mathsf{pay}(u_1, u_2, (v' - v^1) + 1)$	
√	×	√	$\mathsf{chgLink}(u_1, u_2, 0)$	$\mathsf{chgLink}(u_1, u_2, (v^1 - v^0))$
√	√	√	$\mathsf{chgLink}(u_1, u_2, 0)$	$\mathsf{chgLink}(u_1, u_2, (v^1 - v^0))$*

challenge: $\mathsf{tx}^0 := \mathsf{chgLink}(u_1, u_2, v^0)$ - $\mathsf{tx}^1 := \mathsf{chgLink}(u_1, u_2, v^1)$				
tx^0	tx^1	r	$\widehat{\mathsf{tx}^0}$	$\widehat{\mathsf{tx}^1}$
√	√	√	$\mathsf{chgLink}(u_1, u_2, (v^1 - v^0))$*	$\mathsf{chgLink}(u_1, u_2, 0)$

Here, $v' = \mathsf{test}(u_1, u_2)$. Without loss of generality, $v^1 > v^0$ for the balancing transactions marked with *.

TABLE I: Balancing transaction for value privacy game.

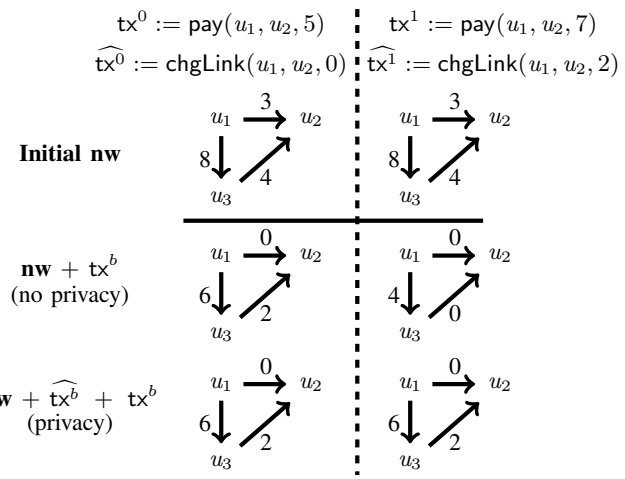

Fig. 3: Example of challenge phase for value privacy game.

thereafter, we need to assume a source of uncertainty in order to prove any meaningful privacy property. In particular, we assume that the adversary does not know the credit on the edge connecting the two honest nodes involved in the transaction. For simplicity, we consider a direct edge, but the same definition and proof would hold true for an honest path (i.e., composed only of honest nodes). This assumption is justified by the fact that the attacker may indirectly try to learn information about the value of an edge between two honest nodes by issuing repeated transactions between compromised nodes connected to such honest nodes, but it cannot be sure that the derived value is the current one, since the two parties can indeed extend credit to each other without being immediately detected by exploiting a direct edge or a honest path.

The idea is that the attacker should not be able to find out whether $\mathsf{tx}^0(u_1, u_2, v^0)$ has been executed in an environment where a transaction $\widehat{\mathsf{tx}^0}$ has been previously executed on the honest edge between u_1 and u_2, or $\mathsf{tx}^1(u_1, u_2, v^1)$ has been executed in an environment where $\widehat{\mathsf{tx}^1}$ has been previously executed on that edge. In other words, the existence of a *balancing* transaction on the honest edge suffices to achieve privacy.

This definition is inspired by the vote privacy definitions of e-voting protocols [11], where the attacker controls all voters except for two, the voter to be protected and another one, called the balancing voter, the uncertainty of whose vote is required to achieve privacy. The challenger flips a bit b and in one case the voter whose vote is supposed to be protected votes for the candidate of her own choice and the balancing voter votes for the candidate supported by the attacker, while in the other case the votes are swapped.

Here the role of the balancing voter is played by the balancing transaction $\widehat{\mathsf{tx}^b}$ on the honest edge. In particular, the challenger sets the response r to successful if tx^0 can be carried out in the current credit network, and to unsuccessful if tx^0 cannot be carried out. Then the challenger randomly chooses a bit b, and picks a *balancing* transaction $\widehat{\mathsf{tx}^b}$ according to Table I. Notice that the illustrated balancing transactions are just examples of possible balancing transactions: in fact, others

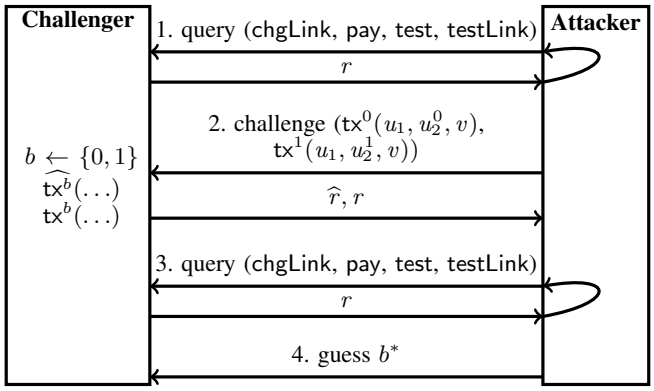

Fig. 4: Cryptographic game for receiver privacy.

are possible, even more if we allow balancing transactions after the challenge. Intuitively, this means that the assumptions under which privacy can be guaranteed are in fact weaker than the ones captured by the balancing transactions in Table I.

The challenger executes $\widehat{\mathsf{tx}^b}$ and tx^b sequentially and sends the respective result $\widehat{r}$ and r to the attacker. Intuitively, the sequences of transactions $\widehat{\mathsf{tx}^0}; \mathsf{tx}^0$ and $\widehat{\mathsf{tx}^1}; \mathsf{tx}^1$ lead to the same network, and are thus indistinguishable. For instance, the first line of Table I says that if both tx^0 and tx^1 fail, then there is no need for balancing transactions. The second line says that if the payment transaction tx^0 would fail on the network known to the attacker, tx^1 would succeed, and the attacker learns that the transaction failed, then the adversary cannot tell whether tx^0 failed but the credit between u_1 and u_2 has been decreased by $v' - v^1$ or the credit between u_1 and u_2 has been decreased by the same amount and then tx^1 failed. The fourth case in Table I, where both payment transactions tx^0 and tx^1 are successfully performed, has been illustrated in Fig. 3. The other lines can be read in a similar manner.

Finally, the attacker can again query the challenger similarly to the first query phase, and finally outputs b^* as his guess of the bit b.

Definition 2 (Value privacy)**.** *A credit network satisfies value privacy if every probabilistic polynomial-time adversary A has negligible advantage in the value privacy game $\mathsf{Exp}_A^{\mathsf{TxValPriv}}(1^\lambda)$. We define the adversary's advantage as $|\mathsf{Pr}[\mathsf{Exp}_A^{\mathsf{TxValPriv}}(1^\lambda) = b] - 1/2|$.*

Next, we define receiver privacy as the cryptographic privacy game $\mathsf{Exp}_A^{\mathsf{TxRcvPriv}}(1^\lambda)$ visualized in Fig. 4. This game is similar to the value privacy game except for the balancing transactions (illustrated in Table II) and the fact that, in

challenge: $\mathsf{tx}^0 := \mathsf{pay}(u_1, u_2^0, v)$ - $\mathsf{tx}^1 := \mathsf{pay}(u_1, u_2^1, v)$				
tx^0	tx^1	r	$\widehat{\mathsf{tx}^0}$	$\widehat{\mathsf{tx}^1}$
×	×	×	-	
×	√	×	$\mathsf{pay}(u_1, u_2^1, (v' - v) + 1)$	
√	×	√	$\mathsf{chgLink}(u_2^0, u_2^1, 0)$	$\mathsf{chgLink}(u_2^0, u_2^1, v)$
√	√	√	$\mathsf{chgLink}(u_2^0, u_2^1, 0)$	$\mathsf{chgLink}(u_2^0, u_2^1, v)$
challenge: $\mathsf{tx}^0 := \mathsf{chgLink}(u_1, u_2^0, v)$ - $\mathsf{tx}^1 := \mathsf{chgLink}(u_1, u_2^1, v)$				
tx^0	tx^1	r	$\widehat{\mathsf{tx}^0}$	$\widehat{\mathsf{tx}^1}$
√	√	√	$\mathsf{chgLink}(u_1, u_2^1, v)$	$\mathsf{chgLink}(u_1, u_2^0, v)$

Here, $v' = \mathsf{test}(u_1$ and $u_2^1)$

TABLE II: Balancing transaction for receiver privacy game.

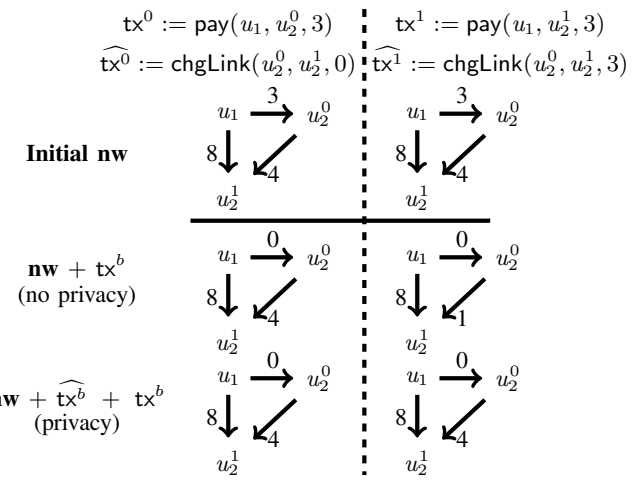

Fig. 5: Example of challenge phase for receiver privacy game.

the challenge phase, the attacker submits two transactions $\mathsf{tx}^0(u_1, u_2^0, v)$ and $\mathsf{tx}^1(u_1, u_2^1, v)$, differing in the receivers. Table II can be read in a similar manner to Table I. An illustrative example is depicted in Fig. 5.

Definition 3 (Receiver privacy)**.** *A credit network satisfies receiver privacy if every probabilistic polynomial-time adversary A has negligible advantage in the receiver privacy game. We define the adversary's advantage as $|\mathsf{Pr}[\mathsf{Exp}_A^{\mathsf{TxRcvPriv}}(1^\lambda) = b] - 1/2|$.*

Notice that, for our definitions, we assume that transactions are executed by the senders and thus we define receiver privacy. It is, however, easily possible to define the complementary sender privacy property if in some credit network setting transactions are executed by the receiver.

B. System Goals

A credit network should further preserve the following system properties.

Performance. Since Internet users are accustomed to real-time online systems that react instantaneously or nearly instantaneously, the response time of a credit network to a transaction request should be small (on the order of a few seconds).

Accuracy. Given the general inefficiency of finding optimal paths in networks, it is inevitable to use approximate, yet accurate, routing algorithms, such as landmark routing. It is expected that a privacy-preserving credit network maintains the same level of accuracy, i.e., achieving privacy does not decrease the accuracy of the system.

Rate limiting. A network design must be able to restrict the number of queries that a (malicious) user issues aiming at reducing the usability for the rest of users in the system.

Generality. Rather than a solution for a particular setting, the envisioned credit network should be general and applicable to many credit network-based systems.

Scalability. A network design must be able to cater to a growing user base without significantly decreasing performance.

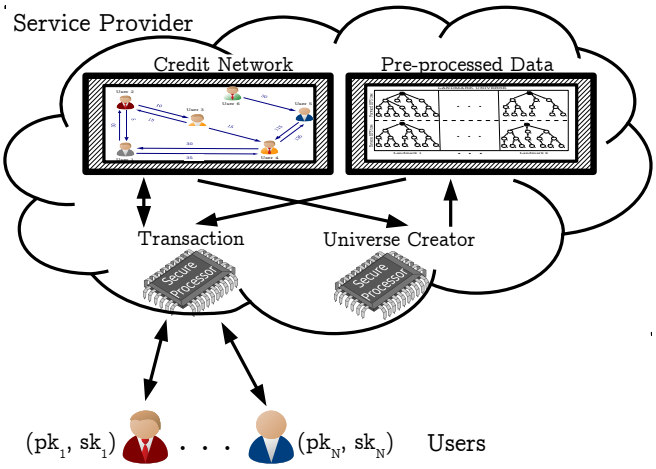

Fig. 6: Overview of PrivPay.

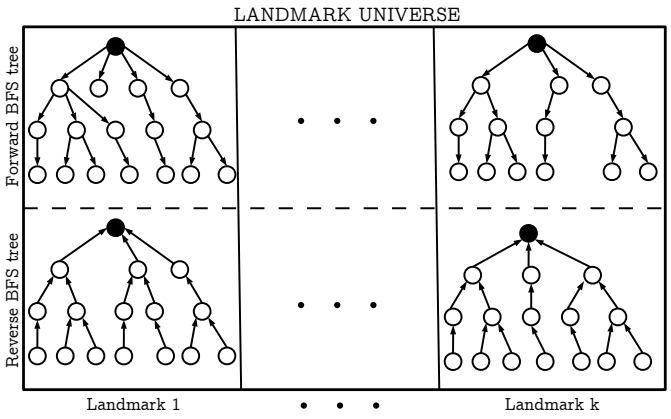

Fig. 7: Output of universe creator module.

C. Threat Model

The service provider is considered to be malicious and, in particular, it can try to tamper with data and deny service to the users, although the latter attack would be economically inconvenient for the server. We assume a trusted execution environment located at the service provider. This environment is considered secure and it is not possible to spy on the computations or read the internal memory. Such an environment may be realized using a trusted platform module or a secure co-processor [3]. Finally, users are considered malicious and may run arbitrary operations with the service provider. We assume a private and authenticated communication channel between user and secure co-processor (as otherwise, our privacy goals are trivially broken).

User anonymity. In the privacy goals, we intentionally do not consider user (in particular sender) anonymity and only protect the receiver and the transacted value. More precisely, we accept that service providers learn the identity of their users, e.g., through the IP address. If user anonymity is desired, users should connect via an anonymous communication network such as Tor [12].

IV. Our Approach

As mentioned earlier, to maintain availability, we aim at a centralized privacy-preserving credit network design. As a result, the fundamental problem is retrieving and updating the credit network information outsourced to an untrusted third party in a privacy-preserving manner. Inspired by other works on private information retrieval and oblivious RAM [1], [3], [4], [25], [50], we solve this problem by relying on a small trusted execution environment, which is in charge of running a novel oblivious algorithm to retrieve and update the status of the credit network without revealing any information about the transaction to the server. In this section, we describe our system in more detail and give an overview of the protocol.

A. System Description

We propose a modular system as depicted in Fig. 6. For the service provider, we require a secure processor (SC) such as the 4765 cryptographic co-processor by IBM [21]. In particular, we require two functionalities from the SC: *secure external storage* and a *programmable execution environment*. The former functionality implements secure storage for sensitive credit network data within the service provider's memory. We rely on it to securely maintain the credit network graph and the auxiliary pre-processed data (i.e., the landmark universe) on the server. The SC internally only stores the cryptographic material necessary to access the encrypted data stored on the server. The latter functionality offers remote code attestation that allows the verification of the code being executed [44]. It is thereby possible to assure that the SC is correctly running the two main modules of our protocol: the universe creator module and the transaction module.

Finally, every user holds a pair of signing keys (pk, sk). The hash of the pk is used as the user's identifier in the credit network. The sk is used by the user to sign the transactions requests before they are sent to the SC. The SC can thereby check that the transactions are issued by authorized parties.

B. Protocol Overview

A user willing to perform a credit operation connects to the service provider and transmits the request. The service provider answers with a successful or an unsuccessful result, depending on the credit network information.

At a high level, the functionality at the service provider is divided into two modules: universe creator module and transaction module. In the following, we present them in more detail and describe how they interact in order to support the credit network operations.

Universe creator module. Following the landmark routing approach [47], the universe creator module uses a new oblivious BFS algorithm to compute paths between every pair of nodes via certain specific nodes called *landmarks*. This results in a landmark universe consisting of pairs of directed graphs: the forward BFS tree (Fig. 7 top), i.e., the tree for the shortest path from the landmark to the nodes, and the reverse BFS tree (Fig. 7 bottom), i.e., the tree for the shortest path from the nodes to the landmark.

In more detail, the universe creator module randomly selects k nodes from the credit network, denoted as landmarks. Given access to the credit network, the universe creator module

executes the oblivious BFS algorithm twice for every landmark in order to calculate the forward BFS tree and the reverse BFS tree. Finally, the universe creator module stores the resulting landmark universe within the pre-processed data storage.

Transaction module. The transaction module receives user requests and answers them with a successful or an unsuccessful result depending on the current state of the credit network. In order to do that, the transaction module employs the routing data computed by the universe creator module. In more detail, upon reception of a user request, the transaction module checks whether the user is allowed to make the request according to a system policy. If the policy is met, the request is processed; otherwise it is rejected. To process a payment request pay *(sdr, rcv, value)*, for all landmarks u, the transaction module computes the shortest paths from the sender *sdr* to the landmark u using the reverse BFS tree in the landmark universe (Fig. 7 bottom), and from u to the receiver *rcv* using the forward BFS tree (Fig. 7 top). Finally, the transaction module prunes overlaps in the paths and calculates the maximum available credit (*max_credit*) from *sdr* to *rcv* along the set of calculated paths. If *max_credit* is larger than or equal to *value*, the available credit in the previously calculated paths from *sdr* to *rcv* is decreased until a total amount of *value* is considered. Otherwise, no change occurs and the user receives an unsuccessful response.

To process a request of the form chgLink *(sdr, rcv, value)*, the transaction module retrieves the link from the sender *sdr* to the receiver *rcv* from the credit network storage, updates its weight to *value* and stores it back.

V. Construction

In this section we present a detailed description of our construction. We first define the necessary building blocks and afterwards describe the core protocol details.

A. Building blocks

ORAM. Assume a scenario where a user wishes to store her data on an untrusted server while preserving its privacy. Solely encrypting the outsourced data does not suffice, as the server can break the privacy by observing the memory access pattern. Oblivious Random Access Model (ORAM) is a construction that allows for completely hiding the access patterns (i.e., read and write operations) from the server [18]. Intuitively, the data is encrypted and the access patterns of any operation are randomized to conceal which data was read or written.

Definition 4 (ORAM security [45])**.** *Let* $\vec{y} := ((op_1, u_1, data_1), (op_2, u_2, data_2), \ldots, (op_M, u_M, data_M))$ *denote a data request sequence of length M, where each* op_i *denotes a* $read(u_i)$ *or a* $write(u_i, data)$ *operation. Specifically,* u_i *denotes the identifier of the block being read or written, and* $data_i$ *denotes the data being written. Let* $A(\vec{y})$ *denote the (possibly randomized) sequence of accesses to the remote storage given the sequence of data requests* $\vec{y}$*. An ORAM construction is said to be secure if for any two data request sequences* $\vec{y}$ *and* $\vec{z}$ *of the same length, their access patterns* $A(\vec{y})$ *and* $A(\vec{z})$ *are computationally indistinguishable.*

We instantiate an ORAM with the Path ORAM protocol [46], since it is compatible with a SC, it is provably secure and efficient, and it requires only a small SC internal storage. Following the study by Stefanov et al. [46], an access to the ORAM has asymptotic costs of $O(\log^2(|V|))$, where $|V|$ is the number of nodes within the credit network. The SC-side storage requirements are $O(|V| \cdot \log(|V|))$.

Data-Oblivious BFS. Due to our use of the landmark routing algorithm, we need to compute shortest paths in the graph. The ObliBFS algorithm (defined in Algorithm 1) is a standard BFS algorithm that is augmented to ensure that it does not leak information about the input graph except its size.

Intuitively, we say an algorithm is a data-oblivious BFS, if the access patterns on the memory depend only on the size (i.e., number of nodes) of the input graph, and not on its structure nor on the weights of the edges. We adapt the definition presented by Blanton et al. [5] to define the security property of ObliBFS formally. In particular, while the Blanton et al. definition takes into account the sequence of instructions and the memory access pattern performed by the algorithm, in our definition we consider only the latter, since, given that the algorithm is run inside the SC, the sequence of instructions is implicitly concealed from the attacker. Therefore, we formally define the security property of ObliBFS as follows:

Definition 5 (Data-oblivious BFS)**.** *Let* G *denote the input graph to a BFS algorithm. Also, let* $A(G)$ *denote the sequence of memory accesses that the algorithm makes. The algorithm is considered data-oblivious if for two inputs graphs* G *and* G' *of equal size, the memory access patterns* $A(G)$ *and* $A(G')$ *are computationally indistinguishable.*

We now describe our ObliBFS algorithm (cf. Algorithm 1). In the initialization phase (lines 1-10), the auxiliary information is initialized as defined in the standard BFS and stored in the auxiliary ORAM.

Within every iteration of the main loop (lines 13-31), the node at the head of the BFS queue is updated as defined in the standard BFS with two main differences: (i) auxiliary and output data is read and written from/to the corresponding ORAMs, and (ii) for every node, a fixed number of adjacent nodes (*MAX_ADJ*) is considered. It is worth noting that even though a node could have fewer neighbours than *MAX_ADJ*, the for loop (lines 20-31) is executed exactly *MAX_ADJ* times. This is done to maintain the obliviousness of the algorithm. In particular, in every iteration, the current node u is set to BLACK (i.e., visited and considered), while the parent and distance information is updated and stored in the output ORAM. Then the nodes adjacent to u are considered. Every node v adjacent to u that has not been visited yet (its color is still WHITE) is accordingly updated. At this point, it is also worth noting that, for every node v, its auxiliary information is read and written back to the auxiliary ORAM. However, only non-visited nodes are updated. This is done in order to maintain the obliviousness of the algorithm.

The ObliBFS algorithm has a computational complexity of $O(|V| \cdot |M| \cdot \log^2(|V|))$, where *M* is the upper bound denoted above as *MAX_ADJ*. Blanton et al. [5] describe a data-oblivious BFS with complexity $O(|V|^2 \cdot \log(|V|))$. The ObliBFS algorithm is tailored for sparse graphs: since credit networks are typically sparse (i.e., $(|M| \cdot \log(|V|) \ll |V|$), our algorithm is better suited for credit networks. Finally, the

Algorithm 1: ObliBFS

Input: o: ORAM storing the credit network graph
s: starting node for BFS
aux: ORAM storing BFS auxiliary information
o': ORAM storing the calculated BFS tree
Result: BFS tree stored in o'

```
1  init_queue(queue q)
2  foreach node i in V do
3      if i == s then
4          color_i = GRAY
5          distance_i = 0
6      else
7          color_i = WHITE
8          distance_i = ∞
9      parent_i = NO_PARENT
10     aux.write(i, (color_i, distance_i, parent_i))
11 iter = 1
12 q.push(s)
13 while iter != |V| do
14     u = q.pop(); iter++
15     aux.read(u, (color_u, distance_u, parent_u))
16     color_u = BLACK
17     aux.write(u, (color_u, distance_u, parent_u))
18     o'.write(u, (u, parent_u, s))
19     o.read(u, adjacents_u)
20     for i in 1 ... MAX_ADJ do
21         if i < adjacents_u.size() then
22             v = adjacents_u[i]
23         else
24             v = 0
25         aux.read(v, (color_v, distance_v, parent_v))
26         if color_v == WHITE then
27             color_v = GRAY
28             parent_v = u
29             distance_v = distance_u + 1
30             q.push(v)
31         aux.write(v, (color_v, distance_v, parent_v))
```

storage size is dominated by the ORAM storage, i.e., the SC-side storage requirement is $O(|V| \cdot \log(|V|)$.

B. Protocol Description

Assumptions. We assume that users can set up a private communication channel with the service provider, and they use it to privately send the transaction information. In practice, such channels can be implemented using the TLS protocol.

We further require that the network graph is stored in the credit network storage and the landmark universe is stored in the pre-processed data storage, both in the form of an ORAM. More specifically, the network graph is stored in two ORAMs, the *fw-graph-oram* and *rvs-graph-oram*: the former contains the network graph itself, and the latter contains the network graph where all the edges are reversed (for computing the reverse BFS trees). The landmark universe is stored in a single ORAM *lm-oram* containing the BFS trees computed by the universe creator module. The necessary keys are managed by the SC. The transaction module and the universe creator module are run within the SC. Both modules interact with the credit network storage and the pre-processed data storage through the respective ORAM interfaces also handled by the SC.

Universe creator module. The purpose of the universe creator module is to select landmarks and to create for every landmark a set of forward and reverse BFS trees.

1) *Selection of landmarks.* The universe creator module randomly selects a set of k random nodes, called landmarks and denoted by L. We leave k as a system parameter.

2) *BFS creation.* For every landmark $u \in L$, the universe creator module executes the ObliBFS algorithm twice, computing the forward BFS tree and the reverse BFS tree. We use a dedicated ORAM to store auxiliary data such as the BFS work queue. The resulting trees are stored in the pre-processed data storage through the *lm-oram* interface.

Transaction module. The purpose of the transaction module is to receive user transactions, perform them, and answer the requesting user according to the credit network state.

Intuitively, we say that an algorithm performs a data-oblivious transaction if the pattern of accessing the memory describing the credit network is independent of the input:

Definition 6 (Data-oblivious transaction)**.** *Let $I := (G, U)$ denote the input to a transaction algorithm, where G denotes the credit network graph and U denotes the auxiliary routing data. Also, let $A(I)$ denote the sequence of memory accesses that the algorithm makes. For two input tuples I and I' with corresponding G and G' of the same size, the transaction algorithm is data-oblivious if the memory access patterns $A(I)$ and $A(I')$ are indistinguishable.*

We next present details of our transaction module.

1) *Transaction Request.* A user sends a transaction request to the SC over the previously established secure channel. The transaction request must be of the form $\mathsf{op}(s_{id}, r_{id}, value)$, where s_{id}, r_{id} respectively contain the sender and receiver's identifier, and $value$ defines the transaction amount.

2) *Policy Check.* The transaction module verifies whether the user is allowed to perform the request according to the system policy. In PrivPay we stipulate a policy conforming to: (i) every operation op must be approved by the corresponding users (see Definition 1); (ii) user i has not exceeded a *threshold* number of operations in a given time period. PrivPay thereby provides rate limiting. If the policy verification succeeds, the transaction module updates the information associated with the user's policy and continues with the next step of the protocol. Otherwise, the transaction module sets the return value tx_{out} to *unsuccessful*, sends it to the user, and stops the processing of the query.

3) *Request Parser.* Depending on the requested operation, the transaction module proceeds as follows:

a) $\mathsf{chgLink}(s_{id}, r_{id}, value)$ modifies the link between s_{id} and r_{id} by *value*. The process updates the *fw-graph-oram* and the *rvs-graph-oram* accordingly, sets the return value tx_{out} to *successful*, and the protocol enters step 7.

b) testLink *(s_{id}, r_{id})* checks the credit available in the link between s_{id} and r_{id}, returns it to the user and closes the private channel.
c) pay *(s_{id}, r_{id}, value)* first checks if the available credit between s_{id} and r_{id} is sufficient for the requested payment, adapts the credit network storage if necessary, and informs the user. We detail the pay operation in steps 4 to 6.
d) test *(s_{id}, r_{id})* checks the available credit between s_{id} and r_{id}. The process is performed similarly to pay, but with two differences: in step 6, the set of paths between s_{id} and r_{id} are not modified; and in step 7, the available credit is sent to the user instead of tx_{out}.

4) *Path Reconstruction.* The transaction module "stitches" all paths between s_{id} and r_{id} using the pre-calculated data by the universe creator module. First, it calculates the set of common landmarks $L_{s_{id}-r_{id}}$ for s_{id} and r_{id}, i.e., the landmarks that are reachable by s_{id} and that can reach r_{id}. Then, it computes the set of paths from s_{id} to r_{id} routed by landmarks contained in $L_{s_{id}-r_{id}}$. At this point, the size of $L_{s_{id}-r_{id}}$ depends on the sender and receiver of the transaction. To preserve obliviousness, we add fake landmarks to $L_{s_{id}-r_{id}}$ resulting in a set $L'_{s_{id}-r_{id}}$ of size *MAX_COMMON_LM*.

In summary, for every landmark $u \in L'_{s_{id}-r_{id}}$, the transaction module computes a path p of the form $s_{id} \rightarrow u \rightarrow r_{id}$, i.e., starting from s_{id}, to r_{id}, via u. The path from u to r_{id} is retrieved from the forward BFS tree; the path from s_{id} to u is retrieved from the reverse BFS tree. At this point, the length of p depends on s_{id}, r_{id} and the landmark u. To preserve obliviousness, we pad p with dummy nodes to make p a fixed size which we denote by *MAX_PATH_LEN*.

Intuitively, the padding on the number of common landmarks and the size of the reconstructed paths fixes the number of ORAM accesses to *MAX_COMMON_LM* times *MAX_PATH_LEN* times. Obliviousness is thereby preserved. This principle is applied to the rest of the steps of the algorithm. Formal details are shown in Section VI-A.

5) *Credit Calculation.* For every path $p \in P$, the transaction module calculates its available credit. The available credit of a path is the smallest weight along the path. The weights are retrieved from *fw-graph-oram*. As before, we pad the execution with dummy steps to conduct *MAX_PATH_LEN* steps per path.

6) *Transaction Result.* The transaction module calculates the total credit available between sender and receiver by summing up the credit of every path calculated in step 5.

If the credit available is sufficient, then we set the return value tx_{out} to successful. Additionally, for each path p_i used in the payment, the links in path p_i are decreased by the value routed through this path. This process is carried out until the transaction value is deducted. Again, we pad the number of operations to maintain obliviousness. If the credit available is not sufficient, we set tx_{out} to unsuccessful. Further, we perform the same number of operations as in the successful case, without changing the credit, to make the unsuccessful case indistinguishable from the successful one.

7) *Transaction Answer.* The transaction module sends the transaction return value tx_{out} to the requesting user and closes the private channel.

Regarding the computational complexity, in the transaction module we have defined four operations: chgLink, testLink, pay and test. Given that chgLink and testLink are simpler, in this analysis we focus only on the pay operation, which has the same complexity as test operation. Hence we consider the operations carried out in steps 4 to 6. Assume that $|L'|$ is the number of maximum common landmarks and $|P|$ is the maximum path length. Step 4 accesses the ORAM $|L'|$ times to compute common landmarks between the sender and the receiver of the payment. Then, for every landmark, the transaction module accesses the ORAM $|P|$ times to stitch together the path. Thus, we perform $O(|L'| \cdot |P| \cdot \log^2(|V|))$ operations overall. Step 5 calculates the available credit for every of the $|L'|$ possible paths. Therefore, it is easy to see that this step takes $O(|L'| \cdot |P| \cdot log^2(|V|))$. Finally, step 6 performs exactly the same number of operations as step 5 to update every path according to the payment result. Therefore, the complexity is the same. In credit networks, the variables $|L'|$ and $|P|$ are typically small. We therefore consider them as constants and obtain a complexity of $O(\log^2(|V|))$.

Regarding the storage complexity, assume that $|L'|$ is the number of maximum common landmarks and $|P|$ is the maximum path length. On the one hand, the transaction module maintains in the clear $|L'|$ sets of paths with length $|P|$, while performing the calculation of the credit available between sender and receiver. Thus, a storage size of $O(|L'| \cdot |P|)$ is required. On the other hand, the transaction module performs ORAM accesses, so a storage size of $O(|V| \cdot \log(|V|))$ is required. Given that $|L'|$ and $|P|$ are small, we consider them as constants and the storage size is dominated by the storage size needed to access ORAM structures (i.e., $O(|V| \cdot \log(|V|))$).

System analysis. PrivPay achieves the goals defined for a credit network. First, PrivPay incurs a small computation overhead. We show performance results in Section VII. Secondly, our study of the asymptotic computational and storage complexity of the protocol modules shows that PrivPay scales well to a large number of users. Thirdly, PrivPay preserves accuracy. In particular, our protocol definition assures that false positives (i.e., a transaction is considered successful although there is not enough credit in the network) never occur. Moreover, PrivPay enforces a policy which checks that transactions are issued by the authorized users and rate limiting is ensured. Finally, thanks to the exported API, PrivPay constitutes a privacy-preserving plugin that can be easily integrated into other credit network-based systems: we show concrete examples in Appendix A.

VI. Security Analysis

In this section, we first analyze the security of the algorithms used in our construction. Then, we argue that PrivPay satisfies the privacy goals defined in Section III-A.

A. Building primitives

PrivPay consists of the universe creator module and the transaction module, which internally use the ObliBFS algorithm and the path stitching algorithm. We now prove that they satisfy the respective security definitions. For the remainder of this paper, we assume that O is a secure ORAM.

Theorem 1 (ObliBFS security)**.** *ObliBFS is a data-oblivious BFS algorithm.*

Proof sketch: In every loop within the algorithm (see Algorithm 1), the number of accesses to O depends only on the size of the graph or on a fixed system parameter. Furthermore, neither of the *if-then-else* branches conditionally accesses an ORAM. Thus, the adversary only notices a set of M accesses $((op_1, \quad u_1, data_1), (op_2, u_2, data_2), \ldots, (op_M, u_M, data_M))$, where op_i are independent of the input graph and M depends only on the graph's size. Therefore, if the adversary is able to distinguish which graph has been chosen, we could use this adversary to break the ORAM definition. ■

Theorem 2 (Transaction module security)**.** *The transaction module is a data-oblivious transaction algorithm.*

Proof sketch: In this proof, we refer to the transaction module paragraph in Section V-B when mentioning algorithm steps. The transaction module performs four types of operations: chgLink, pay, testLink and test. Operations testLink and test perform the same memory access patterns as chgLink and pay correspondingly. Moreover, they do not make any change in the network. Therefore, in the following we prove data-obliviousness for chgLink and pay. Data-obliviousness proof of testLink and test trivially follow from them.

- For a chgLink operation, the algorithm always performs exactly one read and one write operation, i.e., ORAM is accessed a constant number of times.

- In the pay case, the ORAM is accessed exactly *MAX_COMMON_LM* times to retrieve common landmarks. For every potential common landmark, we access the ORAM *MAX_PATH_LEN* times to stitch the path together (see step 4 through 6). At this point, we have potentially stitched one path for every possible common landmark. Thus, we access the ORAM *MAX_COMMON_LM* times *MAX_PATH_LEN* times to determine the available credit (see step 5). Finally, we access the ORAM *MAX_COMMON_LM* times *MAX_PATH_LEN* times to reduce the credit through the paths. If the credit was not sufficient, we still perform the same number of accesses (see step 6).

It follows that the number of ORAM accesses depends only on the system parameters *MAX_COMMON_LM* and *MAX_PATH_LEN*. Therefore, if an adversary is able to distinguish between two transactions, we could use this adversary against the ORAM security property. ■

B. PrivPay privacy

We prove that PrivPay satisfies value privacy and receiver privacy as defined in Section III-A. We start our discussion by instantiating the corresponding cryptographic games.

A cryptographic game consists of a challenger and an attacker. The challenger embodies the trusted execution environment, and thus the SC's internal memory (e.g., ORAM keys and ORAM indexes) such that the memory cannot be observed by the attacker, and the challenger's internal actions are oblivious to the attacker. On the other hand, the attacker embodies the service provider, and provides the challenger with the pointer to his memory region containing the ORAM stores and data for the credit network, and it can observe the challenger's memory accesses when performing the query along with the query's result.

Theorem 3 (Value privacy)**.** *PrivPay provides value privacy as defined in Definition 2.*

Proof sketch: Our proof strategy works as follows: Given the input network state nw known to the adversary and two challenge transactions tx^0 and tx^1, for both values of the choice bit b, we perform the corresponding balancing transaction to obtain an intermediate modified network nw^b_m. We then obtain a network nw'^b by performing the corresponding challenge transaction tx^b over nw^b_m. The results of both the balancing and challenge transactions as well as the network nw'^b are available for the attacker to query; however, notice that the intermediate state nw^b_m is unknown the attacker. Finally, for every case, we argue that the visible changes in nw'^b to nw are indistinguishable for the attacker for both choices of b.

Moreover, in the proofs, when executing an operation of type $\mathsf{chgLink}(u_1, u_2, v)$, if the modification implies a reduction of credit larger than the credit available in the link $u_1 \rightarrow u_2$, the link's credit is set to 0 and the operation returns 1. Furthermore, if the modification implies an increase of credit of the link $u_1 \rightarrow u_2$ larger than an upper bound, the link's credit is set to the upper bound and the operation returns 1. Finally, when executing an operation of type $\mathsf{pay}(u_1, u_2, v)$, we assume that $v > 0$.

We first show our result for challenge transactions of type pay and later for those of type chgLink. Assume that the attacker provides the challenger with two challenge pay transactions: $\mathsf{tx}^0 := \mathsf{pay}(u_1, u_2, v^0)$ and $\mathsf{tx}^1 := \mathsf{pay}(u_1, u_2, v^1)$. To argue indistinguishability, we follow the same case order as in Table I in the challenger definition.

1) tx^0 **and** tx^1 **unsuccessful;** $r :=$ **unsuccessful.** An unsuccessful transaction does not imply any change in the network. Moreover, no balancing transaction is performed in this case. Therefore, for both choices of b, the nw'^b will be exactly the same as nw and thus indistinguishable.

2) tx^0 **unsuccessful;** tx^1 **successful;** $r :=$ **unsuccessful.** Independently of the value of b, the balancing transaction reduces the available credit between u_1 and u_2 in network nw, resulting in a network nw_m such that both tx^0 and tx^1 do not succeed. Thus, for both choices of b, $nw'^b = nw_m$ and thus they are indistinguishable to the attacker.

3) tx^0 **successful;** tx^1 **unsuccessful;** $r :=$ **successful.**

- $b = 0$. The balancing transaction does not imply changes in the network nw. Therefore, $nw^0_m = nw$. Performing tx^0 over nw^0_m now implies a reduction of v^0 credit from the flow between u_1 and u_2, resulting in the network nw'^0.
- $b = 1$. The balancing transaction results in nw^1_m with $v^1 - v^0$ credit added to the link between u_1 and u_2. Performing tx^1 over nw^1_m now implies two actions: (i) the reduction of the extra credit added by the balancing transaction; (ii) reduction of the rest of the credit ($v^1 - (v^1 - v^0) = v^0$) from the flow between u_1 and u_2 to obtain nw'^1.

Independently of the choice of b, the attacker observes that in the network nw'^b there has been a reduction of the flow

between u_1 and u_2 of a credit v^0 with respect to the input network nw. In this case, it is interesting to see that the attacker still learns that at least v^0 credits have been transacted from u_1 to u_2. However, the attacker still does not know if in fact only v^0 credits were transacted, or instead v^1 credits were transacted.

4) tx^0 **successful;** tx^1 **successful;** $r :=$ **successful.** This case follows by the same reasoning as in the previous case, which completes our case analysis for pay.

Assume now that the attacker provides the challenger with two chgLink challenge transactions: $\mathsf{tx}^0 := \mathsf{chgLink}(u_1, u_2, v_0)$ and $\mathsf{tx}^1 := \mathsf{chgLink}(u_1, u_2, v_1)$.

- $b = 0$. The balancing transaction results in a network nw_m^0 with $v^1 - v^0$ added to $u_1 \rightarrow u_2$. Performing tx^0 over nw_m^0 now implies an increment of v^0 credit to $u_1 \rightarrow u_2$. Thus, in the final network nw'^0, $u_1 \rightarrow u_2$ has changed its credit by $v^0 + (v^1 - v^0) = v^1$.
- $b = 1$. The balancing transaction does not imply changes in the network nw. Therefore, $nw_m^1 = nw$. Performing tx^1 over nw_m^1 now implies an increment of v^1 credit to $u_1 \rightarrow u_2$, resulting in the network nw'^1.

Independently of b, the attacker observes that in the network nw'^b there has been an increment of v^1 credits to $u_1 \rightarrow u_2$. However, the attacker still cannot distinguish if the increment has been done in a single transaction or in several transactions. All these cases show that the attacker cannot distinguish the challenge transaction by studying the changes on the credit network. However, the attacker can see the memory access pattern carried out by the challenger when performing both the balancing transaction and the challenge transaction. For this, assume that an attacker A can break the value privacy property on input of the balancing and challenge transactions along with the memory access pattern. One can use such an algorithm A to break the data-oblivious transactions property. However, we prove data-obliviousness of PrivPay transactions in Section VI-A. Hence, the theorem is proved. ■

Theorem 4 (Receiver privacy)**.** *PrivPay provides receiver privacy as defined in Definition 3.*

Proof sketch: In this proof we follow the same strategy as in the proof of Theorem 3. Assume that the attacker provides the challenger with two pay challenge transactions: $\mathsf{tx}^0 := \mathsf{pay}(u_1, u_2^0, v)$ and $\mathsf{tx}^1 := \mathsf{pay}(u_1, u_2^1, v)$. To argue indistinguishability, we follow the same case order as in Table II in the challenger definition.

1) tx^0 **and** tx^1 **unsuccessful;** $r :=$ **unsuccessful.** An unsuccessful transaction does not modify the network. Moreover, no balancing transaction is performed in this case. Therefore, for both choices of b, the nw'^b will be exactly equal to nw.

2) tx^0 **unsuccessful;** tx^1 **successful;** $r :=$ **unsuccessful.** Independently of the value of b, the balancing transaction reduces the available credit between u_1 and u_2^1 in network nw resulting in a network nw_m such that both tx^0 and tx^1 do not succeed; thus, similarly to the previous case, for both choices of b, $nw'^b = nw_m$, and no information is leaked to the attacker.

3) tx^0 **successful;** tx^1 **unsuccessful;** $r :=$ **successful.**

- $b = 0$. The balancing transaction does not modify nw; i.e., $nw_m^0 = nw$. Performing tx^0 over nw_m^0 now reduces v credits from the flow between u_1 and u_2^0 in nw'^0.
- $b = 1$. The balancing transaction results in nw_m^1 with v credit added to $u_2^0 \rightarrow u_2^1$. Performing tx^1 over nw_m^1 now results in reductions of v credit from the flow between u_1 and u_2^0, and v credit from $u_2^0 \rightarrow u_2^1$ to obtain nw'^1.

Independently of b, the attacker observes that in nw'^b there has been a reduction of v credit in the flow between u_1 and u_2^0. Even with this leak of information, the attacker is still not able to determine if u_2^0 is the actual receiver of the transaction or just an intermediary node for a longer payment path.

4) tx^0 **successful;** tx^1 **successful;** $r :=$ **successful.** This case follows by the same reasoning as the previous case, which completes our case analysis for pay.

Assume that the attacker provides the challenger with two chgLink challenge transactions: $\mathsf{tx}^0 := \mathsf{chgLink}\ (u_1, u_2^0, v)$ and $\mathsf{tx}^1 := \mathsf{chgLink}(u_1, u_2^1, v)$.

- $b = 0$. The balancing transaction results in a network nw_m^0 with v credit added to $u_1 \rightarrow u_2^1$. Performing tx^0 over nw_m^0 now implies an increment of v credit to $u_1 \rightarrow u_2^0$, resulting in the network nw'^0.
- $b = 1$. The balancing transaction results in a network nw_m^1 with v credit added to $u_1 \rightarrow u_2^0$. Performing tx^1 over nw_m^1 now implies an increment of v credit to $u_1 \rightarrow u_2^1$, resulting in the network nw'^1.

Independently of b, the attacker observes two changes in the network nw'^b with respect to the network nw: (i) $u_1 \rightarrow u_2^0$ has been incremented by v credit; (ii) $u_1 \rightarrow u_2^1$ has been incremented by v credit. All these cases show that the attacker cannot distinguish the challenge transaction by studying the changes on the credit network. The analysis of the attacker trying to break privacy by observing the memory access pattern of the challenger remains exactly the same as in the proof of value privacy. Hence, the theorem is proved. ■

VII. Performance Analysis

In this section, we describe the implementation and evaluate the practicality of PrivPay using detailed data gathered from the Ripple payment system over a period of four months. We also suggest some implementation-level optimizations to PrivPay and discuss other important system factors.

A. Implementation

We have developed a prototypical C++ implementation [36] to demonstrate the feasibility of our construction. The implementation encompasses both the universe creator module and the transaction module, thus simulating the functionality that would be performed by the SC. For symmetric encryption, we have used the Intel AES-NI library [42] to interact with the AES hardware implementation available on our test machine (see Section VII-C).

Implementation-level Optimizations. The universe creator module operations can be performed in the background independently from the transaction module. Moreover, the execution of the ObliBFS algorithm for each of the landmark nodes

can run in parallel, exploiting the multi-core architecture of modern hardware.

Our ObliBFS algorithm is further optimizable for sparse graphs. In these graphs, the majority of nodes have a small number of neighbours while only a few nodes have a large number of neighbours. Given that, the variable *MAX_ADJ* (see Section V-A) can be set to the maximum number of neighbours among the nodes with a small number of neighbours. Thereby, the number of iterations of the *for* loop in ObliBFS is reduced while the obliviousness is still preserved. For those few nodes with a large number of neighbours, the *for* loop is executed several times, adding a fake access to the ORAM instance for the credit network graph before every extra iteration.

Note that applying this optimization to the universe creator module maintains the privacy of ObliBFS (see Section VI-A) as we assume that a new user indicates her estimated maximum number of neighbours while joining the credit network. Based on this, the user is included in the group of nodes with a large number of neighbours or in the group with a small number of neighbours such that the universe creator module can ensure that, for any given user, the for-loop (lines 20-31) of the ObliBFS instances is executed for the same number of times even when the user's links are modified.

Our ObliBFS algorithm is further optimized. The BFS auxiliary data is handled inside the universe creator module instead of storing it in an ORAM (see ORAM *aux* in Algorithm 1) within the service provider. Finally, we take advantage of the fact that after performing a successful transaction, the users are likely to create a credit link between them. We have adopted this phenomenon in our protocol: when the transaction module stitches the paths between sender and receiver (Algorithm 1 step 4), it always adds an extra path composed of the direct credit link between sender and receiver.

B. The Ripple Dataset

The Ripple payment network [37] is the only credit network system with a publicly available network graph and transaction ledger; thus, it is a natural choice for our experiments. We have used web sockets to perform customized queries to the Ripple server publicly available at s1.ripple.com, and have obtained the full Ripple credit network (i.e., its full ledger) at two different points in time: ledger 2830040 in October 2013 and ledger 4547183 in January 2014, along with all the transactions carried out in this period of time. The obtained raw dataset has been filtered according to the following criteria.

1) We only considered accounts that are funded: a Ripple account is *funded* when it owns a certain amount of XRP[1]. At the time of writing, 20 XRP are needed to fund an account.

2) We took into account transactions for payments and for setting up new links in the credit network. Ripple is a complex system that supports extra operations such as currency exchanges; however, these extra operations are currently outside the scope of our work.

3) Only transactions with fiat currencies are considered, and we have discarded transactions exchanging user's customized currencies and XRP. Customized currencies are avoided as there is no exchange rate in the market to translate them into a known fiat currency and thereby unify them with other transactions, while the XRP transactions are carried out directly from senders to receivers without following any paths.

4) We convert all the credit values into USD, so as to have a uniform credit value format for our experiments. We used the exchange rates available at the MtGox exchange (while it was still functioning) to change currencies into USD.

5) After filtering the original dataset obtained from the Ripple server, we obtained a reduced credit network graph $G(V, E)$ and a subset of transactions T. An edge (or link) capacity α_{ij} for a link $(i, j) \in E$ is computed as $\alpha_{ij} = \textit{credit\_limit}_{ji} - \textit{balance}_{ij}$, where $\textit{balance}_{ij}$ is the balance on the Ripple link and $\textit{credit\_limit}_{ji}$ is its capacity.[2]

The final step toward obtaining our experiment dataset consisted of carrying out every transaction $t \in T$ using the max-flow algorithm to find all possible paths between sender and receiver in G. If t can be successfully carried out, it is added to T'. Therefore, our experiment dataset is composed of the graph G and the transaction set T'.

Our filtered experiment dataset contains a graph with a set of 14,317 nodes and 14,176 links along with a transaction set composed of 8,124 pay transactions and 14,922 chgLink transactions.

C. Performance

We conducted our experiments on a machine with an Intel Xeon E5-4650L 2.60 GHz processor and 790 GB RAM. For our experiments, we first load the initial credit network state and we create a landmark universe out of it. In this way, we simulate the actual state that the payment system would have in a real deployment at the initial state (e.g., October 2013 in our dataset). Then, we set the universe creator module as a background process. In practice, several threads are executing the ObliBFS algorithm, thereby continually creating the corresponding BFS trees in an oblivious manner. Finally, we carry out the transactions in the same order as they happened in the real Ripple payment system by using the transaction module. Therefore, we can compare the outcome of PrivPay with the real-world system. We next analyze the results of our experiments for the crucial components of PrivPay.

Transaction time. We study two transaction types allowed in PrivPay: pay and chgLink. The average time needed to carry out a payment is 1.5 seconds while the average time for changing the credit of a link is only 0.1 seconds. As expected, the response time for a chgLink operation is smaller than the time required for pay. Nevertheless, even in the case of payment operations, the response time of a few seconds is perfectly acceptable for real-time online payments.

Data-oblivious BFS tree creation time. Our data-oblivious BFS algorithm (ObliBFS) is one of the main building blocks in our construction. We have thereby studied the time needed on average to create a BFS tree containing the information of the shortest path from the landmark node to every node in

[1] XRP is the symbol of the Ripple currency.

[2] Following Ghosh et al. [16], to study auctions on trust networks, there is no need to consider the account balances and credit limits on the edges of the credit network separately. All that matters is the remaining credit on a link.

	Adapted Canal	PrivPay
pay time (ms)	0.078	1510
chgLink time (ms)	0.005	95
BFS tree creation time (ms)	50	22000
Accuracy	97%	95%

TABLE III: Comparative study between our adaptation of the (non-private) Canal algorithm and PrivPay

the network. The outcome of our experiments is that PrivPay needs 22 seconds to complete one execution of ObliBFS. This time is as expected larger than the time needed to perform a payment. However, as we stated in Section VII-A, the ObliBFS functionality can be executed in parallel in the background and thus the payment transactions can be handled independently.

Accuracy. We also study the accuracy of our system as it answers transaction requests. Transaction requests of the type chgLink are always correctly answered, given that it is always possible to update a given credit link (or create a new link if it does not yet exist). Transaction requests of type pay are answered with 95% accuracy in PrivPay. Notice that the 5% of transactions that are inaccurately answered are only false negatives (i.e., a transaction is returned as unsuccessful even though the credit network allows such a transaction) and there are no false positives (i.e., a transaction is returned as successful even though the credit network does not allow such a transaction). A false positive is not possible in PrivPay as the available credit in every path is computed over the *current* state of the credit network; moreover, given that transactions are carried out sequentially, the changes from one transaction are reflected in the credit network before the following transaction is processed. As a result, our system does not incur any credit loss for the users.

Canal vs PrivPay. Viswanath et al. [48] gave us access to their Canal prototype code. We have adapted it to support directed graphs; thus we are able to simulate our Ripple credit network dataset. This experiment has allowed us to study the impact of adding privacy into a payment system. Table III compares the important aspects of Canal and PrivPay. The use of data-oblivious algorithms to achieve privacy has resulted in a noticeable increment in the response time for all the factors studied, namely pay, chgLink and BFS creation time. Nevertheless, the PrivPay performance is still acceptable for a payment system. The payment transaction accuracy, on the other hand, remains almost the same.

Scalability. To test scalability of PrivPay we have obtained a more recent snapshot of the Ripple network for the period of one week: from ledger 7513200 in July 1st, 2014 to ledger 7618095 in July 7th, 2014, along with all the transactions carried out in this period of time. The thus raw dataset obtained was filtered following the criteria described earlier in this section. Our filtered experiment dataset contains a graph with a set of 24,467 nodes and 49,396 links along with a transaction set composed of 1,486 pay and 1,014 chgLink transactions.

We observe that the processing time increases only almost linearly with respect to the number of nodes: the average time to carry out a payment increases to 3.4 seconds while the average time for changing the credit of a link grows to 0.2 seconds; finally, the average BFS tree creation time increases to 47 seconds. This demonstrates the scalability of our approach.

VIII. Related Work

Mittal et al. [28] consider the problem of link privacy for social networks. They propose the use of perturbed graphs to provide link privacy by deleting real edges and introducing fake edges in a social network, such that the transitive closure of the graph is still preserved; in particular, for a given link (u, v), they perform a random walk starting from v and ending in a node z such that link (u, v) is replaced by link (u, z). Sala et al. [41] tackle the problem by proposing a mechanism to publish social networks with privacy guarantees by using differential privacy. Given a social network and a desired level of differential privacy guarantee, they generate a new synthetic social network with differential privacy by introducing noise into degree correlation statistics. Zheleva et al. [51] and Hay et al. [20] propose to maintain link privacy by performing clustering of vertices and edges, aggregating them into super-vertices. Information about corresponding subgraphs can thereby be anonymized to a certain extent.

All of the above approaches have a similar characteristic: they modify the network connectivity such that loss of system reliability is bounded. Although such a loss could be tolerated for certain types of systems (e.g., social networking applications), it might not be acceptable within reputation and payment systems such as Bazaar and Ripple.

Carminati et al. [6] propose a flexible mechanism for protecting privacy of relationships between social network users. A node u creates a new relationship by sending a distribution rule to its neighbors in a private manner, where a distribution rule is a tuple (CK, DC), with CK identifying a symmetric key shared with each u's neighbor and DC denoting the condition under which the rule must be applied and/or forwarded by every u's neighbor. In this approach, a node v forwarding a certain distribution rule issued by a neighbor node u can learn the willingness of u to create a new relation (e.g., grant access to nodes located 3 hops away). Maintaining such a system in a centralized manner is a challenge. Moreover, in some credit network systems, such rules may reveal information about the business carried out by a certain node.

Several works such as Dynamix [31], Drac [9], and Pisces [29] provide solutions for communicating anonymously over social networks; however, they are not generalizable to credit systems, as they rely on the existence of unbounded links between users, which may severely harm the liquidity of the credit network.

Several hardware-assisted (or trusted computing-based) privacy solutions have been proposed in the literature. For example, Asanov [1], William, Sion and Carbunar [50] and Bajaj and Sion [4] employ trusted computing and oblivious RAM for database access privacy. Backes et al. [3] apply them to privacy-preserving online advertising, while Maas et al. [25] use them for providing general oblivious computations. Trusted computing has also been used to improve resilience of general secure multi-party computation [2]. In this work, we extend its utility to ensures privacy properties for credit networks.

IX. Conclusions

Credit networks represent trust relationships between users through a directed graph with link capacities corresponding to

trust relationships between users. Thanks to their robustness against intrusion, credit networks have been used in a variety of applications. For instance, Ripple is employed by tens of thousands of users.

This work focuses on the problem of privacy for transactions in a credit network. We show that it is possible to achieve strong privacy properties while retaining availability, scalability, and accuracy of credit networks. Our solution utilizes a new efficient oblivious landmark routing construction and deploys a secure co-processor to implement it securely in the service provider environment. This allows the service provider to perform the payment transactions correctly without learning anything about payment amounts or about one (or both) of the parties involved. We formalize two fundamental privacy goals (i.e., value privacy and receiver privacy) as cryptographic games, and analyze the security of PrivPay against these definitions.

We have implemented PrivPay and have performed an elaborated performance analysis using data gathered from the Ripple payment network. Our analysis demonstrates that the users may perform real-time transactions using PrivPay without any significant reduction in the accuracy of transactions. For our experiments, we considered the Ripple dataset containing real-life transactions. In the future, we plan to create and employ synthetic datasets from eBay in order to emulate even larger datasets for the graphs and transactions.

ACKNOWLEDGMENTS

The authors gratefully acknowledge Viswanath et al. for granting access to the Canal code and providing us with necessary support. This work was supported by the German research foundation (DFG) through the Emmy Noether program and the Cluster of Excellence on Multimodal Computing and Interaction (MMCI), and by the German Federal Ministry of Education and Research (BMBF) through the Center for IT-Security, Privacy and Accountability (CISPA). We thank the reviewers for their helpful comments.

REFERENCES

[1] D. Asonov, *Querying Databases Privately: A New Approach to Private Information Retrieval*. Springer, 2004, vol. 3128.

[2] M. Backes, F. Bendun, A. Choudhury, and A. Kate, "Asynchronous MPC with a strict honest majority using non-equivocation," in *ACM PODC '14*, 2014, pp. 10–19.

[3] M. Backes, A. Kate, M. Maffei, and K. Pecina, "ObliviAd: Provably Secure and Practical Online Behavioral Advertising," in *IEEE S&P (Oakland)*, 2012, pp. 257–271.

[4] S. Bajaj and R. Sion, "TrustedDB: A Trusted Hardware based Outsourced Database Engine," *PVLDB*, vol. 4, no. 12, pp. 1359–1362, 2011.

[5] M. Blanton, A. Steele, and M. Alisagari, "Data-oblivious Graph Algorithms for Secure Computation and Outsourcing," in *ASIACCS '13*, 2013, pp. 207–218.

[6] B. Carminati, E. Ferrari, and A. Perego, "Private Relationships in Social Networks," in *ICDE Workshops*, 2007, pp. 163–171.

[7] 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, pp. 147–156.

[8] P. Dandekar, A. Goel, M. P. Wellman, and B. Wiedenbeck, "Strategic Formation of Credit Networks," in *WWW '12*, 2012, pp. 559–568.

[9] G. Danezis, C. Díaz, C. Troncoso, and B. Laurie, "Drac: An Architecture for Anonymous Low-Volume Communications," in *PETS'10*, 2010, pp. 202–219.

[10] D. DeFigueiredo and E. T. Barr, "TrustDavis: A Non-Exploitable Online Reputation System," in *7th IEEE International Conference on E-Commerce Technology*, 2005, pp. 274–283.

[11] S. Delaune, S. Kremer, and M. D. Ryan, "Verifying Privacy-type Properties of Electronic Voting Protocols," *Journal of Computer Security*, vol. 17, no. 4, pp. 435–487, 2009.

[12] R. Dingledine, N. Mathewson, and P. Syverson, "Tor: The Second-generation Onion Router," in *the 13th Conference on USENIX Security Symposium*, 2004, pp. 21–21.

[13] Y. Dinitz, "Dinitz's Algorithm: The Original Version and Even's Version," in *Theoretical Computer Science*, 2006, pp. 218–240.

[14] A. J. Feldman, A. Blankstein, M. J. Freedman, and E. W. Felten, "Privacy and Integrity are Possible in the Untrusted Cloud," *IEEE Data Eng. Bull.*, vol. 35, no. 4, pp. 73–82, 2012.

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

[16] A. Ghosh, M. Mahdian, D. M. Reeves, D. M. Pennock, and R. Fugger, "Mechanism Design on Trust Networks," in *WINE'07*, 2007, pp. 257–268.

[17] 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.

[18] O. Goldreich and R. Ostrovsky, "Software protection and simulation on oblivious RAMs," *Journal of the ACM*, vol. 43, pp. 431–473, 1996.

[19] "IDG Capital Partners and Google Ventures Invest in Ripple Developer OpenCoin," http://online.wsj.com/article/PR-CO-20130514-908271.html.

[20] M. Hay, G. Miklau, D. Jensen, D. Towsley, and P. Weis, "Resisting Structural Re-identification in Anonymized Social Networks," *Proc. VLDB Endow.*, vol. 1, no. 1, pp. 102–114, 2008.

[21] IBM Systems, "IBM Systems cryptographic hardware products," http://www-03.ibm.com/security/cryptocards/.

[22] A. M. Kakhki, C. Kliman-Silver, and A. Mislove, "Iolaus: securing online content rating systems," in *WWW*, 2013, pp. 919–930.

[23] D. Karlan, M. Mobius, T. Rosenblat, and A. Szeidl, "Trust and Social Collateral," *The Quarterly Journal of Economics*, vol. 124, no. 3, pp. 1307–1361, 2009.

[24] A. Liu, "Ripple Labs Signs First Two US Banks," https://ripple.com/ripple-labs-signs-first-two-us-banks/, 2014.

[25] M. Maas, E. Love, E. Stefanov, M. Tiwari, E. Shi, K. Asanovic, J. Kubiatowicz, and D. Song, "PHANTOM: Practical Oblivious Computation in a Secure Processor," in *ACM CCS '13*, 2013, pp. 311–324.

[26] T. Minkus and K. W. Ross, "I Know What You're Buying: Privacy Breaches on eBay," in *PETS'14*, 2014.

[27] A. Mislove, A. Post, P. Druschel, and K. P. Gummadi, "Ostra: Leveraging Trust to Thwart Unwanted Communication," in *NSDI'08*, 2008, pp. 15–30.

[28] P. Mittal, C. Papamanthou, and D. X. Song, "Preserving Link Privacy in Social Network Based Systems," in *NDSS*, 2013.

[29] P. Mittal, M. Wright, and N. Borisov, "Pisces: Anonymous Communication Using Social Networks," in *NDSS*, 2013.

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

[31] A. Mohaisen and Y. Kim, "Dynamix: anonymity on dynamic social structures," in *ASIACCS*, 2013, pp. 167–172.

[32] A. Mohaisen, H. Tran, A. Chandra, and Y. Kim, "Trustworthy distributed computing on social networks," in *ASIACCS*, 2013, pp. 155–160.

[33] A. Narayanan and V. Shmatikov, "Robust De-anonymization of Large Sparse Datasets," in *IEEE S&P (Oakland'08)*, 2008, pp. 111–125.

[34] A. Narayanan, V. Toubiana, S. Barocas, H. Nissenbaum, and D. Boneh, "A Critical Look at Decentralized Personal Data Architectures," in *Data Usage Management on the Web*, 2012.

[35] A. Post, V. Shah, and A. Mislove, "Bazaar: Strengthening User Reputations in Online Marketplaces," in *NSDI'11*, 2011, pp. 14–14.

[36] "The PrivPay Project Webpage," http://crypsys.mmci.uni-saarland.de/projects/PrivPay/.

[37] "Ripple," https://ripple.com/.

[38] "Ripple FAQ," https://ripple.com/wiki/FAQ.

[39] "Ripple for Bitcoin Exchanges," https://ripple.com/wiki/Introduction_to_Ripple_for_Bitcoiners.

[40] P. Rizzo, "Fidor Becomes First Bank to Use Ripple Payment Protocol," http://www.coindesk.com/fidor-becomes-first-bank-to-use-ripple-payment-protocol/, 2014.

[41] A. Sala, X. Zhao, C. Wilson, H. Zheng, and B. Y. Zhao, "Sharing Graphs Using Differentially Private Graph Models," in *IMC '11*, 2011, pp. 81–98.

[42] I. E. Security, "Intel Data Protection Technology with AES-NI and Secure Key," http://www.intel.com/content/www/us/en/architecture-and-technology/advanced-encryption-standard--aes-/data-protection-aes-general-technology.html.

[43] K. Sharad and G. Danezis, "An Automated Social Graph De-anonymization Technique," *CoRR*, vol. abs/1408.1276, 2014.

[44] S. W. Smith, "Outbound Authentication for Programmable Secure Coprocessors," *Inter. Journal of Information Security*, vol. 3, no. 1, pp. 28–41, 2004.

[45] E. Stefanov, E. Shi, and D. X. Song, "Towards Practical Oblivious RAM," in *NDSS*, 2012.

[46] E. Stefanov, M. van Dijk, E. Shi, C. Fletcher, L. Ren, X. Yu, and S. Devadas, "Path ORAM: An Extremely Simple Oblivious RAM Protocol," in *CCS '13*, 2013, pp. 299–310.

[47] P. F. Tsuchiya, "The Landmark Hierarchy: A New Hierarchy for Routing in Very Large Networks," *SIGCOMM Comput. Commun. Rev.*, vol. 18, no. 4, pp. 35–42, Aug. 1988.

[48] B. Viswanath, M. Mondal, K. P. Gummadi, A. Mislove, and A. Post, "Canal: Scaling Social Network-based Sybil Tolerance Schemes," in *EuroSys '12*, 2012, pp. 309–322.

[49] M. P. Wellman and B. Wiedenbeck, "An empirical game-theoretic analysis of credit network formation," in *Allerton Conference*, 2012, pp. 386–393.

[50] P. Williams, R. Sion, and B. Carbunar, "Building Castles out of Mud: Practical Access Pattern Privacy and Correctness on Untrusted Storage," in *ACM CCS '08*, 2008, pp. 139–148.

[51] E. Zheleva and L. Getoor, "Preserving the Privacy of Sensitive Relationships in Graph Data," in *PinKDD'07*, 2008, pp. 153–171.

APPENDIX

In this section we present how credit networks are instantiated in the related systems proposed in the literature. In addition, we describe how our privacy preserving solution for credit networks can be incorporated into these systems.

Ripple Payment System. Ripple [37], created by Ryan Fugger, is an online payment system allowing the exchange of multiple currencies exchanged as *IOweYou* transactions. It uses a trust network that consists of two directed graphs $G(V, E)$ and $G'(V, E')$ defined on a shared set of vertices V. The set of edges E gives the pairwise account balances between nodes. An edge weight o_{ij} quantifies the *IOweYou* obligations that i has to j. The set of edges E' gives pairwise credit limits between agents. An edge weight t_{ij} specifies that i has extended j a credit line of t_{ij}. Ghosh et al [16] show that to study auctions on trust networks, the two graphs composing the trust network can be reduced to a simple graph comprising the same set of vertices V while the set of edges E represents the credit c, where an edge $(i, j) \in E$ has credit $c_{ij} = t_{ji} - o_{ij}$. A transaction works by paying credits along the path between sender and receiver, as we have defined in Section II-A.

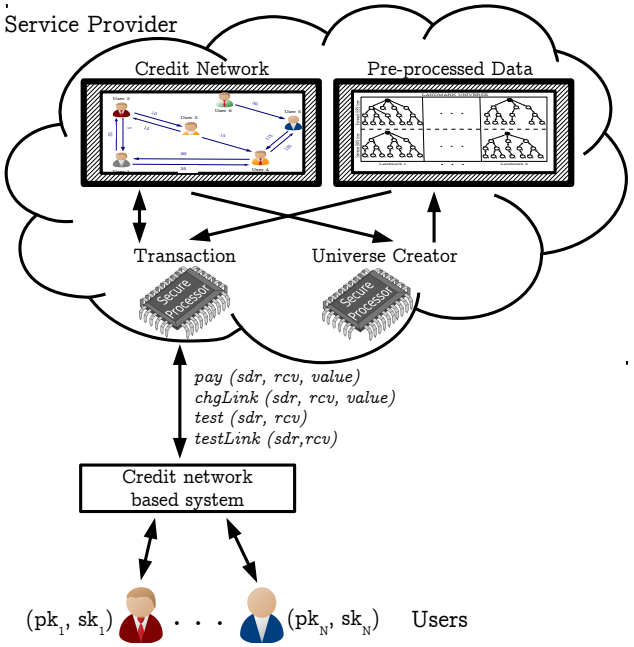

Fig. 8: Integration of PrivPay in a credit network-based system.

Bazaar Reputation System. Bazaar [35] strengthens users' reputation in online marketplaces, such as eBay, in the face of collusion, white-washing and Sybil attacks. The system creates a risk network represented as a graph $G(V, E)$ constructed from the feedback of previous transactions. In particular, the set of edges E represents successful transactions between nodes. An edge weight represents the value of the transactions successfully completed between the linked pair of nodes. A transaction in Bazaar works as defined for credit networks, but with an addition. If the transaction is successful, the paid credit (decreased weight along the path between the sender and the receiver) is restored. Further, the direct credit between the sender and the receiver is increased by the transaction value. If the sender and the receiver do not share a direct link, it is initialized with a credit equal to the transaction value.

Ostra. Ostra [27] is similar in spirit to Gmail's priority inbox and it aims at stopping unwanted communication (i.e., spam) between users. For that purpose, Ostra assumes a social network defined as a graph $G(V, E)$ where an edge weight gives the maximum number of messages that can be sent between the nodes. When a user wants to perform a transaction (i.e., send an e-mail), the credit along the path found between the sender and the receiver is decreased. The amount of credit deducted per transaction is set to one in Ostra.

Adapting credit networks to PrivPay. As depicted in Fig. 8, PrivPay exports an API with methods pay*(sdr, rcv, value)*, chgLink*(sdr, rcv, value)*, test*(sdr,rcv)* and testLink*(sdr,rcv)*. The API can be used for any of the existing credit networks to provide a privacy-preserving service. In particular, a credit network can transfer the network graph to PrivPay by iteratively invoking chgLink. When a user requests a transaction to the network, it can invoke a pay operation to PrivPay and forward the answer returned by PrivPay. Similarly, user queries for credit available on a link or between two users can be simulated by using testLink and test correspondingly in PrivPay.