# SmartPay: A Lightweight Protocol to Enforce Trust Preferences in Mobile Person-to-Person Payments

Barbara Carminati
DiSTA
University of Insubria, Italy
Email:
barbara.carminati@uninsubria.it

Elena Ferrari
DiSTA
University of Insubria, Italy
Email: elena.ferrari@uninsubria.it

Ngoc Hong Tran
DiSTA
University of Insubria, Italy
Email: ngoc.tran@uninsubria.it

## ABSTRACT

The technological advancements in Internet speeds, increased computing power and smart phones have pushed the rise of new digital methods supporting mobile person-to-person (P2P) payments. Despite the growing interest in these new methods, we believe that, to fully enable this increasing rise of digital wallets, there is the need for tools helping a person in judging the risk of a money transfer. For this purpose, this paper aims at exploiting social network connections. This is achieved by making payers/payees able to state their trust preferences with respect to the potential payees/payers. Trust preference evaluation requires to find social connections between a payer and a payee across, possible, different social network realms. We therefore propose a light cryptography protocol, specifically targeted to mobile P2P payments, that besides providing good performance, ensures user information privacy.

## I INTRODUCTION

Today we are experiencing the rise of new digital methods of making person-to-person (P2P) payments, in contrast to traditional checks and cash. A striking example is given by PayPal[1] with more than 110 million active accounts, among which 2.6 million accounts are using Paypal for an average of 98% of their online purchases. Similar services are provided also by other major companies, like Google (Google Wallet[2]), American Express (Serve[3]), AT&T, TMobile and Verizon Wireless (ISIS[4]), just to give some examples. This rise is further fueled by the increasing success of mobile payment services, which allow customers to use their mobile phone to support mobile Person-to-Person payments. As an example, Gartner [1], forecasts that in 2016 there will be 448 million mobile payment users in a market worth 617 billion USD, and the global mobile transaction volume and value averages 42% annual growth between 2011 and 2016.

Despite the success of these new digital P2P payment methods, we believe that, to fully enable this increasing rise of digital wallets, relevant challenges still need to be addressed. Indeed, a survey conducted by Federal Reserve on consumers and mobile financial services, highlights that the major concern for not adopting these services is security [2]. We believe that, in addition of securing mobile environments to prevent private information leakage, malware, etc., there is also the need of tools helping users in judging the risk of accepting a money transfer. As a step in this direction, this paper leverages social network connections to help the decision making processes behind mobile P2P payments. Indeed, in real life, we make decisions on accepting money from or on giving money to someone based on what we know about him/her. This is also a well-established behavior in e-commerce sites, where users refer to other opinions (e.g., ratings/comments) to judge vendors trustworthiness. Therefore, in this paper, we investigate how P2P payment methods can be enhanced to make payers/payees able to exploit their social connections to decide on possible money transfers. More precisely, we propose to make payer (i.e., payee) able to state a set of *trust preferences* stating how the potential payee/payer has to be connected with him/her in order to give/accept money. Trust preferences state conditions on: (1) the type of the relationship that must exist between the payer and the payee (e.g., friend, parents, just-met), (2) the distance between the payer and the payee, i.e., the number of hops in the path connecting them, (3) the trust value associated with this relationship. As an example, a user can state that he/she will accept to transfer money only to payees that are friends of one of his/her friends with a high trust value (i.e., type=friends, distance=2, trust value = high). Before committing a money transfer, both payer and payee evaluate their

[1]https://www.paypal.com/webapps/mpp/ent-online-attract-shoppers
[2]http://www.google.com/wallet/
[3]http://www.serve.com/
[4]http://www.paywithisis.com/

local trust preferences to decide whether the other can be considered enough trusted.

Evaluating a trust preference implies finding connecting paths between payer and payee. In designing a protocol to support this task, we kept into account two main requirements. The first is that the solution has to support all existing mobile P2P payment methods as well as existing social network services, without forcing an integration between these platforms. Another relevant issue is that nowadays persons are used to join several social networks, maintaining different accounts with different contact lists. Therefore, we need to support path finding across different social network realms. To cope with these interoperability issues, we designed a solution such that trust preference evaluation is locally performed on each peer by means of a mobile app called *SmartPay*, assuming that user contacts list is locally managed by the mobile app. Therefore, the main goal is to discover the relationship path between two arbitrary nodes in a distributed network. In designing our protocol, we have considered two essential requirements: (1) *relationship privacy* - relationship information is sensitive personal data. As such, information on the type, trust and depth of a given relationship should be known only by the users establishing that relationship, not by intermediate nodes participating in the protocol; (2) *limitations of mobile devices* - almost all mobile devices are characterized by low battery, low memory, low power, limited throughput. To cope with the above limitations, we propose of light cryptographic protocol for *mobile-oriented decentralized path discovery*. The protocol makes use of Elliptic Curve Cryptography ($ECC$) [3] to obtain a protocol running steadily and fast. To fully protect relationship information, we also propose a privacy-preserving approach to compute trust value between two adjacent nodes according to the well-know Tidal Trust algorithm [4]. As experiments show, this solution can work with mobile devices efficiently.

A preliminary version of the protocol has been presented in [5]. In this paper we extend [5] in several directions: a privacy-preserving intersection protocol is illustrated to complete the basic protocol introduced in [5]; a discussion on security properties of the protocol is presented; comparative experiments with existing path discovery protocol are provided.

The remainder of the paper is organized as follows. Background and notations are presented in Section II. Section III introduces trust-driven mobile P2P payments. Mobile-oriented decentralized path discovery protocol is presented in Section IV, whereas Section V deals with optimization strategy. Section VI discusses the protocol security properties. Experimental results comparing the proposed solution with existing path discovery protocol are illustrated in Section VII. Finally, Section IX concludes the paper.

## II BACKGROUND AND NOTATIONS

### 1 TRUST PREFERENCES IN SOCIAL NETWORKS

We model a social network as a directed labeled graph $\mathcal{G}=(V, E, RT, \phi)$, where $RT$ is a finite set of relationship types, $V$ is a finite set of users, $E \subseteq V \times V \times RT$ is a set of edges representing relationships between users, and $\phi : E \rightarrow [0, 1]$ is a function mapping each edge $e \in E$ to a trust level. Given a relationship between two nodes $v, v'$, denoted by $rel(v, v')$, this is defined as a direct relationship if $rel(v, v') = e \in E$, it is defined as an indirect relationship if $rel(v, v')$ connects the two nodes by a path consisting of more than one edge, all with the same relationship type. In case of indirect connections, the depth of a relationship is given by the number of edges in the connecting path. Literature offers several algorithms to combine and to aggregate the trust values associated with the edges involved in a connecting path so as to compute trust value for an indirect relationship. In this paper, we adopt one of the most well-accepted one, that is, Tidal Trust [4] (cfr. Section IV).

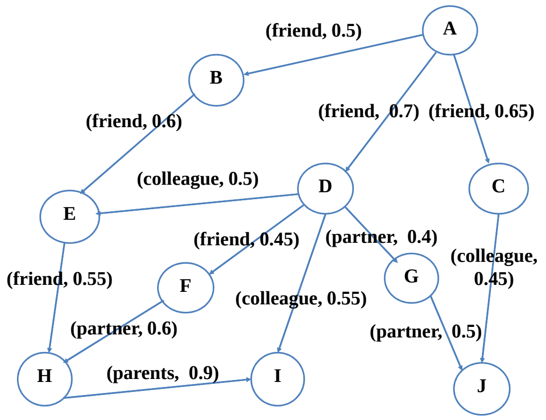


Figure 1: An example of social network

Based on this notation, we can now formalize *trust preferences*. A trust preference is defined as a pair (`P`, `TC`), where `P` indicates whether the trust preference has to be applied for a money transfer where the specified user is the payer (`P`=*payer*), or the payee

(P=*payee*), and TC is a set of *trust conditions* specifying trust requirements. In particular, a trust condition is a tuple TC $= (rt, d_{max}, t_{min})$, which states that between the trust preference owner and the other party there should exist a relationship of type $rt$, with maximum depth equal to $d_{max}$ and a trust level greater than or equal to $t_{min}$.

Consider the graph in Figure 1, the following preference specified by Bob: $(payer, (Colleague, 3, 0.6))$ states that Bob accepts to be the payer for nodes which are connected with him with a direct or an indirect relationship of type *colleague*, maximum depth 3, and minimum trust value 0.6.

For simplicity, in the paper, we assume that a trust preference consists only of one trust condition TC, and we denote with $rt_{TC}$, $depth_{TC}$, and $trust_{TC}$ the conditions specified on relationship type, depth and trust in TC.

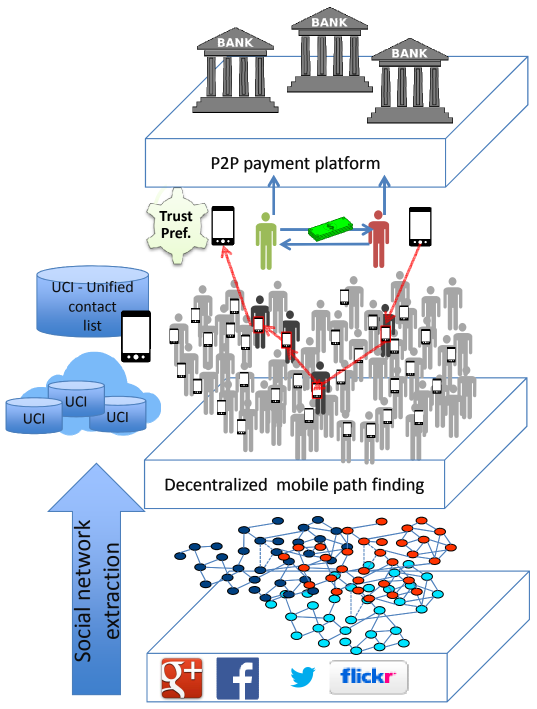

BANK
BANK
BANK
P2P payment platform
Trust Pref.
UCI - Unified contact list
UCI
UCI
UCI
Decentralized mobile path finding
Social network extraction
flickr

Figure 2: Trust-driven mobile Person-to-Person payments

## 2 ELLIPTIC CURVE CRYPTOGRAPHY (ECC)

In our protocol, we use Elliptic Curve Cryptography ($ECC$) [3] based on the binary finite field $F_2^m$. As such, we assume that each node $v_i$ using $SmartPay$ has a pair of keys ($k_i$, $k_i \cdot B$), where the public key $k_i \cdot B$, the Elliptic Curve ($E$) and the base point $B$ will be publicly published when $v_i$ installs $SmartPay$, whereas the private key ($k_i$) is known only by $v_i$. Let us remind $ECC$ encryption/decryption operations by assuming that Alice wishes to send a message $M$ to Bob, and Bob has a pair of keys ($k_{Bob}$, $k_{Bob} \cdot B$), where $k_{Bob}$ is Bob's private key, and $k_{Bob} \cdot B$ is Bob's public key. To encrypt $M$, Alice generates $E(M) = (Y1, Y2)$, $Y1 = r_{Alice} \cdot B$, $Y2 = M + r_{Alice} \cdot (k_{Bob} \cdot B)$, where $r_{Alice}$ is a random integer generated by Alice. Given $E(M)$ Bob uses his private key $k_{Bob}$ to retrieve $M$ as follows: $D(E(M)) = Y2 - k_{Bob} \cdot Y1$.

## III TRUST-DRIVEN MOBILE P2P PAYMENTS

As depicted in Figure 2, $SmartPay$ implements the bridge between social networking services and P2P payment services. We assume that $SmartPay$ periodically synchronizes with user social network accounts so as to extract their contact lists. These are merged into a unique list, called Unified Contact List (UCL). We assume that users UCL can be either locally stored or stored into a cloud public storage service.[5]

SmartPay provides users with the possibility of further classifying their relationships by specifying, for each of them, the type of the relationship, e.g., parent, colleague, classmate, defined according to a standard vocabulary like FOAF [8], and a trust value, representing the strength of the relationship. As described in Section 2, $SmartPay$ also provides a tool for suggesting a user a proper trust value for those users he/she does not know, based on the well-known Tidal-Trust algorithm. Thus, given the UCL, $SmartPay$ can access the user social graph, representing the merge of the partial views the user has on each single social network. Formally, the social graph follows the definition and notation provided in Section II

Before committing to a money transfer, a user evaluates through $SmartPay$ whether the payer/payee

[5]Protection of UCLs at cloud side is out of the paper scope. However, existing encryption techniques for cloud (e.g., [6], [7]) can be easily adopted to make only authorized mobile apps able to access the proper UCLs.

satisfies his/her local trust preferences and, if this is the case, the user proceeds with his/her preferred mobile P2P payment method. This evaluation requires to discover relationship paths connecting the two involved nodes, to verify if at least one of them satisfies the specified trust preferences. Since relationship information is locally managed, we propose a distributed protocol that, by means of node collaboration, traverses the social graph and looks for connecting paths. For each of the traversed path, the protocol collects the information on trust, depth, and relationship type. The collected information is stored into a set of data structures (called *tokensets*) and propagated through the graph. For simplicity, hereafter, we assume this process is always started by the payer. Therefore, once the payer receives a request of a money transfer from a payee through an email or a message on the social network, the payer initializes a tokenset including three distinct tokens, one for each path property (i.e., depth token, trust token, and relationship type token, respectively). The payer then forwards the tokens to his/her direct neighbors. These nodes update the received tokenset with the information on depth, trust and relationship type of the edge to be traversed and forward the updated tokenset to their neighbors. The process is iterated until the payee receives the tokenset, which is then send directly back to the payer by email or mobile internet. As soon as the payer receives back the set of tokensets, he/she retrieves the specified trust condition TC, i.e., the trust condition in the trust preference defined for the payer role. Then, by using the relationship type token, the payer is able to determine whether all the edges in the path have a relationship type equal to the one required by the trust condition. In case the edge relationship types are different, the payer is not able to determine their real values. In contrast, by exploiting depth and trust tokens the payer can determine the number of edges, and the trust value of the indirect relationship, without knowing the trust value of any edge in the path.

## IV MOBILE-ORIENTED DECENTRALIZED PATH FINDING

In this section, the protocol will be described in further details introducing how tokens are initialized, updated and propagated along the graph. Before that, we present notations we adopt hereafter. According to the social network model given in Section II, a path $p$ traversed by the collaborative protocol can be denoted as $p = \{v_0, v_1, \ldots, v_p\}$, where $v_0$ is a payer and $v_p$ is a payee. Thus, given $p$ we denote with $Tr_{(i)(i+1)}$, $rt_{(i)(i+1)}$ and $d_{(i)(i+1)}$, respectively, the trust value, relationship type and depth of the edge connecting $v_i$ to $v_{i+1}$ in $p$. Moreover, the proposed protocol exploits $ECC$ to protect relationship privacy. Data to be encrypted with $ECC$ have to be encoded in points in elliptic curve.[6] Therefore, given an edge $e$=($v_i$,$v_{i+1}$), we denote with $P_{d_{(i)(i+1)}}$, $P_{rt_{(i)(i+1)}}$, $P_{Tr_{(i)(i+1)}}$, respectively, the points in $(E)$ encoding depth, relationship type and trust value of edge $e$.

### 1 DEPTH COMPUTATION

By exploiting $ECC$ properties, depth of the traversed path $p$ can be computed by securely aggregating points in $(E)$ encoding the depth of each single edge in $p$. Thus, each node $v_i$ generates a big random number $r_{(i)(i+1)} \in \mathbb{N}^+$, which is used together with the payer's public key $(k_0 \cdot B)$ to encrypt the distance between $v_i$ and $v_{i+1}$, that is, the value 1. The obtained encrypted value is then added to the content of the received depth token. It is important to note that, even if depth values have always the same value (i.e., 1), analysis of encrypted text from attackers is still hard since each user in the path selects a different big random variable $r_{(i)(i+1)}$. Moreover, only the payer $v_0$ can know the final depth value of the whole relationship path, since intermediate nodes only participate in aggregating the depth value in the relationship path without the ability to decrypt it. Thus, the privacy of the payer and the previous nodes is preserved. Let us see the depth token computation by presenting its initialization, computation and validation.

*Depth token initialization.* The payer $v_0$ generates the point $P_{d_{(0)(1)}}$ on $(E)$ corresponding to $d_{(0)(1)}$. Then, it randomly generates a big number $r_{(0)(1)}$ for encrypting $P_{d_{(0)(1)}}$ with his/her public key $k_0 \cdot B$. We obtain: $E_{(0)(1)} = (Y1_{v_0},\ Y2_{v_0}) = (r_{(0)(1)} \cdot B,\ P_{d_{(0)(1)}} + r_{(0)(1)} \cdot k_0 \cdot B)$. The resulting encryption is then broadcast to each of his/her direct contacts, that is, the node $v_1$ in each path $p$ the protocol will find.

[6]Literature offers several algorithms for point compression (see [9] [10] [11]) that can be used to encode and decode a bit string in $F_2^m$ into a point on a binary elliptic curve $(E)$. These mapping functions do not affect the security of the protocol at all. For this reason, in this work a message $W$ is mapped to a point on $(E)$ as $P = W \cdot B$, where $B$ is the base point, due to the simplicity and the additive property of the mapping.

*Depth token computation at intermediate nodes.* Once an intermediate node $v_i$ receives the depth token from a neighbor $v_{i-1}$, it has to update and send it to each of its neighbors, except from $v_{i-1}$. Each forwarded depth token has to be updated so as to add to its contents, i.e., $E_{(0)(i)}$, the value $E_{(i)(i+1)}$, that is, the encryption with $v_0$ public key of point $P_{d_{(i)(i+1)}}$, where $d_{(i)(i+1)}$ is the depth between $v_i$ and neighbor $v_{i+1}$ to which the token will then be forwarded. As such, we have that: $E_{(0)(i+1)} = E_{(0)(i)} + E_{(i)(i+1)}$, where $E_{(i)(i+1)} = (r_{(i)(i+1)} \cdot B, P_{d_{(i)(i+1)}} + r_{(i)(i+1)} \cdot k_0 \cdot B)$. Thus, $E_{(0)(i+1)} = =(r_{(0)(i)} \cdot B + r_{(i)(i+1)} \cdot B, (P_{d_{(0)(i)}} + r_{(0)(i)} \cdot k_0 \cdot B) + (P_{d_{(i)(i+1)}} + r_{(i)(i+1)} \cdot k_0 \cdot B))$ $=((r_{(0)(i)} + r_{(i)(i+1)}) \cdot B, (P_{d_{(0)(i)}} + P_{d_{(i)(i+1)}}) + (r_{(0)(i)} + r_{(i)(i+1)}) \cdot k_0 \cdot B)$.

Then, $v_i$ broadcasts the resulting $E_{(0)(i+1)}$ to each of its direct neighbors, i.e., $v_{i+1}$.

*Depth token validation.* The depth token is flooded until it reaches the payee $v_p$. Then, for each founded path, $v_p$ sends the corresponding depth token back to the payer $v_0$. By the depth token, payer $v_0$ obtains the aggregated depth value $E_{(0)(p)} = (\sum_{i=0}^{p-1} r_{(i)(i+1)} \cdot B, \sum_{i=0}^{p-1} P_{d_{(i)(i+1)}} + (\sum_{i=0}^{p-1} r_{(i)(i+1)}) \cdot k_0 \cdot B)$. In order to extract depth $P_{d_{(0)(p)}}$ of the traversed path, i.e., $\sum_{i=0}^{p-1} P_{d_{(i)(i+1)}}$, $v_0$ decrypts $E_{(0)(p)}$ with his/her own private key $k_0$. Then, $v_0$ decodes $P_{d_{(0)(p)}}$ to obtain the value $d_{(0)(p)}$. Finally, $v_0$ checks if this value satisfies the trust condition TC.

## 2 TRUST COMPUTATION

Literature offers several algorithms to compute the trust value between two indirectly connected nodes [12]. In general, all of them aggregate, in different ways, the trust values associated with edges (direct relationship) connecting the nodes in the path. In designing the trust token, we started by the assumption that trust values for direct relationships are associated directly by involved users. However, these values could be too much subjective or, even worse, users could not be able to significantly quantify the trustworthiness for a given contact, as they might have not enough information. To cope with this issue, in this paper, we propose a trust computation able to automatically compute trust values between two adjacent nodes, The idea is that the resulting trust value can be considered as a suggested trust value between two users, which can then be multiplied with the user-given trust value, if present, as an *adjusting weight*. Among various trust algorithms, to compute the adjusting weight we selected the Tidal Trust algorithm [4]. This is known as the most famous and highly cited algorithm for inferring the trust between two adjacent nodes due to its simplicity and low complexity $O(V + E)$, which allows high scalability in its application. Thus, given two adjacent nodes $v_i$ and $v_j$, the trust value associated with edge connecting $v_i$ to $v_j$ is $Tr_{ij} = \mu_{ij} \cdot Tu_{ij}$, where $\mu_{ij}$ is the adjusting weight automatically computed between them according to the Tidal Trust algorithm, whereas $Tu_{ij}$ is the trust value given by $v_i$ to $v_j$, if any. Note that in case users are not able to assign a trust value, $Tu_{ij}$ is set to 1. Then, given a path $p$, we compute the aggregated trust value as the average of values $Tr$ associated with edges in $p$. To obtain this value, the trust token is generated and updated so as to contain the sum of $Tr$ values of traversed edges. The payer can simply computes the average by dividing this sum by the path depth.

Before illustrating the trust token in details, we have to note that Tidal Trust algorithm requires that the two adjacent nodes expose some private information, i.e., the list of their direct contacts. To cope with this problem, we propose a protocol so as to make two adjacent nodes able to locally compute Tidal Trust trust value in a preserving privacy way. In the following, we start presenting the privacy-preserving Tidal Trust computation. We then show how the obtained adjusting weights are used in the trust token computation.

***a) Privacy-preserving Tidal Trust computation between two adjacent nodes.*** According to [4], the trust value for two adjacent nodes $v_i$ and $v_j$ is computed as follows:

$$\mu_{ij} = \frac{\sum_{s_t \in adj(i) \ni Tr_{(i)(s_t)} \geq max} (Tr_{(i)(s_t)} \cdot Tr_{(s_t)(j)})}{\sum_{s_t \in adj(i) \ni Tr_{(i)(s_t)} \geq max} Tr_{(i)(s_t)}}$$

Formula 1. The trust value for two adjacent nodes $v_i$ and $v_j$

where $Tr_{(i)(s_t)}$, $Tr_{(s_t)(j)}$ denote the trust values, respectively, between $v_i$ and $v_{s_t}$, and between $v_{s_t}$ and $v_j$; $adj(i)$ is the adjacent node list of $v_i$; $s_t$ is a common neighbor of $v_i$ and $v_j$ whose trust value satisfies the condition that $Tr_{(i)(s_t)}$ is greater than a given threshold $max$; $t$ is the position of a common node $s_t$ in the set $adj(i)$. In order to evaluate Formula 1, $v_i$ and $v_j$ have to execute the following steps:

(1) $v_i$ and $v_j$ collaboratively retrieve the set of common contacts, denoted as $IS$. This is achieved by

a privacy-preserving intersection protocol described later in this section.
(2) For each common node $s_t$, $v_j$ maps the trust value $Tr_{(s_t)(j)}$ between $v_j$ and $s_t$ to a point on $(E)$ denoted as $P_{Tr_{(s_t)(j)}}$. Then, $v_j$ encrypts $P_{Tr_{(s_t)(j)}}$ with the payer $v_0$'s public key $(k_o \cdot B)$ and obtains $E_{(s_t)(j)} = (r_j \cdot B,\ P_{Tr_{(s_t)(j)}} + r_j \cdot k_0 \cdot B)$, where $r_j$ is a random number generated by $v_j$. $v_j$ also encrypts the point on $(E)$ corresponding to $s_t$'s identifier, denoted by $P_{ID_{s_t}}$ with $v_i$'s public key. The resulting encrypted value is denoted by $E_{(j)}(P_{ID_{s_t}})$. All these values are sent to $v_i$ as a set of pairs $[E_{(j)}(P_{ID_{s_t}}), E_{(s_t)(j)}]$, for each $s_t \in IS$.
(3) Once $v_i$ receives pairs $[E_j(P_{ID_{s_t}}), E_{(s_t)(j)}]$, it decrypts the values $E_{(j)}(P_{ID_{s_t}})$, and decodes $P_{ID_{s_t}}$ to obtain identifiers of common nodes, i.e., $ID_{s_t}$. Among the resulting identifiers, $v_i$ selects only those of the common nodes having $Tr_{(i)(s_t)} \geq max$, where $max$ is a lower bound for trust value as in Formula 1. Let $\theta_{(i)(j)}$ denotes the set of common nodes satisfying this condition.
(4) According to Formula 1, for each $s_t \in \theta_{(i)(j)}$, $v_i$ has to compute $Tr_{(i)(s_t)} \cdot Tr_{(s_t)(j)}$. However, to apply the binary $ECC$, $Tr_{(s_t)(j)}$ is mapped to a corresponding point on $(E)$, i.e., $P_{Tr_{(s_t)(j)}}$. Thanks to the additive property of the function used to map data in point on $(E)$,[7], we can calculate $Tr_{(i)(s_t)} \cdot P_{Tr_{(s_t)(j)}}$ instead. Moreover, since $P_{Tr_{(s_t)(j)}}$ is encrypted by $v_0$'s public key in $E_{(s_t)(j)}$, it is possible to exploit the additive property of $ECC$. Thus, to compute $Tr_{(i)(s_t)} \cdot E_{(s_t)(j)}$, $v_i$ adds to $E_{(s_t)(j)}$ the same value $E_{(s_t)(j)}$ for $(Tr_{(i)(s_t)} - 1)$ times.[8] We obtain a value, denoted as $addend_{s_t}$, such that $addend_{s_t} = Tr_{(i)(s_t)} \cdot E_{(s_t)(j)} = \sum_1^{Tr_{(i)(s_t)}} E_{(s_t)(j)} = (\sum_1^{Tr_{(i)(s_t)}} (r_j \cdot B),\ \sum_1^{Tr_{(i)(s_t)}} (P_{Tr_{(s_t)(j)}} + r_j \cdot k_0 \cdot B)) = (Tr_{(i)(s_t)} \cdot r_j \cdot B,\ Tr_{(i)(s_t)} \cdot P_{Tr_{(s_t)(j)}} + Tr_{(i)(s_t)} \cdot r_j \cdot k_0 \cdot B)$, for each $s_t \in \theta_{(i)(j)}$. Then, all these addends are summed together, to obtain an encrypted value of the numerator of the Formula 1, denoted by $EN$, where $EN = \sum_{t=1}^{|\theta_{(i)(j)}|} addend_{s_t} = ((\sum_{t=1}^{|\theta_{(i)(j)}|} Tr_{(i)(s_t)}) \cdot r_j \cdot B,$
$\sum_{t=1}^{|\theta_{(i)(j)}|} (Tr_{(i)(s_t)} \cdot P_{Tr_{(s_t)(j)}}) + (\sum_{t=1}^{|\theta_{(i)(j)}|} Tr_{(i)(s_t)}) \cdot r_j \cdot k_0 \cdot B)$.

(5) $v_i$ can easily compute denominator of Formula 1, denoted as $D = \sum_{t=1}^{|\theta_{(i)(j)}|} Tr_{(i)(s_t)}$. Hence, from the step 4 $v_i$ has $EN = (D \cdot r_j \cdot B,\ \sum_{t=1}^{|\theta_{(i)(j)}|} (Tr_{(i)(s_t)} \cdot P_{Tr_{(s_t)(j)}}) + D \cdot r_j \cdot k_0 \cdot B)$. Then, it calculates the inverse value of the denominator, denoted $D^{-1}$, by a binary polynomial inversion calculation.
(6) Finally, $v_i$ calculates the encrypted weights $E_{(i)(j)}(\mu_{(i)(j)}) = EN \cdot D^{-1}$, by adding $EN$ to the same $EN$ value for $(D^{-1} - 1)$ times according to the additive property of $ECC$, where $E_{(i)(j)}(\mu_{(i)(j)}) = (r_j \cdot B,\ \sum_{t=1}^{|\theta_{(i)(j)}|} (Tr_{(i)(s_t)} \cdot P_{Tr_{(s_t)(j)}}) \cdot D^{-1} + r_j \cdot k_0 \cdot B)$ After the above steps, the resulting encrypted value $E_{(i)(j)}(\mu_{(i)(j)})$ is used for the next step of computing the trust token.

***Privacy-preserving Intersection Set Protocol.*** To retrieve intersection set $IS$ we make use of a privacy-preserving protocol exploiting binary ECC. The protocol ensures that participating nodes will only know their common neighbor nodes, without inferring any information on other no-common nodes.

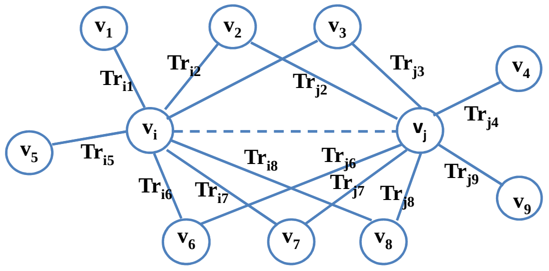


Figure 3: Common nodes of two adjacent nodes $v_i$ and $v_j$

It is assumed that each node in the network has its own identity $(ID)$, defined a pseudonym as well as an element in $F_2^m$. More precisely, given a node $t_i$, we denote by $P_{t_i}$ a point on $(E)$ corresponding to the $ID$ value of node $t_i$. Thus, we assume that each node $v_i$ has the set of ID points corresponding to its direct contacts, denoted as $ID_i = \{P_{v_x} | \exists (v_i, v_x) \in E\}$. Hence, given two nodes $v_i$ and $v_j$ the protocol computes $ID_z = ID_i \cap ID_j = \{P_{v_z} | \exists P_{v_x} \in ID_i, \exists P_{v_y} \in ID_j, P_{v_u} = P_{v_x} = P_{v_y}\}$. In this protocol, we mainly use the component $Y2$ of encryption of the points on $(E)$ (see section 2), as such, hereafter, we omit component $Y1$.

As a first step, $v_i$ encrypts each point $P_{v_x} \in ID_i$ with its public key $(k_i \cdot B)$, obtaining $Y2_{v_x} = P_{v_x} + r_i \cdot k_i \cdot B$, where $r_i$ is a random number generated by $v_i$. Hence only $v_i$ can decrypt these encrypted values and decode $P_{v_x}$ to know the real $v_x$ identity. Then $v_i$

[7]Let us denote a mapping function from a point $P_{W_i}$ on $(E)$ to a value $W_i$ in $F_2^m$ and vice versa as $P_{W_i} = f(W_i)$. We have $f(W_1) + f(W_2) = f(W_1 + W_2)$. In case $W_1 = W_2$, we have $f(W_1) + f(W_2) = f(2 \cdot W_1) = f(2 \cdot W_2)$
[8]In support of this, in the execution of this part of the protocol, we assume that the trust value are mapped from $[0, 1]$ to $[0, 100]$.

rearranges the order of these $|ID_i|$ encrypted values, obtaining a new set called $N_i$, by memorizing their original positions. $v_i$ sends $N_i$ to $v_j$. Once $N_i$ is received, $v_j$ encrypts values in $N_i$ with $v_j$'s public key $(k_j \cdot B)$ obtaining new encrypted values $N'_i$, where $Y2_{v_x} = P_{v_x} + r_i \cdot k_i \cdot B + r_j \cdot k_j \cdot B$, for each entry in $N'_i$. After that, $v_j$ encrypts each point $P_{v_y} \in ID_j$ with its public key $(k_j \cdot B)$, obtaining $Y2_{v_y} = P_{v_y} + r_j \cdot k_j \cdot B$, where $r_j$ is a random number generated by $v_j$. Then, $v_j$ rearranges the order of these encrypted values, by memorizing their original positions. The obtained set $N_j$ is sent to $v_i$ together with $N'_i$. To verify if these two sets have some common values, as first step $v_i$ encrypts $N_j$ with its public key. This results in a new set $N'_j$ that contains $Y2_{v_y} = P_{v_y} + r_i \cdot k_i \cdot B + r_j \cdot k_j \cdot B$, for each entry in $N_j$. Then, to verify if some common encrypted values exist between $N'_i$ and $N'_j$, $v_i$ iterates each possible pairs of $(Y2_{v_x} \in N'_i, Y2_{v_y} \in N'_j)$ by computing the subtraction $Y2_{v_x}$ - $Y2_{v_y}$. If this returns zero, it means that $Y2_{v_x}$, $Y2_{v_y}$ are the encryption of the same point. Indeed, $(P_{v_x} + r_i \cdot k_i \cdot B + r_j \cdot k_j \cdot B)$ - $(P_{v_y} + r_j \cdot k_j \cdot B + r_i \cdot k_i \cdot B) = 0$ only if $P_{v_x} = P_{v_y}$. Thus, once $v_i$ knows that $Y2_{v_x}$ is the encrypted value of a common node, it can easily retrieve the corresponding point $P_{v_x}$, as well as its identity $v_x$.

Let us consider the social graph in Figure 3. Node $v_i$ has the $ID$ point set of direct contacts $ID_i = \{P_{v_1}, P_{v_2}, P_{v_3}, P_{v_5}, P_{v_6}, P_{v_7}, P_{v_8}\}$, $|ID_i| = 7$ and $v_j$ has the ID point set of direct contacts $ID_j = \{P_{v_2}, P_{v_3}, P_{v_4}, P_{v_6}, P_{v_7}, P_{v_8}, P_{v_9}\}$, $|ID_j| = 7$. (1) $v_i$ encrypts all elements in $ID_i$ with its public key $(k_i \cdot B)$ and obtains $N_i = \{Y2_{v_x}/x = \{1, 2, 3, 5, 6, 7, 8\}\}$, $Y2_{v_x} = P_{v_x} + r_i \cdot k_i \cdot B$. $v_i$ rearranges the order of elements in $N_i$ and stores their original position. Then $v_i$ sends $N_i$ to $v_j$. (2) Similarly to node $v_j$, $v_j$ encrypts all elements in $ID_j$ with his/her public key $k_j \cdot B$, and obtains $N_j = \{Y2_{v_y}/y = \{2, 3, 4, 6, 7, 8, 9\}\}$, $Y2_{v_y} = P_{v_y} + r_j \cdot k_j \cdot B$. $v_j$ rearranges the order of elements in $N_j$ and stores their original position. Then, $v_j$ sends $N_j$ to $v_i$. (3) $v_i$ receives $N_j$ from $v_j$, then uses its public key $(k_i \cdot B)$) to encrypt elements in $N_j$ and gains $N'_j$, where $Y2_{v_x} = P_{v_x} + r_i \cdot k_i \cdot B + r_j \cdot k_j \cdot B$. (4) Similarly to $v_j$, $v_i$ receives $N_i$ and encrypts $N_i$ with its public key $(k_j \cdot B)$ and obtains $N'_i$ where $Y2_{v_y} = P_{v_y} + r_j \cdot k_j \cdot B + r_i \cdot k_i \cdot B$. (5) Node $v_i$ stores 14 encrypted values. $v_i$ traverses each element in $N'_i$ and each element in $N'_j$ and subtracts them to obtain $Y2_{v_x} - Y2_{v_y} = (P_{v_x} + r_i \cdot k_i \cdot B + r_j \cdot k_j \cdot B) + (P_{v_y} + r_j \cdot k_j \cdot B + r_i \cdot k_i \cdot B) = P_{v_x} + P_{v_y}$. If $v_x \equiv v_y$, then $Y2_{v_x} - Y2_{v_b} = 0$. Therefore, $v_i$ has a set of common $ID$ nodes which is $\{v_2, v_3, v_6, v_7, v_8\}$. (6) Similarly, $v_j$ has the same set of common $ID$ nodes with $v_i$, that is, $\{v_2, v_3, v_6, v_7, v_8\}$

***b) Trust token.*** As discussed before, the trust token has to contain the summation of the trust values $Tr_{(i)(i+1)} = \mu_{(i)(i+1)} \cdot Tu_{(i)(i+1)}$ of each edge connecting $v_i$ to $v_{i+1}$ in the relationship path, for each $i \in \{0, p-1\}$. To preserve the privacy of participant nodes, each value $Tr_{(i)(i+1)}$ must be encrypted before being aggregated into the trust token. This can be easily done since the execution of the proposed privacy-preserving protocol for Tidal Trust algorithm between $v_i$ and $v_{i+1}$ already returns the encrypted value of $\mu_{(i)(i+1)}$, that is, $E_{(i)(i+1)}(\mu_{(i)(i+1)})$. Hence, we only need to add $E_{(i)(i+1)}(\mu_{(i)(i+1)})$ to $E_{(i)(i+1)}(\mu_{(i)(i+1)})$ values for $(Tu_{(i)(i+1)} - 1)$ times. Thus, according to the additive homomorphic property of $ECC$, we obtain the encrypted product $Tr_{(i)(i+1)}$, denoted by $E_{(i)(i+1)}$. Then, we can aggregate this product $E_{(i)(i+1)}$ into the trust token.

*Trust token initialization.* For each direct node to which the token will be forwarded, i.e., $v_1$, the payer $v_0$ first executes the privacy-preserving Tidal Trust computation to obtain $E_{(0)(1)}(\mu_{(0)(1)})$. Then, it computes $E_{(0)(1)} = E_{(0)(1)}(\mu_{(0)(1)}) \cdot Tu_{(0)(1)} = \sum_1^{Tu_{(0)(1)}} E_{(0)(1)}(\mu_{(0)(1)}) = E_{(0)(1)}(\mu_{(0)(1)} \cdot Tu_{(0)(1)}) = E_{(0)(1)}(Tr_{(0)(1)})$. We denote this encrypted value as $E_{(0)(1)} = (r_{(0)(1)} \cdot B,\ \mu_{(0)(1)} \cdot Tr_{(0)(1)} + r_{(0)(1)} \cdot k_0 \cdot B)$, where $r_{(0)(1)}$ is a random number generated by $v_1$. Then, $v_0$ broadcasts $E_{(0)(1)}$ to its direct neighbors, i.e., $v_1$.

*Trust token computation at intermediate nodes.* Each intermediate node $v_i$ receives from $v_{i-1}$ the trust token. This contains $E_{(0)(i)}$, that is, the encrypted aggregation of trust values on the edges traversed so far. Similarly to the payer $v_0$, $v_i$ executes the privacy-preserving Tidal Trust computation so as to obtain $E_{(i)(i+1)}(\mu_{(i)(i+1)})$, and then to compute $E_{(i)(i+1)} = E_{(i)(i+1)}(\mu_{(i)(i+1)}) \cdot Tu_{(i)(i+1)} = \sum_1^{Tu_{(i)(i+1)}} E_{(i)(i+1)}(\mu_{(i)(i+1)}) = E_{(i)(i+1)}(\mu_{(i)(i+1)} \cdot Tu_{(i)(i+1)})$. We denote this encrypted value as $E_{(i)(i+1)} = (r_{(i)(i+1)} \cdot B, \mu_{(i)(i+1)} \cdot Tr_{(i)(i+1)} + r_{(i)(i+1)} \cdot k_0 \cdot B)$, where $r_{(i)(i+1)}$ is a random number generated by $v_{i+1}$. Then, $v_i$ computes $E_{(0)(i+1)}$, which is given as the sum between the received $E_{(0)(i)}$ and the just computed $E_{(i)(i+1)}$. It obtains: $E_{(0)(i+1)} = E_{(0)(i)} + E_{(i)(i+1)} = (r_{(0)(i)} \cdot B + r_{(i)(i+1)} \cdot B,\ (\mu_{(0)(i)} \cdot Tr_{(0)(i)}) + (\mu_{(i)(i+1)} \cdot Tr_{(i)(i+1)}) + r_{(0)(i)} \cdot k_0 \cdot B + r_{(i)(i+1)} \cdot k_0 \cdot B) = (\sum_{i=0}^{p-1}(r_{(i)(i+1)} \cdot B), \sum_{i=0}^{p-1}(\mu_{(i)(i+1)} \cdot Tr_{(i)(i+1)}) + (\sum_{i=0}^{p-1}(r_{(i)(i+1)}) \cdot k_0 \cdot B))$.

*Trust token validation.* The propagation ends when the tokenset reaches the payee $v_p$. This tokenset then is sent back to the payer $v_0$ that decrypts the tokenset content, i.e., $E_{(0)(p)}$ with its private key $(k_0)$. Thus, $v_0$ obtains the aggregated trust value $Tr_{(0)(p)} = \sum_{i=0}^{p-1}(\mu_{(i)(i+1)} \cdot Tr_{(i)(i+1)})$. As such, the payer $v_0$ obtains the final value as follows: $D(E_{(0)(p)}) = D(\sum_{i=0}^{p-1}(r_{(i)(i+1)} \cdot B), \sum_{i=0}^{p-1}(\mu_{(i)(i+1)} \cdot Tr_{(i)(i+1)}) + (\sum_{i=0}^{p-1}(r_{(i)(i+1)} \cdot k_0 \cdot B)) = \sum_{i=0}^{p-1}(\mu_{(i)(i+1)} \cdot Tr_{(i)(i+1)}) + \sum_{i=0}^{p-1}(r_{(i)(i+1)}) \cdot k_0 \cdot B) - \sum_{i=0}^{p-1}(r_{(i)(i+1)} \cdot B \cdot k_0)$. It results in:

$$Tr_{(0)(p)} = \sum_{i=0}^{p-1}(\mu_{(i)(i+1)} \cdot Tr_{(i)(i+1)})$$

The obtained $Tr_{(0)(p)}$ is still a point on $(E)$, $v_0$ needs to decode it to a value in $F_2^m$.

## 3 RELATIONSHIP TYPE COMPUTATION

We assume that the payer $v_0$ initializes the relationship type token, hereafter *rt* token, which is then forwarded only to those $v_0$'s neighbors with which $v_0$ has a relationship type equal to $rt_{TC}$. The token is initialized as the encryption of the sum of points $P_\tau$ and $P_{rt_{(0)(1)}}$ on $(E)$, respectively, corresponding to a random value $\tau \in F_2^m$ and the relationship type $rt_{(0)(1)}$ that is, the type in $rt_{TC}$. Each intermediate node $v_i$ adds to the content of the received *rt* token the encrypted point $P_{rt_{(i)(i+1)}}$ on $(E)$ corresponding to the relationship type of the edge to be traversed, that is, $e = (v_i, v_{i+1})$. This is done until the *rt* token reaches the payee $v_p$ that then sends the token back to the payer $v_0$. In turn, the payer $v_0$ decrypts *rt* token content and verifies if *rt* is compliant with `TC`. A detailed description is presented as follows.

*Relationship type token initialization.* The payer $v_0$ sets a random value $\tau \in F_2^m$ and encodes $\tau$ to a point $P_\tau$ on $(E)$. $v_0$ also encodes the relationship type $rt_{(0)(1)}$, which is also the one required in $rt_{TC}$, to a point on $(E)$ denoted by $P_{rt_{(0)(1)}}$. Then, $v_0$ uses its public key $(k_0 \cdot B)$ to obtain the encrypted value $E_{(0)(1)}$. As such, $E_{(0)(1)} = (Y1_{v_0}, Y2_{v_0}) = (r_{(0)(1)} \cdot B,\ (P_\tau + P_{rt_{(0)(1)}}) + r_{(0)(1)} \cdot k_0 \cdot B)$, where $r_{(0)(1)}$ is a random integer generated by $v_0$. After that, $v_0$ sends $E_{(0)(1)}$ to each of its direct neighbor, i.e., $v_1$, with which it has a relationship of type $rt_{TC}$.

*Relationship type token computation at intermediate nodes.* Once the intermediate node $v_i$ receives the *rt* token from $v_{i-1}$, it encodes $rt_{(i)(i+1)}$ to a point $P_{rt_{(i)(i+1)}}$ on $(E)$. Then, $v_i$ computes the encrypted value of $P_{rt_{(i)(i+1)}}$, that is, $E_{(i)(i+1)} = (Y1_{v_i},\ Y2_{v_i}) = (r_{(i)(i+1)} \cdot B,\ P_{rt_{(i)(i+1)}} + r_{(i)(i+1)} \cdot k_0 \cdot B)$, where $r_{(i)(i+1)}$ is a random integer generated by $v_i$. Then, $v_i$ aggregates $E_{(i)(i+1)}$ into the *rt* token content, i.e., $E_{(0)(i)}$, and obtains $E_{(0)(i+1)} = E_{(0)(i)} + E_{(i)(i+1)} = ((r_{(0)(i)} + r_{(i)(i+1)}) \cdot B,\ (P_\tau + P_{rt_{(0)(i)}} + P_{rt_{(i)(i+1)}}) + (r_{(0)(i)} + r_{(i)(i+1)}) \cdot k_0 \cdot B)$, where $P_{rt_{(0)(i)}} = \sum_{l=0}^{l=i-1} P_{rt_{(l)(l+1)}}$ is an aggregated point on $(E)$ corresponding to the aggregated relationship types on the edges $e = (v_{(l)}, v_{(l+1)})$ traversed from $v_0$ to $v_i$. Then, $v_i$ sends $E_{(0)(i+1)}$ to $v_{i+1}$. The propagation is iterated until the *rt* token reaches the payee $v_p$, then $v_p$ sends the *rt* token back to the payer $v_0$.

*Relationship type token validation.* Once the payer $v_0$ receives $E_{(0)(p)}$ from the payee $v_p$, $v_0$ decrypts $E_{(0)(p)}$ with its own private key $k_0$ and obtains $P_\tau + P_{rt_{(0)(p)}} = P_\tau + \sum_{i=0}^{p-1} P_{rt_{(i)(i+1)}}$. Then, $v_0$ subtracts the decrypted value to $P_\tau$ and gets $P_{rt_{(0)(p)}}$

Finally, if all types on the relationship path are equal, it means that: $rt_{(0)(p)} = \sum_{i=0}^{p-1} rt_{TC} = \lambda \cdot rt_{TC}$, where $\lambda$ is the number of hops to reach the payee from the payer or the depth through which the *rt* token propagates. We have the above equation due to the basic point scalar multiplication operator on $(E)$. Then, $v_0$ divides $rt_{(0)(p)}$ by $rt_{TC}$, by a binary polynomial division. If the division does not result in a remainder and the quotient is equal to the relationship depth extracted from the depth token, it means that all edges in the traversed path have the type equal to $rt_{TC}$. Otherwise, at least an edge has $rt \neq rt_{TC}$, so the trust condition is not satisfied.

## V FLOODING OPTIMIZATION

The path finding protocol described so far assumes that each intermediate node sends the tokenset to each of its direct contacts. However, that broadcast approach might not be suitable in a mobile network. Although the tokenset computation does not require plenty of resources (see experiments in Section 2), the broadcast approach might imply a high bandwidth usage since the number of tokensets that are exchanged rapidly increases (see experiments in Section 1). In order to cope with this issue, we introduce an optimization method aiming at reducing the number of propagated tokensets. An easy way for drastically reducing the number of tokensets is to let the intermediate nodes be aware of which type of relationship the payee is looking for. Given this information, they would be able to forward the tokenset only to the contacts with which they have the required relationship, rather than to broadcast to all their neighbors. However, this solution would reveal the required relationship type. In order to restrict the number of messages by at the same time protecting the required relationship type, we adopt a solution inspired by $k$-anonymity [13]. The basic idea is to let the intermediate nodes know a set, denoted by $A_{rel}$, of $k$ relationship types, $k-1$ of which are chosen randomly according to a uniform distribution from $RT$ whereas one is that required in `TC`. The flooding is then limited only to those edges labeled with a relationship type among the $k$ types in $A_{rel}$.

The $rt$ token is computed and validated as described in Section 3. Thus, when the payer $v_0$ obtains the final aggregated relationship type, it decrypts the value and verifies if `TC` is satisfied. Similarly to $k$-anonymity, the inference of the real relationship type required by a trust preference depends on the selected $k$ value. This value can be selected based on the domain requirements, trying to trade off between efficiency and privacy of trust preferences, since the smaller is $A_{rel}$, the better is the protocol performance, but the greater is the possibility to infer the required relationship type.

## VI SECURITY PROPERTIES

In this section we discuss the security properties ensured by the proposed protocol. As stated in Section 1, we are interested in protecting relationship information. In what follows, we describe which information can be inferred by nodes involved in the protocol. This discussion is done under the assumption that nodes are *honest but curios*, as such, they all perform steps required by the protocol, by trying to gain reserved information, if possible. The discussion is organized according to the node's role.

*Intermediate nodes.* We recall that intermediated nodes receive tokens containing encrypted aggregated information of path traversed so fa. Then, by exploiting $ECC$, they add the properties (i.e., trust, depth, relationship type) of the edge through which the token is going to be sent. According to token definitions, only $v_0$ is able to decrypt the aggregated path information, since all tokens have been encrypted with $v_0$'s public key. Moreover, $ECC$ keys used in $SmartPay$ have sizes larger than 160 bits [14], which meets the security conditions in cryptanalysis [3]. Additionally, the fact that a relationship path includes many of nodes implying the combination of the random big numbers makes adversaries hard to run a Bruce Force algorithm to analyze the tokens. This, even for the depth token that has alway value '1', since different nodes give different encrypted values with different random big numbers.

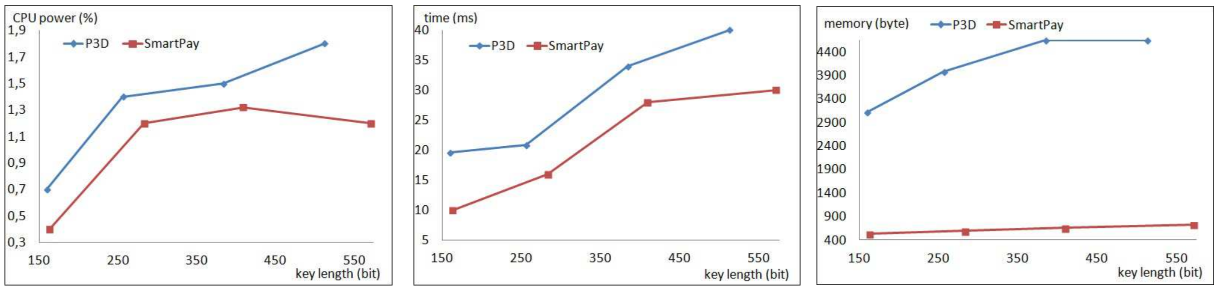
*Payee.* This node has only to send back to the payer the received token. Similar to intermediate nodes, it is not able to access aggregated data. According to honest-but-curios model, this node can neither generate nor add fake tokens to the received one, before sending it to payer.[9]

*Payer.* This is the only node able to decrypt aggregated data, as such obtaining the depth, type and trust of the traversed path $p$. These are all aggregated data, that do not reveal nodes that are involved in $p$, as well as, property of each edge. However, there is a particular case, where from the aggregated data payer can infer private information. This is the case of paths of depth 2. Indeed, in this case the path has a unique intermediate node, say X, which is also common friend of both payer $v_0$ and payee $v_r$. Thus, $X$ has an incoming edge from $v_o$, and an outgoing edge to $v_r$. $v_0$, once decrypted the token, is aware of the depth, that is, it is aware that one of its contacts has a direct relationship with $v_r$. It also knows that it is one to which the token has been sent, thus, the one with which $v_0$ has a relationship of $rt_{TC}$ type. However, it does not know exactly who, but it can guess with a probability $P(X) = \frac{1}{\|NC\|}$, where $\|NC\|$ is the number of direct contacts of $v_o$ with $rt_{TC}$ type. Larger is $\|NC\|$, smaller is the ability of $v_0$ to infer information about $X$. We have to note that, even if this is possible, this is not likely to happen as, in general, social network users have several hundreds of friends. As an example, on average, Facebook users have 130 friends. If we consider that SmatPay merges different user social network accounts, we believe that $\|NC\|$ ensures a low probability of guessing who is X.

## VII EXPERIMENTS

We carried out several experiments to test the proposed protocol. In particular, the first experiment illustrates the cost in terms of CPU, memory and time of token computation. Then, we present the performance of the optimizing flooding strategy introduced in Section 2. The physical resource used for the experiments is a PC configured with Core2 Duo CPU 3GHz, 4.0GB DDRAM, 64 bit Windows 7

[9]It is important to note that even in case of not-honest behavior, adding fake edge properties will decrease the possibility of satisfying the trust condition $TC$. For example, dding a depth token increases the depth making thus more difficult to satisfy the length limit stated in $depth_{TC}$



a) CPU Power  b) Time  c) Memory

Figure 4: System resource consumption for creating a token on a node

Professional.

## 1 COMPUTATIONAL COSTS

In this section, we compare the costs of the proposed protocol with another decentralized path finding protocol, that is, $P^3D$ [15] (see Section VIII for more details) due to its similarity with our protocol. Technically, in order to aggregate path properties, $P^3D$ uses the cryptographic hash function $SHA-1, 2$ combined with either the property of logarithm function or Elgamal cryptographic algorithm. According to the experimental results of $P^3D$, the logarithm function is much faster than Elgamal. Therefore, we make a comparison between $SmartPay$ and $P^3D$ variant based on $SHA-1, 2$ and logarithm function. The key sizes of $SHA-1, 2$ are sequentially 160, 256, 384, 512 bits. In $SmartPay$, we use *Koblitz elliptic curves* (or, *anomalous binary curves*), which are one among the 15 recommended elliptic curves analyzed in [16]. These elliptic curves are based on the binary $F_2^m$ and the size of underlying fields are 163, 283, 409, and 571 bits in turn. (See more details in [16], [9], [10], [11]). Our first experiment consists in computing the tokenset, i.e., depth, trust and rt token, tracing the system resources usage, i.e., CPU, memory, and time. The tokenset has been computed 10 times. The final resources usage is then given as the average of these 10 executions.

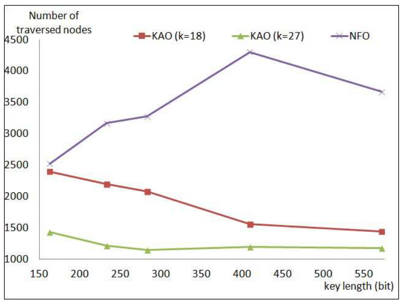
Figures 4-a), 4-b), and 4-c) show the comparison in terms of CPU power, memory, and time usage between $SmartPay$ and $P^3D$. From experimental results, we can easily see that resource usage of $SmartPay$ is less than $P^3D$. With the largest key size of $SmartPay$ (571 bits) and $P^3D$ (512 bits), $P^3D$ used 1.8% CPU power w.r.t. 1.2% of $SmartPay$ (see Figure 4-a). Meanwhile, the space for $P^3D$ is 4640 bytes while $SmartPay$ uses 728 bytes (see Figure 4-c). $SmartPay$ consumes 6.3 times less than $P^3D$, which is an interesting property considering the resource-constrained mobile domain. Moreover, the time spent on creating the tokenset with $P^3D$ is 40ms, 1.3 times more than $SmartPay$ that requires only 30 ms (see Figure 4-b). Let us consider the smallest key size of $SmartPay$ (163 bits) and $P^3D$ (160 bits): $P^3D$ used 0.7% CPU power while $SmartPay$ used 0.4%, 1.75 times more than $SmartPay$ (see Figure 4-a). Meanwhile, the space for $P^3D$ is 3120 bytes, while $SmartPay$ uses 536 bytes (see Figure 4-c). $P^3D$ consumes 5.8 times more than $SmartPay$. Time spent on creating a token with $P^3D$ is 19.6ms, 1.96 times more than $SmartPay$, i.e., 10ms (see Figure 4-b). With other key sizes, $P^3D$ also consumes resource much more than $SmartPay$. Consequently, we conclude that $SmartPay$ is more efficient than $P^3D$, especially in mobile environment.

## 2 FLOODING OPTIMIZATION

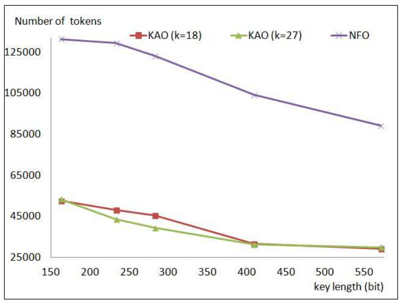
In order to verify the optimizating flooding strategy introduced in Section V, we make experiments on an existing social network dataset. In particular, we used the Epinions, dataset[10] which has been extracted from a Who-trusts-whom online social network.[11]

This is a dataset representing a directed social graph with $75,879$ nodes and $508,837$ edges. In order to perform the experiments, we modified the dataset so as to associate with each edge a relationship type and a trust value. The relationship type is uniformly and randomly picked up from [17] based on the ontological FOAF vocabulary specification which provides a set of 35 relationship types. A trust value set on an edge $e \in E$ is also randomly generated uniformly

[10]http://snap.stanford.edu/data/
[11]http://www.epinions.com/



a) Number of traversed nodes per second



b) Number of flooding tokens per second

Figure 5: A comparison between NFO and KAO

from a range $[0,1]$.

In order to measure the effects of the flooding optimization, we execute the protocols several times by tracing: the *number of traversed nodes* and the *number of generated tokens*. We execute the protocol according to the broadcast technique, i.e., *no flooding optimization - NFO* and the *k-anonymity based optimization - KAO* with different $k$ values. In particular, in choosing the value $k$ we have considered that if this $k$ is too large, the expected KAO performance is close to the one of NFO. In contrast, if $k$ is too small, the relationship type can be easily inferred. To set a good trade-off we have selected $k = 18$ and $k = 27$, that is half and 3/4 of available relationship types in [17]. Besides, we use the key sizes of 163 bits, 233 bits, 283 bits, 409 bits and 571 bits. Figures 5-a), 5-b) present the comparison of the flooding methods in which the $x$ axis represents the key size of *ECC* algorithm; and the $y$ axis reports the number of traversed nodes in Figure 5-a), and the number of tokens flooding in the network in Figure 5-b).

In case of the smallest 163 bit key size, in NFO, the number of traversed nodes is 2522 whereas the number of relevant tokens is 131313. In contrast, KAO with $k = 18$ traverses 2393 nodes (1.05 times less than NFO) and 52377 relevant tokens (2.51 times less than NFO), with $k = 27$ traverses 1424 nodes (1.77 times less than NFO) and 53230 relevant tokens (2.47 times less than NFO). Now let us consider the largest key size (571 bits), NFO generates 89139 tokens flooding through 3669 nodes. Meanwhile, KAO ($k = 18$) generates 28972 tokens (3.08 times less than NFO) going through 1437 nodes (2.55 times less than NFO), while KAO ($k = 27$) method generates 29799 tokens (2.99 times less than NFO) going through 1175 nodes (3.12 times less than NFO). With the other key sizes, the experimental results also prove that KAO is completely effective.

## VIII RELATED WORK

This paper is related to work in support of Person-to-Person payment systems. In this area, it is possible to find very interesting initiatives, like Zopa[12], Ratesetter[13] and Funding Circle[14] who support person-to-person initiatives. Bitcoin [18] is another decentralized e-cash system, addressing some privacy issues [19] due to only pseudonym usage but the transaction log is public. Consequently, Zerocoin [20] proposed a crypto-based solution to ensure anonymity of Bitcoin transactions. This is achieved by using RSA cryptography, with 3072 bit as key size at least. Although RSA is a strong cryptography asymmetric algorithm, it is not a compatible candidate for the mobile environment. Another notable example of P2P payment system is given by Ripple [21], which is a protocol for an open source payment system. However, Ripple and all above mentioned proposals do not allow users to specify trust preferences.

A further area related to the research conducted in this paper is the one of decentralized path discovery in social networks. With respect to this issue, some interesting proposals have been already published. For instance, [22] proposed a decentralized solution for path discovery based on public key cryptography, but the user triggering the path discovery can learn all properties of each relationship between two nodes in the path. To overcome this drawback in [23] authors enhanced the protocol with hormomophic encryption so users are able only to access aggregated value of relationship properties. In [24], two indirect nodes can compute the depth of all the paths between

[12]http://zopa.com/
http://uk.zopa.com/about-zopa/peer-to-peer-lending
[13]http://www.ratesetter.com/
[14]https://www.fundingcircle.com/

them without referring to other nodes. However, the protocol in [24] is only able to deal with discovery of paths with a specific length, whereas relationship type and trust are not considered. In addition, it is ineffective for a real large social network because of the initial required token flooding phase. In our previous work, a fully decentralized protocol $P^3D$ [15] for the social networks based on homomorphic cryptography has been proposed, solving the above mentioned problems. However, the main difference between the protocol proposed in this paper and $P^3D$, as well as all the above mentioned protocols, is that we make use of light cryptography techniques. As such the proposed protocol can easily be executed by mobile devices due to its more effective consumption of system resources, and faster performance. Moreover, to the best of our knowledge, this protocol is the first proposing a privacy-preserving approach for Tidal Trust computation in decentralized network.

## IX CONCLUSION

In this paper, we proposed a privacy-preserving path discovery protocol on support of trust preference enforcement in decentralized mobile payment systems. The protocol is able to protect information about the relationship type, depth and trust of the discovered paths. Future works include carrying out the experiments to show the efficiency of the proposed protocol w.r.t. the state of the art, investigating additional optimization strategies, handling offline nodes, and implementing a full *SmartPay* prototype to be integrated with existing mobile payment systems. In this step, we will make experiments on Android mobile devices, with simulation on ORBIT MANET.[15]

### References

[1] Gartner, "Gartner says worldwide mobile payment transaction value to surpass usd 171.5 billion." http://www.gartner.com/newsroom/id/2028315, 2012.

[2] B. of Governors of the Federal Reserve System, "Consumers and mobile financial services 2013." http://www.federalreserve.gov/econresdata/consumersandmobilefinancialservicesreport-201303.pdf, 2013.

[3] V. Katiyar, K. Dutta, and S. Gupta, "A survey on elliptic curve cryptography for pervasive computing environment," *Int. Journal of Computer Applications*, 2010.

[4] J. Golbeck, "Personalizing applications through integration of inferred trust values in semantic web-based social networks," in *Semantic Network Analysis Workshop co-located with ISWC'05*, 2005.

[5] B. Carminati, E. Ferrari, and N. H. Tran, "Enforcing trust preferences in mobile person-to-person payments," in *IEEE International Conference on Information Privacy, Security, Risk and Trust*, 2013.

[6] M. Tebaa, S. E. Hajji, and A. E. Ghazi, "Homomorphic encryption applied to the cloud computing security," in *World Congress on Engineering (WCE 2012)*, 2012.

[7] E. Mohamed, S. El-Etriby, and H. Abdul-kader, "Randomness testing of modern encryption techniques in cloud environment," in *Informatics and Systems (INFOS)*, 2012.

[8] D. Brickley and L. Miller, "Foaf vocabulary specification 0.98," 2010. http://xmlns.com/foaf/spec/.

[9] B. King, "A point compression method for elliptic curves defined over $gf(2^n)$," in *Work. Theory and Practice in Public Key Cryptography*, 2004.

[10] B. King, "Mapping an Arbitrary Message to an Elliptic Curve when Defined over $GF(2^n)$," *Int. Journal of Network Security*, vol. 8, no. 2, 2009.

[11] P. Eagle, S. Galbraith, and J. Ong, "Point compression for koblitz elliptic curves," *Advances in Mathematics of Communication*, vol. 5, no. 1, 2011.

[12] G. Liu, Y. Wang, and M. A. Orgun, "Trust transitivity in complex social networks," in *AAAI Conference on Artificial Intelligence*, 2011.

[13] L. Sweeney, "kanonymity: A model for protecting privacy," *Int. Journal on Uncertainty, Fuzziness and Knowledgebased Systems*, vol. 10, no. 5, 2002.

[14] A. Lenstra and E. Verheul, "Selecting cryptographic key sizes," *Journal of Cryptology*, vol. 14, 2001.

[15] X. Mingqiang, B. Carminati, and E. Ferrari, "P3d - privacy-preserving path discovery in decentralized online social networks," in *IEEE COMPSAC*, 2011.

[15] http://www.orbit-lab.org/

[16] D. Hankerson and A. Menezes, *Encyclopedia of Cryptography and Security (2nd Ed.)*, ch. NIST Elliptic Curves. searching, 2011.

[17] I. Davis and V. J. E., “A vocabulary for describing relationships between people.” http://vocab.org/relationship/, 2013.

[18] S. Nakamoto, “Bitcoin: A peertopeer electronic cash system.” http://bitcoin.org/bitcoin.pdf, 2009.

[19] E. Androulaki, G. O. Karame, M. Roeschlin, T. Scherer, and S. Capkun, “Evaluating user privacy in bitcoin,” in *Financial Cryptography and Data Security*, 2013.

[20] I. Miers, C. Garman, M. Green, and A. D. Rubin, “Zerocoin: Anonymous distributed ecash from bitcoin,” in *IEEE Symposium on Security and Privacy*, 2013.

[21] Ripple, “Ripple payment system.” https://ripple.com/.

[22] J. Domingo Ferrer, “A public key protocol for social networks with private relationships,” in *Modeling Decisions for Artificial Intelligence (MDAI)*, 2007.

[23] J. Domingo Ferrer, A. Viejo, F. Sebe, and N. Gonzalez, “Privacy homomorphisms for social networks with private relationships,” *Journal of Computer Networks*, vol. 52, no. 15, 2008.

[24] G. Mezzour, A. Perrig, V. Gligor, and P. Papadimitratos, “Privacy-preserving relationship path discovery in social networks,” in *Cryptology and Network Security(CANS)*, 2009.