Self-reported Verifiable Reputation with Rater Privacy

Rémi Bazin, Alexander Schaub, Omar Hasan, Lionel Brunie

Self-reported verifiable reputation with rater privacy

Rémi Bazin[1], Alexander Schaub[1], Omar Hasan[2], and Lionel Brunie[2]

[1] Department of Computer Science, École polytechnique 91120 Palaiseau, France
[2] University of Lyon, CNRS INSA-Lyon, LIRIS, UMR5205, F-69621, France

Abstract. Reputation systems are a major feature of every modern e-commerce website, helping buyers carefully choose their service providers and products. However, most websites use centralized reputation systems, where the security of the system rests entirely upon a single Trusted Third Party. Moreover, they often disclose the identities of the raters, which may discourage honest users from posting frank reviews due to the fear of retaliation from the ratees. We present a reputation system that is decentralized yet secure and efficient, and could therefore be applied in a practical context. In fact, users are able to retrieve the reputation score of a service provider directly from it in constant time, with assurance regarding the correctness of the information obtained. Additionally, the reputation system is anonymity-preserving, which ensures that users can submit feedback without their identities being associated to it. Despite this anonymity, the system still offers robustness against attacks such as ballot-stuffing and Sybil attacks.

1 Introduction

Reputation systems are very common on the Internet as they help the users learn about the quality of a product, document or other items of interest. Examples of reputation systems include the systems used on Amazon or Taobao. All these examples are based on centralized reputation systems, which implies that their security relies on the assumption that the underlying server is honest and secure.

Decentralized protocols (e.g. BitTorrent [1], Bitcoin [20]) have emerged for mainly two reasons: releasing the central server from resource consuming tasks to distribute these among the peers, or getting rid of the security dependency on the central server. Indeed, should a reputation protocol be centralized, a privacy disclosure such as the AOL search data leak in 2006 always remains a possible threat [4]. Neither are we safe from sponsoring i.e. increasing a certain entity's reputation in exchange for some fee – be it a public practice or a hidden activity. Although we usually trust well known entities such as the ones quoted above to behave honestly, we want to get rid of these trust requirements for a wider range of systems. These reasons justify our need for a decentralized scheme.

Another feature that we wish to provide is to preserve the anonymity of the raters. This choice is motivated by studies, such as the one on eBay [25], that show how sellers might discriminate against customers based on their previous

feedback. Two solutions arise to achieve this goal. The first one is to preserve the confidentiality of the rating values while making the list of raters for a specific vendor public. The other one is to hide everything but the aggregated reputation score by making the feedback entries unlinkable with the transactions and the identities of the customers. We will choose the latter proposition.

The protocol that we propose is also resistant against Sybil attacks [14]. These attacks consist of multiple fake identities or bots controlled by a single malicious user acting like legitimate clients in order to do ballot stuffing and send a high amount of either positive feedback values (self-promotion) or negative ones (bad-mouthing). We rely on blind signatures in our proposed protocol to achieve resistance against bad-mouthing attacks. Our scheme will also incorporate *tokens* as a way to prevent self-promotion.

A key contribution of the protocol is that the ratee himself stores the reputation values, yet the integrity of the reputation score is maintained. The querier is able to verify the integrity of the reputation score. This enables constant time retrieval.

The target application of our reputation system will be e-commerce: we will consider Service Providers (SPs) who want to sell goods, and clients who wish to buy the goods. The SPs will be the ratees i.e. the ones who receive ratings, whereas the clients will be the raters.

Our protocol fits into this e-commerce environment, while being both anonymity preserving and decentralized. It is based on Merkle trees [19], blind signatures [10] and non-interactive zero-knowledge proofs, and will be efficient (constant-time) when retrieving reputation.

The rest of the paper is organized as follows. Section 2 provides an overview of the state of the art concerning privacy-preserving reputation systems. Section 3 illustrates the model for the environment in which our protocol is to be used, while Section 4 highlights the objectives of our work. Our construct of *tokens* is described in Section 5. Section 6 describes the core protocol. An analysis of the protocol with regards to the previously defined objectives is presented in Section 7. Finally, we conclude in Section 8.

2 Related Work

Many privacy-preserving reputation systems have already been proposed in the literature. However, some of the papers in this domain use theoretical adversarial models that may not be appropriate for the real-world: for instance, the assumption that there will be no collusion among malicious peers is not realistic (e.g. [23]). Some other works are nonetheless more secure and resistant to small groups of malicious peers: the StR^M algorithm by Dimitriou et al. [12] and the Malicious k-shares protocol by Hasan et al. [17] are examples of such schemes. However, these protocols are rather confidentiality-preserving than privacy-preserving in the sense that they do not hide the list of users who participated in the rating.

Hence, we will focus on anonymity-preserving methods that completely hide the identity of the raters. Protocols of such type do already exist, but each one of them has some attributes that we want to avoid. The works of Androulaki et al. [3] and Petrlic et al. [24], for example, are instances of pseudonym based schemes. Nonetheless, these two require a centralized Trusted Third Party (TTP), and are thus not truly decentralized. The works of Anceaume et al. [2] and Lajoie-Mazenc et al. [18] on the other hand are more decentralized, but they rely on properties that we want to avoid: the first one prevents Sybil attacks by charging a fee, and in the second one, accredited signers are required to make resource heavy calculations for each rating of each SP. Even though this last recent contribution is very close to what we are looking for, we believe that our protocol succeeds better in distributing the computational costs among the different peers, notably by assigning the feedback records management to the specific service provider that is concerned. The work of Schaub et al. [26] is also decentralized and uses a blockchain to attain some similar objectives, but ballot-stuffing is still possible should the service provider be willing to pay fees for some additional custom feedback. Finally, the paper by Bethencourt et al. [6] illustrates a promising scheme based on signatures on published data. While this protocol is very interesting and secure, it has monotonic feedback, which allows an attacker to take advantage of his old good reputation without being affected by any new dissatisfaction that his recent activity might cause. We do however take inspiration from this work and use the same kind of zero knowledge proofs.

3 Model

The model we choose for our protocol is consistent with that of an e-commerce environment: we will consider a simple two-sided model where there are Service Providers (SPs) who sell goods and clients who buy them. We will only consider ratings provided by clients and destined for SPs.

Each transaction between a SP and a client should provide the client with a way to later post a feedback about the SP. The triggering event that enables a feedback to be sent should be the financial transaction itself. Moreover, only a single feedback may be valid per user per SP to prevent ballot-stuffing.

In our scheme, to maintain unlinkability between the client and the feedback, the feedback record would need to be sent by the client a certain amount of time after the transaction. This time-out may vary with the pace at which other clients' feedback is sent to the corresponding SP. Each user may be able to change this time-out privacy parameter according to his needs.

4 Objectives

Our objectives are to design a reputation system that is efficient, anonymity-preserving, decentralized and robust. The main novelty we propose is to ensure all of these contrasting properties in a single protocol. In the literature, we only

find protocols that fulfill a subset of these attributes ([2,3,6,12,17,18,23,24,26]), as discussed in the related work section.

4.1 Efficiency

Clients may need to browse through the list of a large number of SPs before choosing to transact with a specific SP. Therefore, the ability to quickly retrieve reputation values without overwhelming the network nor requiring excessive computation and latency is an essential requirement in a reputation system. The protocol must therefore ensure that it is efficient for the clients to retrieve the reputation value of a SP. As a matter of fact, we want to have a constant-time reputation retrieval procedure, which is uncommon in decentralized systems in the literature. Efficiency on the user side is a key advantage of the protocol that we aim to propose.

4.2 User anonymity preservation

Anonymity is achieved by maintaining two types of unlinkability:

1. **Transaction – rating unlinkability** The transaction itself may disclose the identity of the client, because of his shipping address for instance. This first kind of unlinkability consists in separating the transaction and the rating, which should be anonymous. However, we still want the transaction to enable the rating.
2. **Rating – rating unlinkability** It has been shown ([4,21]) that this second kind of unlinkability – between several ratings of a unique user – is also primordial to preserve the anonymity of the users.

We do not aim to hide the identities of the SPs in our protocol though. This means that they will be linkable to all their previous ratings. In the e-commerce context, this behavior is indeed often desirable.

4.3 Decentralization

Our objective is also to design a decentralized scheme. Security and privacy are better preserved in a decentralized environment in the sense that one does not have to rely on a single central entity that can become a single point of failure.

We do not exclude a Certification Authority if we want to use it as a way to prevent Sybil attacks. This authority shall nonetheless not have any other role in the protocol than giving certificates. Moreover, it may be offline most of the time since the only requirement is that it correctly delivers certificates.

4.4 Robustness

In our scheme, we place ourselves in a situation where peers may be malicious and colluding together. However, a majority of the peers that we call trackers is considered to be non-colluding honest-but-curious.

5 Tokens – security against Sybil attacks

In this section, we give a highlight of what the tokens are - a key building block that we use in our protocol. Their utility is to prevent Sybil attacks, and more precisely self-promotion, as highlighted in the introduction. They might be used in other contexts for protection against Sybil attacks in general. In that sense, their goal is to distinguish bots from real users.

In our protocol, the tokens are to uniquely identify a couple client/SP. Only the corresponding client should be able to generate such a token, and yet this token should not disclose any information related to his identity. One should therefore be able to tell if two tokens are issued by the same client or not, even though the anonymity of that client is preserved.

We also don't want other people to be able to reuse the token once the corresponding feedback record has expired. To achieve this, we include in each token a commitment to the one-time public key K that is used in the feedback records (see Section 6.2).

We design our tokens with a certificate-based implementation that is described below.

5.1 Certificate-based method

For this method of generating tokens, we assume the existence of a Certification Authority (CA), at least at some point. This authority might however go offline after delivering the certificates since only these ones are used. We leave the criteria required for admission up to the implementation.

In order to have identity-based tokens that are unlinkable with the identity of the user himself, we will use non-interactive zero-knowledge proofs of knowledge (NIZKs). The role of the NIZK proof is to check the hidden credentials of the client (both their integrity and the validity of the certificate from the CA) and to assert that the plaintext value *value* that is included in the token is uniquely identifying the client and the SP. Additionally, it should also contain a signature by the client on the one-time public key K of the feedback records (see Section 6.2) so as to prevent any subsequent use of the token.

5.2 Formalization

We denote CRED.VERIFY(pk, sk) to refer to the verification of a public and private key pair, CERT.VERIFY$(cert, pk)$ for the verification of a certificate *cert* about a public key pk, and SIG.VERIFY$(sign, M, pk)$ for the verification of a signature *sign* on the data M using the public key pk. TOKENIZE(v_{SP}, sk) will be a procedure that creates a token value uniquely identifying the SP v_{SP} and the client whose private key is sk.

Using the notation introduced by Camenisch and Stadler [9], we want to construct the following proof :

$$\text{NIZK}\left\{\begin{array}{l}pk_C, sk_C, cert : \\ \text{CRED.VERIFY}(pk_C, sk_C) \\ \bigwedge \text{CERT.VERIFY}(cert, pk_C) \\ \bigwedge \text{SIG.VERIFY}(sign, K, pk_C) \\ \bigwedge [value = \text{TOKENIZE}(v_{SP}, sk_C)]\end{array}\right\} \tag{1}$$

where the hidden variables pk_C, sk_C and $cert$ would respectively be the public key of the client, his private (secret) key, and the certificate from the CA validating his public key. The external values $sign$ and $value$ should be given alongside with the zero-knowledge proof. They are respectively a signature on the one-time public key K and the unique identifier for the couple SP / client. v_{SP} should be a unique and publicly-known identifier for the SP that is involved.

An implementation of such a NIZK proof is detailed in the technical report that extends this paper [5]. The NIZK model proposed by Groth et al. [16] is at the core of that implementation.

6 Description of the protocol

6.1 Outline

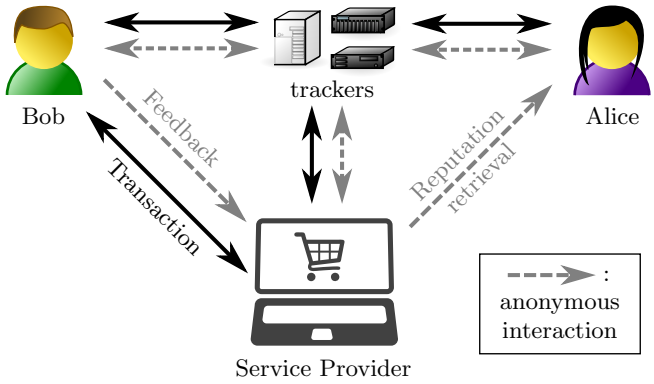

Fig. 1. Overview of the different entities and some primitive operations

Our protocol involves three kinds of nodes, as listed below and as shown in Figure 1.

– **Clients:** They are the ones who buy goods. Every user can be a client, assuming that they can produce tokens.

– **Service Providers (SPs):** They are the ones who sell goods. They are publicly registered. A SP is in charge of saving all the feedback records that are related to it. These form a "local blockchain" which is signed by the trackers, and made publicly available by the SPs.
– **Trackers:** They are a group of servers who are in the system mainly to control the good behavior of the SPs. Their role is therefore limited to the security and robustness of the scheme. We will minimize their involvement in the protocol in terms of resource usage.

The term *peer* will denote a computing unit that may be either of the three kinds of nodes above.

Below are brief summaries of the primitive operations in our reputation protocol, which are described in more detail in Section 6.3

– **Reputation retrieval:** The client obtains the reputation value of a SP from the SP itself, and verifies the trackers' signature.
– **Transaction:** The client asks for a blind signature from the SP while paying for his purchase. This will enable him to post a feedback record later. He also anonymously declares his purchase.
– **Sending feedback:** The client generates a feedback record from a token, the previous blind signature and his feedback value – which may also contain a comment. He sends it to the concerned SP who is required to include it in his next *block* of records (see Section 6.2).
– **Feedback aggregation:** The trackers periodically (e.g. once a day) sign the header of the next block, containing the current aggregated reputation value, so that the SPs can distribute it directly to the peers without any trust requirement between peers and SPs.

6.2 Setup

Trackers What we will call *trackers* are a group of several servers whose aim is to guarantee the security of the scheme. They fulfill this task by providing the following public information, which they can provide along with a time-stamp and a signature:

– The list l_t of all the current trackers.
– A hash table b_t^1 containing proofs of malicious behavior and / or proofs of intentional withdrawal of old trackers.
– A hash table b_t^2 containing for each SP a list – which is possibly empty – of proofs of bad behavior.

Anyone should not be able to become a tracker since the corruption of a majority of them threatens the security of the scheme. However, even if a few become corrupted, the security is still upheld as long as a majority of them is honest. Assuming that they have divergent interests (to avoid collusion), competition and fear of fraud discovery are good safeguards.

Feedback records A feedback record is comprised of a tuple $(d, v, c, \mathbf{t}, K, s_1, s_2)$ containing:

- d: Date of publication
- v: Feedback value
- c: Feedback comment (optional, may be empty)
- $\mathbf{t}$: Token (see Section 5)
- K: A one-time public key (part of a signature key pair)
- s_1: (Blind) Signature of the SP on K
- s_2: Signature on $(d, v, c, \mathbf{t}, K, s_1)$, verifiable with K

SP – Persistence of the records The SP is in charge of maintaining the data of its records, meaning the records that rate him. This is a fair task allocation since: the more feedback records a SP has, the more known he is and therefore the more computational resources we may reasonably ask him to deliver.

The records are to be kept in a special list of data blocks where each block contains the records data for a given time period T. T must not be too long (for adaptability to new feedback) nor too short (for efficiency reasons). We will take the compromise $T = 1$ day to simplify the description. Another parameter also drives the temporal aggregation function: the number n_t of periods – days – during which a given feedback record is valid. For a living duration of the feedback records of one year, for instance, we would have something like $n_t = 365$. In other words, it is the number of blocks that account for the current overall reputation value. The length of the list of blocks that the SP should save and publish should be $n_t + 1$ for verification purposes. Once a new block is added, the oldest one is discarded from the list, provided that the SP is at least $n_t + 1$ days old.

Each block is a tuple $(d, v_T, v_{tot}, h, s_3, data)$ where:

- d: Date of publication
- v_T: Aggregated reputation value over the latest period T
- v_{tot}: Aggregated reputation value over the period $n_t T$
- h: Hash of (SP, h', r_1, r_2, r_3) with SP being the identity of the SP, h' the hash of $data$ and r_1, r_2 and r_3 the root labels of the three Merkle trees $\mathcal{T}_1$, $\mathcal{T}_2$ and $\mathcal{T}_3$ that are detailed below
- s_3: Signature of the trackers on (d, v_T, v_{tot}, h)
- $data$: All the feedback records for this period T

This block is designed so that it can be sent without its $data$ element for any client to be able to retrieve and verify the current aggregated reputation value (with v_{tot}, s_3 and d).

In addition to these blocks of data, the SPs are required to maintain three Merkle trees. The first one $\mathcal{T}_1$ is to contain all the one-time public keys K that have been used so far in the blind signature scheme and published in feedback records. The second one $\mathcal{T}_2$ gathers all the currently used identities (i.e. token values). The third and last one $\mathcal{T}_3$ contains the identities of all the clients who

made at least one financial transaction with the SP, regardless of whether or not they posted a feedback record. These identities are tokens that have been generated based on a derived version of the SP domain name identifier SP, along with the date of the last transaction, committed inside the token. In this last Merkle tree, each node will also contain the number of leaves – i.e. identities – beneath it in addition to the usual hash of its children. That way, any peer can quickly retrieve the total number of buyers from the root of the tree, and verify it with r_3. The inspectors for the updates of this total number also benefit from the structure of this Merkle tree, because only the updated branches need to be verified.

6.3 Primitive operations

Reputation retrieval Each peer who wants to know the reputation of a SP just has to ask this SP for its reputation and the SP is expected to send back the signed data. The querying peer can then check the signatures of the trackers and retrieve the aggregated reputation value, as well as ask the SP for the rest of the block which contains the feedback records, i.e. the comments and individual feedback values.

As anybody can ask the SP for his reputation, clients have the choice to either ask him directly or use an anonymous connection such as Tor [13]. If he is asked directly, the query is no longer anonymous, but it is faster.

The main reputation retrieval procedure on the client side is detailed in Algorithm 1 below, which aims at retrieving the reputation of a SP SP at date d. The returned value is a tuple $(v_{tot}, header, s_4)$ where v_{tot} is the aggregated reputation we want, while $header$ and s_4 may be used for further analysis and data retrieval.

Algorithm 1 Retrieve the reputation of a SP

procedure RepRet(SP, d)
  if $(d > \text{today}()) \vee (d < \text{today}() - n_t T)$ **then**
    fail with Wrong date d
  if $\neg$ConnectTo(SP) **then**
    fail with Unable to connect to SP
  $(header, s_4) \leftarrow$ AskBlockHeader(SP, d)
  $(d', v_T, v_{tot}, h, s_3) \leftarrow header$
  if $(d \neq d') \vee \neg$CheckSig $(SP, s_4, header)$ **then**
    fail with Non-cooperative SP
  if $\neg$CheckSig (trackers, $s_3, (d', v_T, v_{tot}, h)$) **then**
    $p \leftarrow$ (RepRet, $(header, s_4), \varnothing$)
    Send (SP, p) to the trackers
    fail with Bad behavior
  return $(v_{tot}, header, s_4)$

Transaction The transaction proceeds in three steps:

1. The client generates a one-time couple of public and private keys for a signing scheme, the public key being called K. He asks the SP to blindly sign his public key K during the financial transaction (see Section 6.4).
2. He gives a token generated with his identity and a derived version $\widetilde{SP}$ of the SP identifier SP to the SP, so that it is included in the Merkle tree $\mathcal{T}_3$.

The client memorizes the keys and the blind signature so that he might use them later to publish a feedback value. In the corresponding Algorithm 2 executed by the client, SP is the SP with whom the client is to pay for a specific good, and *context* contains information about this good and the purchase in general. It uses the blind signature Algorithm 4: BlindSig to do the financial transaction in itself (see Section 6.4). It also uses the token creation scheme CreateToken that takes the identifier of the Service Provider and the date (to prevent reuse of the token) as arguments.

Algorithm 2 Make a transaction (client side)

procedure CTransaction($SP, context$)
  $(sk, K) \leftarrow$ KeyGen()
  $s_{SP} \leftarrow$ BlindSig($SP, K, context$)
  $d \leftarrow$ today()
  $\mathbf{t} \leftarrow$ CreateToken($v_{SP}||$".transaction", d)
  SP.TransactionToken($\mathbf{t}, d$)
  return (sk, K, s_{SP})

Sending feedback When a client wants to rate a SP, after having done a transaction with it, he proceeds as follows:

1. The client waits until the anonymity set of the SP satisfies him, which means until there are enough buyers for this client to remain sufficiently anonymous (k-anonymity with sufficiently large k).
2. He fills a feedback record with the public key K and the blind signature that were generated during the transaction, the value and the comment of the feedback itself, and a token (see Section 5). He then signs the record so that it can be verified with K.
3. He anonymously gives the record to the SP, and asks a signed commitment from the SP stating that he will include this feedback record in his next block.
4. He checks for its effective inclusion later on.

For the whole publication part, it is assumed that the client uses an anonymous connection. The wait duration before sending the feedback record may be randomized in order to prevent any relevant statistical analysis that would break anonymity. A deadline may be set for feedback dispatch as to limit the effect of rosy retrospection and prevent any "reputation lag attack".

Should the record not be included in the next block, the client can then send the signed commitment he received from the SP as well as the signed block which should have contained the record to the trackers. This data is in itself a proof of bad behavior which would then be appended inside the corresponding hash map entry in each tracker.

Should the SP even refuse to deliver the signed commitment when being sent the feedback record, the client also has the possibility to send his feedback record to the trackers so that they try themselves to get this commitment and send it back to the client. If the SP also refuses to them, the trackers build a proof of bad behavior based on that fact, which is signed by all the trackers – or at least a majority of them.

In the following Algorithm 3 that describes this procedure, the client calls SendFB with the identity of the concerned SP, the tuple that was returned by a previous call to the procedure CTransaction, the feedback value v to submit and a comment c that is possibly empty.

Algorithm 3 Send a feedback record

procedure SendFB($SP, (sk, K, s_{SP}), v, c$)
  Wait if necessary before proceeding.
  $d \leftarrow$ today()
  $\mathbf{t} \leftarrow$ CreateToken(v_{SP}, K)
  $s_2 \leftarrow$ CreateSig($sk, (d, v, c, \mathbf{t}, K, s_{SP})$)
  $rec \leftarrow (d, v, c, \mathbf{t}, K, s_{SP}, s_2)$ ▷ Feedback record
  $\alpha \leftarrow$ AnonymousConnection(SP)
  α.Send(rec)
  $C \leftarrow \alpha$.ReceiveCommitment()
  if $\neg$CheckSig($SP, C,$ (Received, rec)) **then**
    Send rec to a few trackers
    if They don't send back some valid C **then**
      fail with Bad behavior
  Wait (schedule the following) for the next day or later
  repeat
    $\alpha \leftarrow$ AnonymousConnection(SP)
    $(v_{tot}, header, s_4) \leftarrow \alpha$.RepRet($SP, d + 1$)
    $hinfo \leftarrow$ CheckHash($SP, header, s_4$)
    $data \leftarrow \alpha$.DatRet($SP, header, s_4, hinfo$)
  until Reputation retrieved or too many fails
  if $rec \notin data$ **then**
    $p \leftarrow$ (SFB, ($rec, C, header, s4, hinfo, data$), $\varnothing$)
    Send (SP, p) to the trackers
    fail with Bad behavior

Feedback aggregation In order to minimize the workload of the trackers, we want them to collectively only sign the header of each SP's new block for each period, so that the clients asking for the reputation of a SP directly ask the SP instead of asking the trackers. Since they only sign one block per period per SP, it is possible to check its integrity afterwards, and maybe create a proof of bad

behavior that will be validated thanks to this signature. Of course, we could also decide that the trackers verify it, in full or in part, before giving their signature.

The deterrence that are the proofs of bad behavior make it possible to increase the efficiency of the computation. Indeed, only a partial verification of the data should be sufficient to dissuade malicious SPs from misbehaving.

The trackers do need to check the hash h however, to ensure that this block header won't be used by another SP.

This scheme is designed so that the computation and verification of the reputation v_{tot} is made easier thanks to the daily values v_T. Indeed, to check a new v_T, one needs to go through all the records of the day. To check a new v_{tot} however, one should only need to take into account the previous total aggregated reputation (v_{tot}) of the day before, and the values v_T for the incoming and expiring days. Being able to calculate a reputation based on a previous reputation and aggregated new and old feedback values is the only requirement that we want for the aggregation formula. We leave the choice of this formula up to the implementation. A large number of frequently used aggregation formulas are consistent with the previous prerequisite, as is detailed in the extended version of this paper [5].

6.4 Blind signatures

Many algorithms exist for blind signatures: from the most well-known and simple one based on RSA cryptography [15] to other more complex ones [8, 22]. Some anonymous e-cash schemes may also be derived to be used as blind signature schemes. This is the case for the untraceable electronic cash by D. Chaum et al. [11], which is actually only a singular case of the RSA blind signature.

We face the following issue in implementing blind signatures for our scheme: how can we ensure that the blind signature is executed simultaneously with the payment? Indeed, should one of the two procedures terminate before the other, the one or the other of the two parties can stop the trade in the middle and use the half-trade to his advantage. If the blind signature finishes before the payment for instance, the user is able to rate the SP without even doing the trade (blind signature rendered useless). If it happens after the payment, the SP would be able to refuse the signing, and therefore the feedback. Even though it might not be much of a problem in this case, since refusing signatures means less feedback and less reputation for the SP, we still want to propose an alternative solution.

We detail below the implementation of a modified BLS blind signature that fulfills our requirements. The protocol, formalized in Algorithm 4, comprises of the following steps:

1. The client asks the SP to give a signature over the commitment that, should the signature of the client over the financial transaction be published, he is to blindly sign a specific masked message – $G^r M$.
2. Once he does so, the client can then safely sign the financial transaction.
3. The client asks the blind signature, and uses the signed commitment as well as the system of the trackers in case of bad behavior.

Algorithm 4 RSA Blind signature (client side)

procedure BLINDSIG($SP, K, context$)
  $T \leftarrow SP$.GETTRANSACTIONTOSIGN($context$)
  $(\mathcal{G}' = \mathcal{G}^e \in \mathbb{G}_1, \mathcal{H}' = \mathcal{H}^e \in \mathbb{G}_2) \leftarrow$ public key of SP for the blind signatures
  $r \leftarrow \text{random}(), \mathcal{M} \leftarrow \text{HASH}(K) \in \mathbb{G}_1$
  $\mathcal{M}' \leftarrow \mathcal{G}^r \mathcal{M}$
  $\tilde{T} \leftarrow SP$.GETCOMMITMENT($T, \mathcal{M}'$)
  if $\neg$CHECKSIG($SP, \tilde{T}$, (BLINDCOMMIT, $\mathcal{M}', T$)) **then**
    fail with Wrong commitment from SP
  $s_T \leftarrow$ SIGNTRANSACTION(T)
  $\bar{m} \leftarrow SP$.ASKBS(s_T)
  if $\bar{\mathcal{M}}^e \bmod n \neq m$ **then**
    Send $(T, \mathcal{M}', \tilde{T}, s_T)$ to a few trackers
    if They don't send back some valid $\bar{m}$ **then**
      fail with Bad behavior
  return $\bar{\mathcal{M}} \mathcal{G}'^r$

7 Analysis

Below is a brief analysis covering each one of the objectives we set in Section 4. A detailed analysis may be found in the extended version of the paper [5].

7.1 Efficiency

Our main goal regarding the efficiency aspects was to have a quick and light way of retrieving the reputation of a SP. This objective is achieved in our protocol because this operation operates in constant time in both network usage and computing power. Table 1 highlights how small the computations are to retrieve the reputation of a SP and to perform the other client-side operations, in the likely case that the SP does not misbehave. The only downside is the linearity with N – the number of feedback records of the day – for sending feedback, but this may be reduced to a logarithmic complexity by taking advantage of the Merkle trees.

	Computation time	Network payload	Network messages
Reputation Retrieval	2.7ms	140B	2
Transaction	46.3ms	1574B	4
Sending Feedback	33ms	2443B+N*1735B	12

Table 1. Computational cost of the different client operations

Assumptions for Table 1: Table 1 assumes SHA1 hashes (256 bits), ECDSA signatures based on the `ASN1::secp160r1` curve (336 bits) and AES-128 security level for pairings. The computational time has been measured on a computer with an Intel Core i3-5005U CPU, using the MIRACL and Crypto++ libraries. A back-and-forth interaction is counted as two network messages. All the costs of

the different operations have been measured in the best-case scenario where the SP does not misbehave. Also, the interactions that are not part of the protocol – e.g. getting to know the SP or doing the financial transaction in itself – are not accounted for. Finally, the computation time overhead generated by the network communication, be it anonymous or not, is not included.

The SPs are the ones who are required to do most of the computations, but that is not problematic since the amount of work delegated to them is proportional to their number of feedback values, that is to say to their popularity and presumably to their computational capacity.

7.2 User anonymity preservation

We see in the extended version of this paper [5] that under some reasonable assumptions on the security of the building blocks, we have:

Property 1 *The client can choose his anonymity set (or a lower bound of it) for a future feedback submission.*

Property 2 *The client / feedback unlinkability remains in agreement with the anonymity set defined by the client.*

Property 3 *The feedback / feedback unlinkability is guaranteed within the anonymity set chosen by the client.*

7.3 Decentralization

The only potentially centralized entity in this protocol is the CA. The presence of a CA to create the tokens is a necessity to prevent Sybil attacks (and more precisely self-promotion) if we want to avoid the use of fees. However, this CA could be comprised of several entities and thus decentralized, using a multi-signature scheme such as [7].

The trackers are also comprised of several distributed entities. Even though there has to be a limited number of trackers for the protocol to remain efficient, it is still to be considered as decentralized.

7.4 Robustness

Assumption 1 *If the peers have a list of N running trackers, at least $\lfloor \frac{N}{2} \rfloor + 1$ of them are honest.*

Assumption 2 *The CA only delivers certificates to individual and unique users.*

As explained in the long version of this paper [5], the following property holds given the two previous assumptions:

Property 4 *The reputation score that is retrieved by clients is the aggregated feedback from all the buyers and the few identities Ω_{SP} colluding with the SP, if this SP does not undertake any detectable malicious behavior.*

Two main deterrents that protect the protocol against any misbehavior of SPs are as follows:

- Misbehaviors such as always being offline, not providing reputation or not allowing transactions are detrimental to the SP itself.
- Misbehaviors such as refusing to publish negative feedback allow the client to forge a verifiable proof of bad behavior that can be escalated to the trackers.

8 Conclusion

In this article, we have presented a reputation system that is consistent with our objectives: efficient, anonymity-preserving, decentralized, and robust against various known attacks against reputation systems, such as ballot-stuffing and Sybil attacks. To the best of our knowledge, this is the only scheme in the state-of-the-art that achieves these attributes concurrently in a single protocol. We use Merkle trees and signed blocks of data to minimize the workload on the clients and trackers and to fairly distribute the record maintenance tasks to the service providers. Clients are able to retrieve the reputation of a given service provider in constant time. Despite the fact that the SPs are in charge of maintaining their own reputation records, the proofs of malicious behavior provided by the protocol deter them from acting maliciously. For future work, some improvements can be considered to further minimize the role of the trackers.

References

1. Bittorrent protocol, `http://www.bittorrent.com/`
2. Anceaume, E., Guette, G., Lajoie Mazenc, P., Prigent, N., Viet Triem Tong, V.: A Privacy Preserving Distributed Reputation Mechanism (Oct 2012)
3. Androulaki, E., Choi, S.G., Bellovin, S.M., Malkin, T.: Reputation systems for anonymous networks. In: Proceedings of the 8th International Symposium on Privacy Enhancing Technologies. pp. 202–218. PETS '08, Springer-Verlag, Berlin, Heidelberg (2008)
4. Barbaro, M., Zeller Jr, T.: A face is exposed for aol searcher no. 4417749 (August 2006)
5. Bazin, R., Schaub, A., Hasan, O., Brunie, L.: A decentralized anonymity-preserving reputation system with constant-time score retrieval (technical report). Cryptology ePrint Archive, Report 2016/416 (2016), `http://eprint.iacr.org/2016/416`
6. Bethencourt, J., Shi, E., Song, D.: Signatures of reputation. In: Proceedings of the 14th International Conference on Financial Cryptography and Data Security. pp. 400–407. FC'10, Springer-Verlag, Berlin, Heidelberg (2010)
7. Boldyreva, A.: Threshold signatures, multisignatures and blind signatures based on the gap-diffie-hellman-group signature scheme. In: Desmedt, Y. (ed.) Public Key Cryptography PKC 2003, Lecture Notes in Computer Science, vol. 2567, pp. 31–46. Springer Berlin Heidelberg (2002)
8. Camenisch, J., Koprowski, M., Warinschi, B.: Efficient blind signatures without random oracles. In: Blundo, C., Cimato, S. (eds.) Security in Communication Networks, Lecture Notes in Computer Science, vol. 3352, pp. 134–148. Springer Berlin Heidelberg (2005)

9. Camenisch, J., Stadler, M.: Proof systems for general statements about discrete logarithms. Tech. Rep. 260, Institute for Theoretical Computer Science, ETH Zurich (Mar 1997)
10. Chaum, D.: Blind signatures for untraceable payments. In: Chaum, D., Rivest, R., Sherman, A. (eds.) Advances in Cryptology, pp. 199–203. Springer US (1983)
11. Chaum, D., Fiat, A., Naor, M.: Untraceable electronic cash. In: Goldwasser, S. (ed.) Advances in Cryptology CRYPTO 88, Lecture Notes in Computer Science, vol. 403, pp. 319–327. Springer New York (1990)
12. Dimitriou, T., Michalas, A.: Multi-party trust computation in decentralized environments in the presence of malicious adversaries. Ad Hoc Netw. 15, 53–66 (Apr 2014)
13. Dingledine, R., Mathewson, N., Syverson, P.: Tor: The second-generation onion router. In: Proceedings of the 13th Conference on USENIX Security Symposium - Volume 13. pp. 21–21. SSYM'04, USENIX Association, Berkeley, CA, USA (2004)
14. Douceur, J.R.: The sybil attack. In: Proceedings of 1st International Workshop on Peer-to-Peer Systems (IPTPS) (2002)
15. Goldwasser, S., Bellare, M.: Lecture notes on cryptography (2001), page 235
16. Groth, J., Sahai, A.: Efficient non-interactive proof systems for bilinear groups. In: Smart, N. (ed.) Advances in Cryptology EUROCRYPT 2008, Lecture Notes in Computer Science, vol. 4965, pp. 415–432. Springer Berlin Heidelberg (2008)
17. Hasan, O., Brunie, L., Bertino, E., Shang, N.: A decentralized privacy preserving reputation protocol for the malicious adversarial model. IEEE Transactions on Information Forensics and Security 8(6), 949–962 (2013)
18. Lajoie-Mazenc, P., Anceaume, E., Guette, G., Sirvent, T., Viet Triem Tong, V.: Efficient Distributed Privacy-Preserving Reputation Mechanism Handling Non-Monotonic Ratings (Jan 2015)
19. Merkle, R.: A certified digital signature. In: Brassard, G. (ed.) Advances in Cryptology CRYPTO 89 Proceedings, Lecture Notes in Computer Science, vol. 435, pp. 218–238. Springer New York (1990)
20. Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system (2008)
21. Narayanan, A., Shmatikov, V.: Robust de-anonymization of large sparse datasets. In: Security and Privacy, 2008. SP 2008. IEEE Symposium on. pp. 111–125 (May 2008)
22. Okamoto, T.: Efficient blind and partially blind signatures without random oracles. In: Halevi, S., Rabin, T. (eds.) Theory of Cryptography, Lecture Notes in Computer Science, vol. 3876, pp. 80–99. Springer Berlin Heidelberg (2006)
23. Pavlov, E., Rosenschein, J., Topol, Z.: Supporting privacy in decentralized additive reputation systems. In: Jensen, C., Poslad, S., Dimitrakos, T. (eds.) Trust Management, Lecture Notes in Computer Science, vol. 2995, pp. 108–119. Springer Berlin Heidelberg (2004)
24. Petrlic, R., Lutters, S., Sorge, C.: Privacy-preserving reputation management. In: Proceedings of the 29th Annual ACM Symposium on Applied Computing. pp. 1712–1718. SAC '14, ACM, New York, NY, USA (2014)
25. Resnick, P., Zeckhauser, R.: Trust among strangers in internet transactions: Empirical analysis of eBay' s reputation system, chap. 6, pp. 127–157
26. Schaub, A., Bazin, R., Hasan, O., Brunie, L.: A trustless privacy-preserving reputation system. IFIP SEC - Privacy (2016)