A trustless privacy-preserving reputation system

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

[1] Ecole polytechnique, Palaiseau,
[2] LIRIS UMR 5205
INSA de Lyon, Campus de la Doua

Abstract. Reputation systems are crucial for distributed applications in which users have to be made accountable for their actions, such as e-commerce websites. However, existing systems often disclose the identity of the raters, which might deter honest users from posting reviews, out of fear of retaliation from the ratees. While many privacy-preserving reputation systems have been proposed, none of them was at the same time truly decentralized, trustless, and suitable for real world usage in, for example, e-commerce applications. After discussing the weaknesses and shortcoming of existing solutions, we will present our own blockchain-based trustless reputation system, and analyze its correctness and the security guarantees it promises.

1 Introduction

These days, reputation systems are implemented in various websites, where they are crucial for the customer experience. One of the first and best-studied systems in the e-commerce domain is the reputation system at `ebay.com` [40]. Its main objective is to help prospective customers to determine the trustworthiness of the sellers, and thus minimize the risk of fraud. Applications other than e-commerce also rely on reputation systems. For example, they are also used by online communities such as the StackExchange network in order to attest the trustworthiness of the members from those communities. Filesharing and other peer-to-peer applications also need reputation systems in order to incentivize users to participate in the network and avoid free-riding [27].

A study [40] showed that users may retaliate in case of negative feedback, and thus raters are less likely to provide negative feedback. In order to avoid this problem, several privacy preserving solutions have been proposed. Some of them try to hide the identity of the ratee [8,36,7,30,12], while others try to hide the rating [34,16,17,21,22,20,28] while making the aggregated reputation public.

While some of the existing privacy preserving reputation systems might be suitable for e-commerce applications, we observe that each one of them comes with its drawbacks. For example, Kerschbaum's system [28] has been specifically designed with e-commerce in mind. However, it is a centralized system, and thus can potentially be abused by the central authority. The system by Lajoie-Mazenc et al. [30] is decentralized and ratings are transaction-specific, therefore, it could be used in an e-commerce context. However, they do *not* consider that the identity of the customer might need to be revealed during the transaction. This is not very realistic, as the service provider will, in most situations, need the address of the customer to ship the purchased items.

Other schemes achieve anonymity even in this context, but are not trustless. One example of such a scheme would be the system proposed by Hasan et al [22]. Although it hides the rating submitted by each participant, the participants need to be able to quantify the trustworthiness of their peers (e.g. give an upper bound on the probability that a given peer will collude and try to help reveal their rating).

Given these considerations, we would like to achieve a *trustless reputation system*, i.e. one that does not require the participants to trust other users or entities to not disrupt the protocol or to breach their privacy. This privacy-preserving reputation model should be suitable for e-commerce applications, and we will therefore suppose that the identity of the customer is revealed during the transactions that they can rate.

Trustlessness seems hard to achieve without decentralization. One way to obtain decentralization is to use a distributed database in order to store the ratings submitted by the customers. We will achieve this using *blockchains*

The blockchain technology, which became popular thanks to the BitCoin protocol[31], has been used in various applications. Among these applications, we can count a domain name system (DNS) named Namecoin. The blockchain can be more generally seen as a public distributed database, with all the participants agreeing about its state in a secure manner. In BitCoin, for example, this database serves to store a ledger of the coins that each user owns, as well as the transactions between the users.

Anonymous reputation systems are a natural application for the blockchain technology. There have already been some attempts at building such systems [13], however, there seems to be no usable solution yet [4].

We will leverage this technology in order to achieve the objectives of our reputation system. It will enable us to build a truly decentralized system, that doesn't require the participants to trust other users, as the integrity of the rating-history can be verified by every user.

Our contribution

We propose a truly trustless, decentralized, anonymity preserving reputation system that is suitable for e-commerce applications. It is based on the blockchain technology, and will induce as little overhead as possible for the processing of transactions, while at the same time be robust and allow customers to submit ratings as well as textual reviews.

Outline

The rest of the paper is organized as follows. In Section Section 2, we will analyze existing privacy-preserving systems and explain in further detail why they are not suitable for e-commerce applications. In Section Section 3, we will explain the model used for our system, and list the properties that we want to achieve. Then, in Section Section 5, we will present our system, and in Section Section 6, we will describe the blockchain technology. Finally, we will explain in Section Section 7, why this system meets the expected goals and conclude in Section Section 8.

2 Related Work

Privacy preserving reputation systems have been studied in the literature for a long time. One of the first proposed systems was designed by Pavlov et al.[34] and uses primitives such as the secure sum and verifiable secret sharing. It protects the confidentiality of the feedback by hiding the values of the submitted ratings. These ideas were further developed by Dolev et al. [17], who introduced the use of new ideas such as homomorphic and commutative cryptography. However, these systems were only conditionally secure and not resistant against large numbers of colluding malicious adversaries (e.g. those who could deviate from the protocol). Hasan et al. [21,22] later introduced a system based on additive homomorphic cryptography and Zero-Knowledge proofs where the privacy of a given user can be preserved even in the presence of a large majority of malicious users. A little later, Dimitriou et al. [16] proposed two protocols with a similar architecture to the systems presented by Hasan et al., with slightly worse asymptotic complexity, however, less demanding in terms of resources for the querier (he has to relay less messages, verify less proofs, etc.).

Those protocols ([34,17,21,22,16]) are truly decentralized and the feedback is retrieved from the participants every time a querier wishes to learn the reputation of another participant. Therefore, all the nodes have to stay online in order to contribute to the reputation system, which is not suitable for e-commerce applications, but might be useful in other contexts, such as P2P applications.

Furthermore, these protocols are rather confidentiality-preserving than privacy-preserving in that they do not hide the list of users who participated in the rating. This way of partially hiding information leads to multiple issues linked to the mutability of the set of participating peers (the contribution of a user to the aggregated reputation might be revealed if the user goes offline between two reputation-queries.)

Hence, we will focus on privacy-preserving methods that completely hide the identity of the raters. Protocols of such type do already exist, however each one of them has its own weaknesses. The works of Androulaki et al. [8] and Petrlic et al. [36], for example, are instances of pseudonym based schemes. Nonetheless, these two require a Trusted Third Party (TTP), and are thus not truly decentralized. As the TTP has to be completely trusted for certain operations, its misbehavior could breach the privacy of the users or the correctness of the system. In Androulaki's proposal, for example, the TTP is the "bank" that handles transactions of the so called "RepCoins". The more coins one user owns, the higher his reputation. It requires the users to spend reputation in order to make other users gain some, which might be problematic, and is not trustless because the bank is considered trustworthy. A misbehaving bank could easily alter the reputation scores of any participant.

Anceaume, Lajoie-Mazenc et al. [7,30] proposed slightly different solutions. Instead of all the information about the reputation of the users being held by a single TTP, they distribute the trust using a DHT-structure (such as, for example, Koorde [26] or Chord [41]): every peer holds some part of the information, which allows to compute the reputation of a service provider. They also use witnesses for each transaction, which guarantee that the reputation submission will be correctly performed even if one of the two parties was to abort the protocol. Moreover, in their system, peers rate *transactions* between customers and service providers, rather than directly rating service providers. This seems more suitable for e-commerce applications, as one would typically rate every transaction made with a service provider, rather than periodically update their opinion on a given service provider. It also allows to introduce proofs of transactions, which guarantee (more or less) that only transactions that really took place can be rated. However, as the service provider creates those proofs, it is complicated to ensure that he doesn't generate proofs for transactions that did not happen, in order to submit positive reviews by himself and wrongfully increase his own reputation. Anceaume's and Lajoie-Mazenc's systems only offer little protection against these attacks. The system proposed in [30] also makes use of complicated zero-knowledge proofs and is thus quite costly to perform (several seconds for each participant, up to a minute in certain cases). ion authority that needs to be non-malicious.

Finally, Bethencourt et al. [12] present a scheme which uses signatures on reviews to guarantee their correctness. While the system is quite secure, it's usefulness is lessened by an important drawback : the system does not handle negative feedback. Service providers could advertise their past good reputation without being affected by recent customer dissatisfaction.

None of those protocols are trustless, and therefore need either the customers or the service providers (or both) to trust some entities not to tamper the system or to break privacy, without being able to verify that there is no bad behavior. This will not be needed in the system we will present.

3 Our model

3.1 Participants

For our system, we choose a model that is as close as possible to actual e-commerce systems. As stated in Section 1, we will consider two types of users : service providers (SP) who will sell goods or services, and customers who might buy them. The most important part in e-commerce systems is the rating of the service providers. Therefore, we will only consider ratings *from* the customers *about* the SPs. Only customers might be raters, and only SP will be ratees.

We will also suppose that the transaction will ***disclose*** the identity of the customer : the SP will need the customer's credentials, such as his credit card number or address, in order to process the order. Even if the transaction is done via an anonymous electronic currency such as `Dashcoin`[19] or `Zerocash` [11], the service provider will most certainly need the customer's address in order to deliver the good. We suppose that after every transaction between a customer and a SP, the customer might rate the SP.

More formally, we will introduce the following notations :

S The set of all the service providers (i.e. ratees)

C The set of all the customers (i.e. raters)

P The set of all the participants, $P := S \cup C$. It is simply the set of all the nodes participating in the network.

B The blockchain. As the blockchain defines an ordered set of blocks, we will denote by $B[n]$ the n-th block in the chain, $B[0]$ being the genesis block. In the case of a fork, two block-chains B and B' might coexist for a certain time, however, a given chain and a given position define an unique block. A block is simply a set of operations that are aggregated for maintenance reasons (it is more efficient to store them this way). The blockchain can also be seen as a database whose state will be the initial state (that is hard-coded) on which all the operations contained in the blocks $B[0]$ to $B[n]$ are applied.
Every time a new block is constituted, an award will be payed to the user that constituted it. This works in a similar fashion as in the so-called "alt-coins". In our system, owning coins is mandatory in order to be allowed to receive reputation. It also helps preventing spam and other kinds of attacks (as described in section 7.3).

A The set of all the addresses of the participants. These addresses will be used for maintenance. Every service provider will own one address. They will be used, in particular, to hold and spend the coins generated by the blockchain, but also to identify the service providers.

Service providers will have a unique address, as issuing reputation tokens will cost coins and owning an address is necessary in order to own and transfer them. As a service provider will not gain anything from having more than one address (see section 7.3 for more details), there is no need to try and enforce this policy. Furthermore, it would be complex to enforce it in a decentralized fashion.

The concept of *addresses* is bound to the block-chain system, while the concept of *participants* (customers and service providers) is bound to the reputation system built on top of the block-chain. For $s \in S$, we will denote by $A(s)$ the corresponding address, and for an address $addr$, we will denote by $S(addr)$ the corresponding service provider if any (regular customers could own addresses, in order to win coins by helping to maintain the blockchain).

3.2 Operations

We will next describe the functions that are needed in our system. The protocols that implement these functions will be described in the later sections. Most, if not all, of these functions will be performed with respect to a given blockchain B, or need to make calls to a random number generator (RNG). These are implicit inputs of the protocols.

For the customer These operations will be performed by a certain customer $c \in C$.

- $\mathsf{setup}()$
  Generates a new public key, usable by the customer, for the transaction.
- $\mathsf{get\_reputation}(s)$
  Allows the customer to query the reputation of a service provider $s \in S$.
- $\mathsf{get\_token}(s, x)$
  Allows the customer to request a token that will prove that he was engaged in a transaction x with the service provider s. Outputs a blinded token $\bar{t}_x$.
- $\mathsf{unblind\_token}(\bar{t}_x)$
  Unblinds the token that was retrieved using the `get_token` protocol. Outputs an unblinded token t_x. The token is bound to the transaction. However, given two tokens t_x and $t_{x'}$, the service provider s will not be able to tell which token belongs to which transaction.
- $\mathsf{publish\_review}(s, t_x)$
  Allows the customer to publish a review about the service provider $s \in S$, using the token t_x previously unblinded.

For the service provider This operation will be performed by the service provider.

- $\mathsf{issue\_token}(c, x)$
  In response to a `get_token` request from the customer $c \in C$, issues a blinded token $\bar{t}$ and sends it to the customer, if the customer is entitled to receive one, i.e. he was really engaged in the transaction x.

Block-chain related operations These operations are mostly independent from the underlying reputation model. However, a blockchain mechanism is needed in order to store the reputation values in a reliable way. These operations can be performed by any node in the network.

- $\mathsf{broadcast}(op)$
  Broadcasts an operation op (which can be a review, a transaction, etc.) to all the nodes running the protocol.
- $\mathsf{compute\_balance}(s)$
  For a service provider $s \in S$, representing an address $addr$, computes the balance (in terms of coins) associated with this service provider.

– create_new_block($addr, b$)
  Broadcasts a newly mined block b. It will contain, among other data, reviews and transactions, but also a proof that *addr* has the right to constitute the next block.

The incentive for creating blocks are the *coins*. Every address who correctly creates a block and broadcasts it will receive a certain amount of coins. They serve as a currency within this system, in a similar fashion as in all the cryptocurrencies that have been developed since BitCoin. However, in our system, the main usage of the coins is *not* to serve as an alternative currency. Rather, they will be needed by the service providers in order to have the right to deliver tokens. This also serves to limit spam from the service providers, who could simply create as many tokens as they desire in order to boost their reputation.

3.3 Adversarial model

We consider a malicious adversarial model with collusions. This model implies that any participant in the protocol may behave arbitrarily and deviate from the protocol at any time as deemed necessary. Service providers may want to learn the identity of the customers that rated them, they might try to raise their own reputation, and collaborate with other service providers. Customers may try to submit reviews without having previously interacted with service providers, might try to use the received token in order to rate other service providers, or might try to otherwise disrupt the service.

We will also suppose that there might be attempts to disrupt the blockchain, such as forking in order to confuse new participants.

3.4 Objectives

The objectives for our system are the following :

– *Trustlessness*
  In an e-commerce system, we cannot expect customers to have pre-existing trust towards other customers of the same SP. Therefore, our system should not suppose that there is pre-existing subjective trust between users. No customer should have to trust any entity *not* to deviate from the protocol in order to break its privacy or change its rating. The protocol should ensure that privacy and correctness are preserved even if other participants deviate from the protocol. Therefore, it should also not rely on Trusted Third Parties, or Certification Authorities, which, by definition, must be trusted to behave faithfully.
– *Suitability for e-commerce*
  As the identity of a customer will be most certainly revealed during a transaction, the system should enforce the unlinkability of transactions and ratings, i.e. for a given rating, it should not be possible to determine which transaction it is related to (it should however be possible to identify the related

SP). It should, however, not be possible for a customer to submit a rating if no transaction took place.

- *Decentralization*
  We want to avoid any central point of failure as well as any single point of control. Therefore, the system should not depend on one, two, or a small number of nodes in order to work properly. We will even exclude Certification Authorities, because they have proven unreliable in the past, either because they became subject to attacks [1,2] or because they issued themselves fraudulent certificates [3], and because they would induce some centralization aspects in the system.
- *Anonymity preservation*
  The anonymity of the customers should be preserved. More precisely, the ratings and the identities of the customers should be unlinkable, as well as the ratings among themselves. The later kind of unlinkability is also crucial to preserve the anonymity of the users, as highlighted in [32] and [9].
- *Robustness*
  Our system should be robust to classical attacks on reputation systems, in particular bad-mouthing, ballot-stuffing, Sybil attacks [18] and whitewashing.

4 Building blocks

In order to be able to build this protocol, we will need two basic building blocks : the blockchain and blind signatures.

4.1 Blockchain

A blockchain can be seen as a distributed, public database, which can be read by every user running the appropriate program, but on which writing has a cost, or cannot be done at any time by any user. Every action that modifies this database is broadcasted among all the users in the network, and they are recorded as “blocks”. The creation of those “blocks” is controlled by mechanisms that vary between the different blockchain algorithms, and the state of the database is the sum of all the actions in all the blocks at a given moment in time. This concept has become popular due to the BitCoin currency [31], which seems to be the first application making use of this idea. Since then, several alternative coins have been launched, using similar concepts. Those first blockchain-systems are called “Proof-of-Work” blockchains, because, in order to write to the public database, you must prove that you did a certain amount of work, in the form of a partial collision using hash functions. This has led to a ”race of arms”, with hardware dedicated for BitCoin mining becoming quite popular [35].

An other family of blockchain-based systems was born with PPCoin [29], and further developed by BlackCoin [42], NXT [6] and NeuCoin [15]. The Proof-of-Work is replaced with a “Proof-of-Stake”, meaning that every participant randomly gains the right to write to the public database, the probability of gaining this right being proportional to his “stake”, i.e. the amount of coins he owns. This makes dedicated hardware useless, as additional computing power doesn’t allow any participant to gain more coins, and it is therefore less costly to run such a protocol. Some authors argue that because of some inherent problems about Proof-of-Stake, such as the low cost of “forking” the blockchain (i.e. creating two different histories of the public database), it is impossible to maintain distributed consensus on the state of the public database using Proof-of-Stake in most situations [37,23]. However, these issues have been tackled in detail, notably by the NeuCoin whitepaper [15], which itself takes most of its ideas from PPCoin and BlackCoin. Therefore, maintaining a distributed consensus using Proof-of-Stake *does* seem possible today.

4.2 Blind signatures

A blind signature scheme is a protocol in which the signer of a message doesn’t learn anything about the content of the message that was signed. We expect from such a system the following properties :

Unforgeablility
: The signature cannot be falsified (only the user knowing some secret information, such as a private key, can issue valid signatures).

Blindness
: The signer does not learn anything about the message it signs (given the information available to the signer, all possible messages are equally likely to be about to be signed).

Untraceability
: Once the message and signature have been revealed, the signer cannot determine when the message was signed.

For example, the blind signature scheme proposed by Okamoto [33], based on bilinear pairings, could be used to instantiate this primitive, or the simpler version based on the RSA algorithm, first proposed by Chaum [14]. As Chaum’s version is simpler and faster, we will use this scheme in order to explain our protocol.

5 Specification of the protocol

5.1 An Overview

The proposed protocol could be summarized as follows :

1. Before contacting the service provider in order to perform a transaction, the customer may compute the service provider’s reputation using the get_reputation protocol.
2. Once the customer retrieved the reputation of the service provider, he decides whether to engage in a transaction with the SP or not.

3. If the customer decides to engage in a transaction, before a transaction takes place, the customer creates a new *public key*, derived from a private/public key pair, for the process. This key should be kept secret from the service provider for the moment. It will be used to avoid token-theft (only the person knowing the private key associated with this public key will be able to use the token that will be issued by the service provider, even if it is intercepted by an other participant). Then, the transaction takes place : for example, the customer sends the money, and the SP starts to deliver the good.
4. Just after the transaction takes place, the customer asks the SP for a *blinded token* by performing the get_token protocol (which takes the freshly generated public key as input), and verifies that the SP has a sufficient balance for issuing a token (using the compute_balance protocol). The balance should be greater than some n coins, since n coins will be deduced from the SP's account when the review will be integrated in the blockchain. The customer then verifies the token (i.e. verifies that the signature is correct) and unblinds it for later use with help of the unblind_token protocol.

   Requiring some coins to be spent in order to receive a review helps to prevent ballot- stuffing attacks, as the SPs may, theoretically, issue an unlimited amount of tokens to themselves and could therefore submit an unlimited number of positive reviews for themselves. As for the token, it serves as a proof that a transaction really occurred. It therefore helps to greatly reduce the risk of bad mouthing attacks. It has to be blinded, so that the service provider cannot link the token, and therefore the rating, to the transaction and the identity of the customer.
5. Once the customer is ready to review the SP, he will broadcast a message containing the address of the SP, the token, along with the rating of the transaction and (optionally) a written review, a signature on this information, as well as a pointer to the last review concerning the same service provider. This is done via the publish_review protocol. The pointer enables any participant to compute the reputation of a given service provider much faster (instead of the whole blockchain, only a limited number of blocks have to be scanned in order to get all the reports concerning the SP). While the verification of this pointer may cause some computational overhead, it is not much more costly than, for example, to verify in the BitCoin protocol that the output of a transaction was not double-spent, which is the most basic feature one can expect from a cash system.
6. A participant wishing to earn coins (in order to be allowed to grant tokens for example) verifies if he is allowed to constitute the next block. If it is the case, he will run the constitute_block protocol. The creation of blocks helps to maintain a unique history of actions, avoids double-use of tokens, and is incentivized through the reward in coins.

In the next sub-sections, we will describe the protocol in more detail.

5.2 Public key creation

Before the transaction takes place, the customer creates a new public key that will be used for one transaction only (similar to what is recommended for BitCoin addresses for example). This will be the public part of an ECDSA key [24] (note that a deterministic ECDSA scheme can be used, see [39]). It must not be communicated to the SP during the setup phase. An outline of the `setup` protocol could look like follows :

Algorithm 1 Setup protocol

```
1: procedure SETUP(T)
2:     $(p, a, b, G, n, h) \leftarrow T$ // Those are the parameters of the elliptic curve used for
       ECDSA, for example those of secp256k1
3:     $privKey \leftarrow rand(0, n)$
4:     $pubKey \leftarrow privKey * G$
       return $(pubKey, privKey)$
```

5.3 Blinded token exchange

Before the transaction takes place, the customer will receive a token from the SP that will guarantee that its review will be accepted. For this purpose, the customer hashes the previously generated public key and requests a blind signature on this, for example using Okamoto's provable blind signature scheme (in the complete, not partial blinding setup), or the much simpler Chaum's blind signature algorithm. This will make the token unlinkable to the transaction, and therefore guarantee the anonymity. The customer will also check that there are enough coins in the wallet associated with the SP. Then, the transaction can take place.

If Chaum's blind signature is used, then we can define the three protocols get_token(s), unblind_token($\bar{t}$) and issue_token(c) as follows :

In order to perform these protocols, the customer will need to learn, by some means, the service provider's public key. Therefore, the public key must be known to all participants. One possibility would be to include in the blockchain the association between a service provider and a public RSA key used for the blinded tokens. This would, however, make the system vulnerable to Denial Of Service attacks (in which one attacker would simply generate a lot of those associations and broadcast them). Requiring a fee (in terms of coins generated by the system) might help to reduce the risk of DoS attacks. An other solution would be to get the public RSA key by extraneous means (for example, using the service provider's homepage) and ask for an ECDSA signature on the public key. The public key, along with the ECDSA signature, could then be published along with all the verification operations involving the RSA key. As there only

Algorithm 2 Token exchange

```
1: procedure GET_TOKEN($s, pubKey, e, n, x$)  ▷ $(e, n)$ is the service provider's public RSA key pair, $x$ the identifier of the transaction
2:     $m_1 \leftarrow hash(pubKey)$  ▷ $hash$ is a cryptographic hash function such as sha256
3:     $r \leftarrow rand(0, n)$
4:     $m_1 \leftarrow m_1 r^e \bmod n$
5:     $send((m_1, x), s)$
6:     return $(m_1, r)$
7: procedure ISSUE_TOKEN($c, m_1, d, n, x$)  ▷ $(n, d)$ is the service provider's private RSA key pair
8:     if verify($x$) then  ▷ the service provider has to specify verify
9:         $\bar{t} \leftarrow m_1^d \bmod n$
10:        $send(\bar{t}, c)$
11:        return $\bar{t}$
12: procedure UNBLIND_TOKEN($\bar{t}, r, e, n$)
13:     $t \leftarrow \bar{t} r^{-1} \bmod n$
14:     return $t$
```

needs to be one signature on the RSA key (so that every one can verify the association between RSA key and address), the service provider could also publish the signature along with the RSA key and its address on its homepage.

5.4 Broadcasting the review

Once the transaction is finished, the customer might want to wait for some time (so that he is not the SP's only customer for this period). After this period of waiting, he might choose a rating for this transaction (say, an integer in $[|0; 5|]$) and write a review about it. The review can give helpful information to prospective customers, explain a bad rating, and helps distinguishing between trustworthy and fake ratings. This information will be broadcast in the network, along with the identifier of the SP, the token and the signature on the token. This message will also contain the signature of the customer, and a pointer to the last review concerning the service provider. The message that will be broadcast can be represented as follows :

Depending on how the blind signature is implemented, two fields, containing the SP's RSA-key and the signature on this key, might be necessary.

The pointer should refer to the most recent, in terms of blockchain history, review on the service provider. It might happen that two (or more) reviews about the same SP have to be included in the same block. In this case, the creator of the block changes the pointer : he keeps one of them (arbitrarily chosen), pointing to a review in a previous block, and changes the second pointer to point towards the other review (which will be in the same block). If there are more than two reviews, the first one is unchanged, the second one points to the first one in the same block, the third one to the second, etc. The ordering is performed

Table 1: Structure of a broadcasted message containing a review

Field	**Description**
$addr_s$	Address of the SP
$pubKey$	The public key used for the transaction
$token$	Token obtained from the SP (i.e. blind signature on $pubKey$)
$rating$	Rating of the transaction (for example, an integer in $[0;5]$)
$review$	A textual review on the transaction (optional)
sig	Signature, using $privKey$, of $(adds_s\|\|pubKey\|\|token\|\|rating\|\|review)$
$pointer$	Pointer to the last review about the same SP

arbitrarily. In the same fashion, if the last review about a SP has been included in a block containing more than one review about the SP, then the first of the reviews in the new block should point towards the *last* review of the older block (given the ordering described earlier. The last review will be the one which is *not* referenced by any other review in this block). Also, before signing the new block, the creator of the block has to verify that there is no newer block containing an opened review about this SP. If there is one, he has to update the pointer.

5.5 Computing the reputation

In order to compute the reputation, a new customer only needs the last block containing a review about the SP whose reputation it seeks. Once this block has been found, it is sufficient to follow the pointers in order to retrieve all the reviews about this SP. For each review, the prospective customer might also verify the correctness of the blinded tokens. Then, the customer can choose any aggregation function he wishes (mean, median, or beta-reputation [25]), and also read the textual reviews for outlier ratings (especially high or, more probably, especially low ratings). Therefore, he has to download at most one block for each review.
In order to keep the system scalable, one can choose to ignore ratings that are older than a given age. Fewer blocks have to be downloaded in order to compute the reputation of a given SP, and, more importantly, in case of a new SP entering the system, instead of scanning the whole blockchain for previous reviews, only the last blocks have to be verified.

6 The Proof-of-Stake blockchain

The blockchain mechanism is needed in order to maintain a distributed consensus about the state of the reputation system. The blockchain in Bitcoin and other altcoins serves to avoid double-spending attacks (i.e., spending the same coins twice). The equivalent in our system would be a re-use of the token issued by the SP. However, this would have much less impact, as it would be detected easily if it happens in the same chain, and two different ratings in two different chains wouldn't even be a problem (the customer might have changed his/her mind about the product). There is no incentive for a customer to "double-spend" the token anyway. However, there is an incentive for the SP to fork the chain before a negative rating on his behalf is issued. This, however, is not a serious threat, as the customer could simply broadcast the review again.

6.1 Types of messages protected by the blockchain

The blockchain is a succession of blocks, each block containing a certain number of "messages" (similar to the transactions in Bitcoin). Those messages can be of three different types :

1. *Reviews*, who contain the address of the related SP, the valid address of a customer, the signature on the token (i.e. on the hash of the address), review and rating, as well as the signature of this message (using the secret key corresponding to the customer's public key used for the transaction), and the pointer to the last review about the same SP,
2. *Transactions*, which will increase the balance of one wallet and decrease the balance of an other one by the same amount. This transaction must be signed by the private key of the account whose balance will be decreased (this is similar to a BitCoin transaction). They may be useful for SP who lack of coins and would like to purchase them from other accounts who need less coins for their transactions,
3. *Associations* between the RSA-key used by the SP in order to build blind signatures, and the address of the SP. As this kind of operation can be easily created by anyone, a transaction fee could be required in order for it to be included in the blockchain, in order to avoid spam and Denial-of-Service attacks,

Different types of operations could be added later on to the protocol. The implementation details are out of scope of this paper, but using for example JSON-style messages as in BitCoin could be a viable solution to exchange those messages and include them in the blockchain.

6.2 Signing a block

In a Proof-of-Stake algorithm, the signer of the next block is chosen at random given the blockchain history, and the accumulated "stake" of the stakeholder (either the coin age as in PPCoin [29], or the accumulated coins that have not been moved for a certain time as in NXT [6]). The general proof-of-stake algorithm has been heavily improved with PPcoin introducing the so-called *stake-modifiers* which make long-running fork attempts much harder, and BlackCoin (as well as NeuCoin) proposed a *dynamic* stake modifier instead of a static one, further improving the security against long-running attacks. Both NeuCoin and BlackCoin also remove the coin age from the parameters used to determine the next node that will constitute a block. We will choose a solution similar to BlackCoin and NeuCoin, and explain in more details how these improvements help to mitigate threats to the PoS algorithm.

1. Once a block has been constituted, a new *difficulty modifier* is computed
2. Every second, every user computes one hash of the address aggregated with the *stake modifier* (which will be explained later) and the current *timestamp*. The concatenation of the timestamp, stake modifier and address will be called the *kernel*. If this value is smaller than the difficulty modifier multiplied by the *effective balance*, or *stake,* of the account, than the address will have the right to constitute the next block (i.e. if $hash(kernel) \leq stake \times difficulty$). The stake which verifies this equation with the smallest timestamp should be accepted as the new block-creator, timestamps too far in the future should not be accepted (in order to avoid spamming the network). If two stakes verify the equation with the same timestamp, the stake with the higher value should be accepted. If the stakes have the same value, the stake with the lower hash value should be accepted (we suppose that there will be no collision on the hash function). Hashes that are broadcast too late on the network should not be accepted.
3. The address that has been chosen constitutes the new block, signs it, and broadcasts it along with the hash that proves that it has the right to constitute the new block. It is then added to the blockchain.
4. Upon receiving this new block, the other participants verify its correctness and, if it is the case, add it to the blockchain. The difficulty parameter is adjusted, and a new round of Proof-of-Stake can begin.

6.3 Avoiding forks

Proof-of-Stake blockchains have been criticized for facilitating the creation of forks, because of the so-called *nothing-at-stake* problem [23]. It is also difficult to determine, when presented with two different blockchains, which one has been worked on longer and is the "honest" chain. We will describe how these problems could be mitigated.

Avoiding short-living forks After each block, the address that will be allowed to constitute the next block will be determined. As there is, for a given block, only one address that is allowed to constitute this block, there is no risk that a honest user creates a fork by mistake. If, however, two blocks with the same parent node are detected on the network, the only responsible is the address that created and signed those two blocks. It can therefore be punished, for example, by setting its account and reputation to zero. This resembles a simplified version of the Slasher algorithm [5]. Then, a new address can be elected in order to sign the next block.

Avoiding long-running forks While the first category of forks can be (fairly) easily mitigated, there is an other possibility of forking that could disrupt the network : suppose that at block n, the address *add* has the right to constitute a block. It will do so as it was asked, and broadcast it, but at the same time try to build a block with a different hash (using dummy transactions with a small amount of coins between other addresses under its control for example) so that the next block-creator belongs to it. It will continue then to do so, creating more dummy blocks and keep up with the actual blockchain. Then, after a certain time, it will release the constituted blockchain. If the solution intended for short-living forks was to be applied, the whole system would go back into a state many, many blocks ago, which may not be desirable (all the work since then would be lost). So, the network has to choose between those two chains. How should it do so, and convince at the same time a new node that one chain is legit and the other isn't, when we require this system to be trustless ?

In the BitCoin protocol, the chain with the highest cumulated difficulty would be accepted. Here, we can use a similar rule : the chain with the highest cumulated stake should be accepted as the "main" chain. However, when accepting a new block, when multiple blocks are received, it's not the block that was constituted with the highest stake that is to be accepted, but the block with the lowest timestamp. A threshold is therefore needed, after which the system switches to accepting the chain with the highest stake.

6.4 Computing the balance of an address

In order to compute the balance of an address, the whole blockchain has to be traversed, so that every transaction is taken into account and the total balance can be computed. This might be very costly if the blockchain grows large. A simple solution to avoid this problem would be to publish the new balance of an account along with the operations that modify it. The nodes that constitutes the new block would have to verify the correctness of this value. It can simply use the last published value which is supposed to be correct. Therefore, a pointer to the last transaction involving the addresses should be included along the new balance. If two operations modify the balance of the same account in the same block, then the node that constitutes the block must re-organize them. Therefore, this information should *not* be signed by the address broadcasting this operation.

6.5 Bootstrapping the system

Since the Proof-of-Stake algorithm is based on the amount of coins owned by each address, an initial distribution of the coins has to be performed. This could be done, like in PPCoin, by using a first phase of pure Proof-of-Work, and then switching to Proof-of-Stake, or like in Ethereum, where a certain amount of coins could be purchased before the genesis block has been issued. This problem can be solved on a later point.

7 Analysis of the protocol

7.1 Security Analysis

In this section, we will prove the security objectives of our protocol. We will do so in the random oracle model, for practical reasons, as we already have to rely on one-way hash functions in several parts of our protocol.

For our proofs, we define a *security parameter* k, which represents for example the size of the public key parameters, in bits. A function $\epsilon(k)$ is *negligible* if its inverse grows faster than any polynomial function (that is, $\forall n > 0, \epsilon(k)k^n \underset{k \to +\infty}{\longrightarrow} 0$).

Theorem 1 (Token unforgeability). *Given a Service Provider's public key, and a poly-bounded (in a security parameter k) number of signatures from the Service Provider on arbitrary messages, a user is not able to generate one more token (i.e. signature on the hash of an address) except with negligible probability $\epsilon(k)$.*

Proof. This follows directly from the assumption that the blinding signature scheme used is resistant against the "one-more" forgery attack as described by Pointcheval and Stern [38]. In the case of Okamoto, this is even true in the standard model, without supposing that hash functions are "true" random oracles, while Chaum's signature scheme requires this assumption (as well as the rather strong assumption, the hardness of the one-more-RSA-inversion problem[10]).

Remark 1. This implies that badmouthing attacks are not possible on this system. Indeed, since no token can be forged (except with negligible probability), no user can issue a report about a service provider without being involved in a transaction concerning it. Ballot-stuffing attacks, however, cannot be completely mitigated, as a service provider can freely issue tokens. The currency introduced in this protocol helps reducing the risk of ballot stuffing, as the service provider is limited on the number of tokens he can issue, by the number of coins he owns.

Theorem 2 (Reputation unforgeability). *Given the public blockchain history, no service provider is able to advertise a reputation that is not its own (except with negligible probability $\epsilon(k)$).*

Proof. The reputation is bound to an address representing a public key. The service provider can easily convince the customer that he owns this address, for example, when the token is issued, the blind signature could be itself signed using the service provider's address. The service provider could also publish a signature on the public key used for the blind signature scheme, signed using the private key corresponding to the public key of its address. This property is ensured as long as the "classical" signature scheme is secure (i.e. the signatures are unforgeable).

Theorem 3 (Customer anonymity). *Given a rating published in the blockchain concerning a given service provider, the identity from the customer that originated this rating is indistinguishable, from the service provider's point of view, among all the customers that were previously involved in a transaction with that service provider.*

Proof. For each transaction, the service provider knows :

– the identity of the customer,
– the content of the transaction (date, value, goods and currency exchanged, etc.),
– a randomized view of the user's address that is *unlinkable to the actual signature* (this is guaranteed by the blind signature scheme)

The review issued by the customer contains :

– an address generated by the customer for this transaction
– the identity of the service provider
– a signature, by the service provider, on the (hash of) the address
– a rating and a textual review

The customer generates a new address at random for each transaction, the address is therefore independent of the identity of the customer. The service provider already knows its own identity, but is unable to link the actual signature that was issued, to the randomized view it was presented during the token-issuing phase. Finally, we suppose that the service provider does not know, in advance, the rating that will be issued by the customer, not the review associated with it.

Theorem 4 (Customer report unlinkability). *Given two different reports, it is not possible to determine whether they were issued by the same customer or not better than guessing at random.*

Proof. Different reports are signed and issued by randomly generated addresses that are truly independent (if a proper random number generator is used in order to generate them). The signatures on the tokens are unlinkable to the actual transactions as shown earlier, it is therefore not possible to determine whether two reports were generated by the same user or not.

The proposed protocol is able to hide the identity of the rater among all the customers who interacted with a given service provider in a certain time interval, therefore providing some kind of k-anonymity. However, with this system, it is not possible to determine the value of k, because customers do not know when the SP they desire to rate, interacts with other customers. One possibility to overcome this would be to make the service provider issue two unlinkable tokens, and make the customers publish the first one in order to signal that a transaction occurred, so that prospective raters have a way of determining k. This, however, would induce an additional payload on the system, and the SP could simply forge tokens in order to trick the customers into revealing their rating. However, if the rate at which transactions occur is not too low, and if not all the customers end up by rating their transactions, then the anonymity might be guaranteed.

We can devise a simple model that will give an idea of the indistinguishability of the customer reviews. Suppose that customers purchase goods from a given service provider. The arrival of customers can be modeled by a Poisson process with parameter λ. We can also suppose that, after receiving their good, customers will wait for a certain amount of time before submitting a review. In our model, the customer will wait for a time T that is uniformly distributed over $[0;\tau]$ for some $\tau > 0$. In this case, we have the following property :

Theorem 5 (Indistinguishability). *If the arrival of customers is modeled as a Poisson process of parameter λ and if customers wait for a duration that is uniformly distributed over $[0;\tau]$ before submitting their review, then the identity of a customer will be indistinguishable over a set of $\lambda\tau$ customers in average.*

Proof. Let's denote by X_c the arrival time of the customer c, and by Y_c the submission time of the corresponding review. Given a customer c, the service provider does not know which review is the one he submitted among the submission that were sent during $[X_c; X_c + \tau]$. During this period, there will be on average $\lambda\tau$ review submissions (because, if $N(t)$ is a Poisson counting process with parameter λ, then $\forall t, \tau : \mathbb{E}(N(t + \tau) - N(t)) = \lambda\tau$).

In order to verify this, we simulated the arrival of customers and the submission of reviews over a period of 60 days, with parameters $\lambda = 15d^{-1}$ and $\tau = 4d$, with a granularity of a day (meaning that we only recorded the number of arriving customers per day, and the day of the review submission. The review was submitted between the day D of the transaction and $D + 3$, giving thus an interval equal to four days). The results are presented in figure 1, and the average number of reviews which are possible for each customer is close to the theoretical value of 60 (for this experiment, we found 61.8).

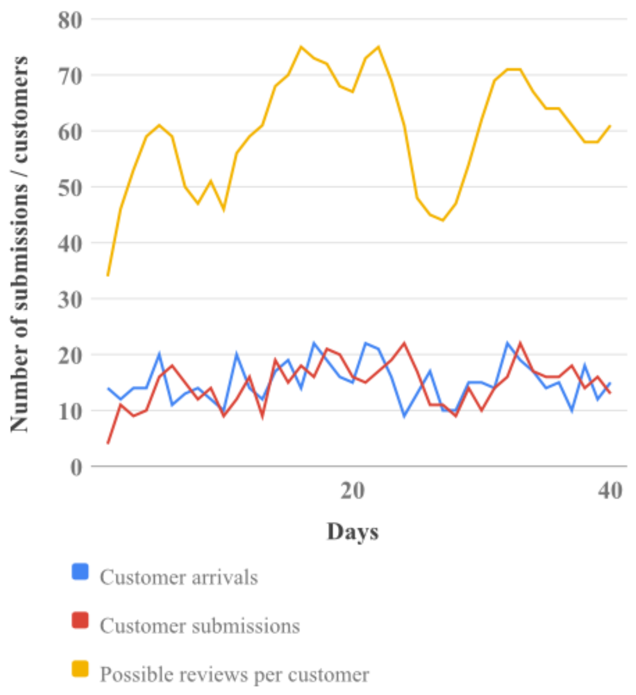
7.2 Malicious behavior

We supposed that the customer receives the blinded signature at the same time that he payed the SP. However, in reality, either the customer first sends money and then gets the blinded token, *or* he first gets the blinded token and then has to pay the SP.

Neither of those two solutions is perfect.

In the first case, the service provider could choose not to issue the token after having received the money. Therefore, the customer will be unable to issue a rating concerning the service provider.

Fig. 1: Simulating the arrival and review-submission of customers ($\lambda = 15d^{-1}, \tau = 4d$)



In the second case, the customer could abort before having sent the money, and use the token in order to issue a review, albeit no transaction between the customer and the SP took place. This could be used in order to perform bad-mouthing attacks against the service provider.

There is, however, a better solution to this problem, if we suppose that an electronic currency is used in order to pay for the good (or the service). In Bitcoin for example, in order to use the money that was send to an address, the sender can specify a script which must be run by the receiver in order to claim the coins. This condition could be "Show a valid RSA signature on a given (blinded!) hash". In order to spend those coins, the SP would have to broadcast this signature, and the customer could unblind it and use the token. There are, however, two drawbacks. First, the SP could wait for a long time before claiming the transfered coins, and in this case, the customer would have to wait before receiving the token. Second, and more importantly, this would require to use a cryptocurrency instead of classical money in order to pay for the good or services. Furthermore, even BitCoin could not be used, because the scripting language used is not Turing-complete, and RSA signatures are not used at all in BitCoin. Therefore, the condition "Show an RSA-signature on a particular hash" could not be expressed in the BitCoin scripting language. Ethereum, however, could be used, as the included scripting language is Turing-complete.

As it should be possible to use this system with "classical" money, a choice has to be made between the first two solutions. If the customer gets the token first, the system could be easily exploited by any malicious person - for example a competitor. As the customer remains anonymous, the SP could not do much in order to prevent the bad-mouthing. If the customer gets the token after having payed, the SP could withhold the token and prevent the customer from submitting the review. In this case, however, the customer will be able to identify the malicious party, and could force them to behave by legal means for example. Therefore, it seems a better solution to *first* pay the SP, and *then* retrieve the token.

7.3 Robustness against generic attacks

In this section, we will explain how our proposed system copes with generic attacks against reputation systems : bad-mouthing, ballot stuffing, Sybil attacks, and whitewashing.

Bad-mouthing Bad-mouthing consists in lying about the performance of a service provider in order to decrease his reputation. This could be done, for example, by a competitor. Our system prevents bad-mouthing thanks to the usage of **tokens** : a customer can only submit a review about the performance of a service provider if he was really engaged in a transaction with this service provider. However, this does not *completely* mitigate this threat : the customer could still buy a good from the service provider and then lie about the transaction. In this case, outlier detection could be applied, and ratings filtered, in a similar fashion as in the paper by Anceaume et al. [7], although the customer unlinkability decreases the performance of this method (which lead to a second paper, by Lajoie-Mazenc, Anceaume et al. [30] that allowed customer linkability).

Ballot-stuffing Ballot stuffing is the opposite of bad-mouthing. This attack consists in increasing one's own reputation. As the service providers generate the tokens that allow feedback-submission on their own, this attack could only partially be mitigated with the use of **coins**. This limits the number of tokens that can be generated by the service providers. Therefore, if the service provider does a ballot-stuffing attack, he would have less tokens left for tokens related to "real" transactions, and this would cost him money (as the transactions could not take place). Of course, the service provider could still buy tokens with money (if an exchange platform was to be set up), and use them in order to perform a ballot-stuffing attack. This would, again, cost the service provider money.

Whitewashing Whitewashing consists in exiting a system after having accumulated bad reputation, in order to re-enter it again and removing the accumulated bad reputation. As the initial reputation of a new service provider is 0, the service provider would not gain much from leaving and re-entering the system with a new identity. However, bad reviews are worse than no reviews, so there could be an incentive in order to do so. One way to limit this would be to bind the identity of a service provider to, for example, his website, through a specific operation on the blockchain. The service provider could still change the domain name, but again, this would cost money.

Sybil attacks Sybil attacks combine more or less the attacks described above. They consist in creating multiple identities in the system in order to disrupt it. They pose no more threat than the other types of attacks, as there is no concept of "identity" for the customers in our system, and creating multiple identities for a service provider can only be used to either perform whitewashing (if he creates one identity after another) or ballot-stuffing (if he creates multiple fake transactions).

These results are summarized in **Table 2**.

Table 2: Resistance against generic attacks

Attack	Possible solutions	Possible to implement
Badmouthing	Proof of Transaction	Yes
	Outlier Detection	Partial
Ballot-stuffing	Limiting reviews	Yes
	Verify transaction	No
Whitewashing	Newcomer disadvantage	Yes
	Detect returning SP	Partial

8 Conclusion

Reputation systems need to be privacy-preserving in order to work properly, without the raters having to be afraid of retaliation. Building a reputation system that is privacy-preserving without any trust assumptions is not a trivial task. However, such a system would be highly valuable, because there is much less risk that the privacy of the users could be breached. We described such a reputation system for e-commerce applications, and analyzed the security guarantees. Some points would still need further discussion, such as the exact way of generating coins that would ensure that service providers have enough of them in order to be able to supply enough tokens for their customers, but at the same time still limit ballot-stuffing attacks. The performance of the protocol was not tested either. While it seems that the performance should by good, due to the simplicity of the design, this could be verified by implementing it.

References

1. Comodo report of incident - comodo detected and thwarted an intrusion on 26-mar-2011. `https://www.comodo.com/Comodo-Fraud-Incident-2011-03-23.html`.
2. Key internet operator verisign hit by hackers. `http://www.reuters.com/article/2012/02/02/us-hacking-verisign-idUSTRE8110Z820120202`.
3. Maintaining digital certificate security. `http://googleonlinesecurity.blogspot.fr/2015/03/maintaining-digital-certificate-security.html`.
4. Reputation systems. `https://github.com/ethereum/wiki/wiki/Problems#12-reputation-systems`.
5. Slasher: A punitive proof-of-stake algorithm. `https://blog.ethereum.org/2014/01/15/slasher-a-punitive-proof-of-stake-algorithm/`.
6. Whitepaper:nxt. `http://wiki.nxtcrypto.org/wiki/Whitepaper:Nxt`.
7. Emmanuelle Anceaume, Gilles Guette, Paul Lajoie Mazenc, Nicolas Prigent, and Valérie Viet Triem Tong. A Privacy Preserving Distributed Reputation Mechanism. October 2012.
8. Elli Androulaki, Seung Geol Choi, Steven M. Bellovin, and Tal Malkin. Reputation systems for anonymous networks. In *Proceedings of the 8th International Symposium on Privacy Enhancing Technologies*, PETS '08, pages 202–218, Berlin, Heidelberg, 2008. Springer-Verlag.
9. Michael Barbaro and Tom Zeller Jr. A face is exposed for aol searcher no. 4417749, August 2006.
10. Mihir Bellare, Chanathip Namprempre, David Pointcheval, Michael Semanko, et al. The one-more-rsa-inversion problems and the security of chaum's blind signature scheme. *Journal of Cryptology*, 16(3):185–215, 2003.
11. E. Ben Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, and M. Virza. Zerocash: Decentralized anonymous payments from bitcoin. In *Security and Privacy (SP), 2014 IEEE Symposium on*, pages 459–474, May 2014.
12. John Bethencourt, Elaine Shi, and Dawn Song. Signatures of reputation. In *Proceedings of the 14th International Conference on Financial Cryptography and Data Security*, FC'10, pages 400–407, Berlin, Heidelberg, 2010. Springer-Verlag.
13. Davide Carboni. Feedback based reputation on top of the bitcoin blockchain. *arXiv preprint arXiv:1502.01504*, 2015.
14. David Chaum, Amos Fiat, and Moni Naor. Untraceable electronic cash. In *Proceedings on Advances in cryptology*, pages 319–327. Springer-Verlag New York, Inc., 1990.
15. Kourosh Davarpanah. Neucoin: the first secure, cost-efficient and decentralized cryptocurrency. 2015.
16. Tassos Dimitriou and Antonis Michalas. Multi-party trust computation in decentralized environments in the presence of malicious adversaries. *Ad Hoc Netw.*, 15:53–66, April 2014.
17. Shlomi Dolev, Niv Gilboa, and Marina Kopeetsky. Efficient private multi-party computations of trust in the presence of curious and malicious users. *Journal of Trust Management*, 1(1):8, 2014.
18. John R. Douceur. The sybil attack. In *Proceedings of 1st International Workshop on Peer-to-Peer Systems (IPTPS)*, 2002.
19. Evan Duffield and Daniel Diaz. Dash : A privacy-centric crypto-currency, 2014. `https://www.dashpay.io/wp-content/uploads/2015/04/Dash-WhitepaperV1.pdf`.

20. Ehud Gudes, Nurit Gal-Oz, and Alon Grubshtein. Methods for computing trust and reputation while preserving privacy. In Ehud Gudes and Jaideep Vaidya, editors, *Data and Applications Security XXIII*, volume 5645 of *Lecture Notes in Computer Science*, pages 291–298. Springer Berlin Heidelberg, 2009.
21. Omar Hasan, Lionel Brunie, and Elisa Bertino. Preserving privacy of feedback providers in decentralized reputation systems. *Comput. Secur.*, 31(7):816–826, October 2012.
22. Omar Hasan, Lionel Brunie, Elisa Bertino, and Ning Shang. A decentralized privacy preserving reputation protocol for the malicious adversarial model. *IEEE Transactions on Information Forensics and Security*, 8(6):949–962, 2013.
23. Nicolas Houy. It will cost you nothing to'kill'a proof-of-stake crypto-currency. *Available at SSRN 2393940*, 2014.
24. Don Johnson, Alfred Menezes, and Scott Vanstone. The elliptic curve digital signature algorithm (ecdsa). *International Journal of Information Security*, 1(1):36–63, 2001.
25. Audun Jsang and Roslan Ismail. The beta reputation system. In *Proceedings of the 15th bled electronic commerce conference*, volume 5, pages 2502–2511, 2002.
26. M Frans Kaashoek and David R Karger. Koorde: A simple degree-optimal distributed hash table. In *Peer-to-peer systems II*, pages 98–107. Springer, 2003.
27. Sepandar D. Kamvar, Mario T. Schlosser, and Hector Garcia-Molina. The eigentrust algorithm for reputation management in p2p networks. In *Proceedings of the 12th International Conference on World Wide Web*, WWW '03, pages 640–651, New York, NY, USA, 2003. ACM.
28. Florian Kerschbaum. A verifiable, centralized, coercion-free reputation system. In *Proceedings of the 8th ACM Workshop on Privacy in the Electronic Society*, WPES '09, pages 61–70, New York, NY, USA, 2009. ACM.
29. Sunny King and Scott Nadal. Ppcoin: Peer-to-peer crypto-currency with proof-of-stake, 2012.
30. Paul Lajoie-Mazenc, Emmanuelle Anceaume, Gilles Guette, Thomas Sirvent, and Valérie Viet Triem Tong. Efficient Distributed Privacy-Preserving Reputation Mechanism Handling Non-Monotonic Ratings. January 2015.
31. Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008. `https://bitcoin.org/bitcoin.pdf`.
32. A. Narayanan and V. Shmatikov. Robust de-anonymization of large sparse datasets. In *Security and Privacy, 2008. SP 2008. IEEE Symposium on*, pages 111–125, May 2008.
33. Tatsuaki Okamoto. Efficient blind and partially blind signatures without random oracles. In *Theory of cryptography*, pages 80–99. Springer, 2006.
34. Elan Pavlov, JeffreyS. Rosenschein, and Zvi Topol. Supporting privacy in decentralized additive reputation systems. In Christian Jensen, Stefan Poslad, and Theo Dimitrakos, editors, *Trust Management*, volume 2995 of *Lecture Notes in Computer Science*, pages 108–119. Springer Berlin Heidelberg, 2004.
35. Morgen Peck. The bitcoin arms race is on! *Spectrum, IEEE*, 50(6):11–13, 2013.
36. Ronald Petrlic, Sascha Lutters, and Christoph Sorge. Privacy-preserving reputation management. In *Proceedings of the 29th Annual ACM Symposium on Applied Computing*, SAC '14, pages 1712–1718, New York, NY, USA, 2014. ACM.
37. Andrew Poelstra et al. Distributed consensus from proof of stake is impossible, 2014.
38. David Pointcheval and Jacques Stern. Provably secure blind signature schemes. In *Advances in Cryptology – ASIACRYPT'96*, pages 252–265. Springer, 1996.

39. Thomas Pornin. Deterministic usage of the digital signature algorithm (dsa) and elliptic curve digital signature algorithm (ecdsa). `https://tools.ietf.org/html/rfc6979`.
40. Paul Resnick and Richard Zeckhauser. *Trust among strangers in internet transactions: Empirical analysis of eBay' s reputation system*, chapter 6, pages 127–157.
41. Ion Stoica, Robert Morris, David Karger, M Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. *ACM SIGCOMM Computer Communication Review*, 31(4):149–160, 2001.
42. Pavel Vasin. Blackcoin's proof-of-stake protocol v2. `http://blackcoin.co/blackcoin-pos-protocol-v2-whitepaper.pdf`.