A Decentralized Anonymity-Preserving Reputation System with Constant-time Score Retrieval

Rémi Bazin*, Alexander Schaub*, Omar Hasan† and Lionel Brunie†
*Department of Computer Science, École Polytechnique
91120 Palaiseau, France
†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.**

I. 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 eBay, Amazon or Shopzilla. 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 [2]) have emerged for mainly two reasons: releasing the central server from resource consuming tasks to distribute these among the peers, and 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 [3]. 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 [4], 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 privacy of the customers shall therefore be entirely preserved.

The protocol that we propose is also resistant against Sybil attacks [5]. 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. The prevention of self-promotion attacks however requires an external way of differentiating bots from actual clients. This may for instance be done with fees or external registration. Our scheme will incorporate *tokens* as a way to prevent self-promotion, and several methods for generating these tokens are formulated later in the paper.

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 [6], blind signatures [7] and non-interactive zero-knowledge proofs, and will be efficient (constant-time) when retrieving reputation. We also distribute the computational and memory load over the SPs in a fair manner.

The rest of the paper is organized as follows. Section II provides an overview of the state of the art concerning privacy-preserving reputation systems. Section III illustrates the model for the environment in which our protocol is to be used while Section IV highlights the objectives of our work. Our construct of *tokens* is described in Section V. Section VI describes the core protocol. An analysis of the protocol with regards to the previously defined objectives is presented in Section VII. Finally, we conclude in Section VIII.

II. 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. As a matter of fact, some solutions proposed by Pavlov et al. [8] and Dolev et al. [9] suffer from this weakness, despite their significant contribution in the domain: they do not resist an attack by a few colluding malicious peers, which could reveal private feedback information. Some other works are nonetheless more secure and resistant to small groups of malicious peers: the StR^M algorithm by Dimitriou et al. [10] and the Malicious k-shares protocol by Hasan et al. [11] 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. This way of partially hiding information leads to multiple issues linked to the mutability of the set of participating peers. Even though these issues might be tackled (e.g., by using *source managers* in [11]), the aim of this paper is to find a fully privacy-preserving scheme that satisfies all of our objectives.

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. [12] and Petrlic et al. [13], 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. [14] and Lajoie-Mazenc et al. [15] 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. [16] 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. [17] illustrates a promising scheme based on signatures on published data. While this protocol is very interesting and secure, we can only regret the monotonic aspect of the feedback that 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 in our paper.

III. Model

The model we choose for our protocol is consistent with that of an e-commerce environment: as it was previously mentioned, 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. This feedback may however be replaced by a more recent one before it expires.

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.

IV. 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 ([8], [9], [10], [11], [12], [13], [14], [15], [17]).

A. 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 we aim to propose.

Furthermore, any processing and memory overhead generated by each new rating should be entrusted to the SPs, since it is in their own interest to maintain their reputation. This distribution of the computational costs to the SPs proportionally to their popularity seems fair.

B. 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. This kind of unlinkability is illustrated in Figure 1.
2) **Rating – rating unlinkability** It has been shown ([3], [18]) that this second kind of unlinkability – between several ratings of a unique user – is also primordial to

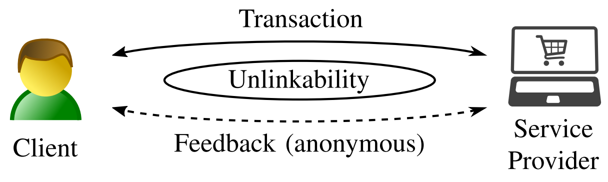

Fig. 1. Transaction – rating unlinkability

preserve the anonymity of the users. Ratings by a unique client and for the same SP will nonetheless still be linkable. It is indeed essential to prevent ballot-stuffing.

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 our e-commerce context, this behavior is indeed desirable: these SPs would generally wish to be known anyhow.

C. *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. If we accepted centralized schemes and TTPs, a reputation system such as the one used by Amazon would clearly be sufficient for most e-commerce websites.

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. A distributed authority may be used for increased security, and its members may use multi-signatures [19] to create a compact and secure certificate, that can be verified as quickly as a single signature.

D. *Robustness*

Different levels of trust are usually considered in reputation systems: each peer might be entirely honest, semi-honest or malicious. In the semi-honest model, also known as honest-but-curious, the peers do not deviate from the protocol, but they try to gain as much information as possible with the data they possess. Another aspect to consider, which makes curious peers more threatening, is whether or not the peers collude together. In our scheme, we place ourselves in a situation where some peers might be colluding malicious, but most of them – among what we will call the *trackers* – are honest or at most non-colluding curious.

Thus we want our system to be able to resist attacks such as ballot-stuffing, Sybil attacks and whitewashing. In order to prevent self-promotion from the SPs, we will consider the possibility to add a Certification Authority to the scheme.

V. Tokens – security against Sybil attacks

In this section, we describe the *tokens*, 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. There are several ways to achieve this, which will be mentioned thereafter. Other uses of these tokens may include:

- Replacing CAPTCHAs used in registration forms.
- Preventing comment attacks in web pages where anybody can post comments, without the risk of rejecting the comment of a rightful user and without requiring any further identification of the users.
- Setting up one-time discounts and offers for individuals.

In our protocol, as well as in all these cases, the tokens are to uniquely identify a couple user/service. In our case, it will be a couple client/SP. However, we don't want other people to be able to reuse the token once the corresponding feedback record has expired. To achieve this, we will either use one-time tokens, that act like a fee for each single review, or add the possibility to include a commitment to the one-time public key K that is used in the feedback records (see Section VI-B2).

The following subsections describe different ways of implementing such tokens. They could moreover be combined together. On the one hand, one could require clients to use simultaneously two types of tokens grouped into one big token. On the other hand, we might use one of the following token implementations as a certificate initiator. For example, we could ask clients to pay a fee once and for all, that will allow them to request a certificate from a group of peers (what we will call the *trackers* for instance). This certificate could then be used with NIZK proofs as in the last implementation detailed below.

A. *Using CAPTCHAs*

The goal of our tokens is to tell computers and humans apart. This is somewhat reminiscent of the well-known techniques grouped under the name CAPTCHA. These CAPTCHAs may indeed be used for the generation of the tokens in our scheme, even though it is not highly recommended because of the unfriendly time consumption it requires from the users. In addition to this, they become more and more vulnerable nowadays [20] and are thus also discouraged from the security point of view.

The decentralized way to use them would be to carefully select some witness peers in the network – using a Secure MultiParty Computation algorithm such as a collective coin flipping for instance. These peers would generate a CAPTCHA problem, send it to the requiring user and sign the token demand if and only if the answer is correct. The token itself would thus be a collective signature (be it a group of several signatures or a compact multi-signature [19]) over some service-identifying data.

B. *Using proofs of work*

Another way to do this is to use proofs of work (as in the Bitcoin protocol [2]), but this could be troublesome because it might be profitable for vendors to massively compute these proofs of work, especially with dedicated hardware.

C. Using fees

A related proposition would be to use small fees for posting feedback values. This could be achieved by using an anonymous electronic cash system such as Dashcoin [21] or Zerocash [22]. However, the SP could still find it profitable to cheat, given that these fees would have to be quite low for the system to be adopted.

D. Using certificates and NIZKs

The three previous methods have a common asset: they are completely decentralized. However, they also have a common drawback: the SP can get as many tokens as he wants, provided that he has enough money to afford them. Apart from fees, we can indeed easily imagine that proofs of work and even CAPTCHA problems may be massively solved if given the right amount of money. If the price for creating false feedback is too low, the SPs will probably benefit from generating false feedback records. On the contrary, if this price is too high, it will be dissuasive for clients to use this reputation system. Because of the lack of a good compromise, we will prefer another solution to get rid of this issue: we will use a system with certificates to differentiate people.

1) Presentation: 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. For decentralization purposes, we may nonetheless use multiple CAs and require each one of them to give its personal signature for the resulting certificate to be valid. This can be done without increasing the size nor the nature of the signature in the certificate by using the multi-signature scheme by Blodyreva et al. [19].

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 VI-B2) so as to prevent any subsequent use of the token.

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 [23], we want to construct the following proof :

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

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.

3) Implementation: In our proposed implementation, the signatures would be implemented using the Boneh-Lynn-Shacham (BLS) Signature Scheme [24]. As previously mentioned though, we might use a group of CAs that would act collectively to sign a client's public key, using a compact multi-signature [19].

We will use the NIZK model proposed by Groth et al. [25] to create the tokens. To this end, we will also use a bilinear map between groups of prime order p, which is denoted by $e : \mathbb{G} \times \hat{\mathbb{G}} \rightarrow \mathbb{G}_T$. g and $\hat{g}$ shall be generators for the two groups $\mathbb{G}$ and $\hat{\mathbb{G}}$.

In our implementation, we propose the following setup and verification procedure for a user's credentials:

$$\text{CRED.VERIFY}(pk_C, sk_C) := \left[pk_C = g^{sk_C}\right]$$

where $sk_C \in \mathbb{Z}_p$ and $pk_C \in \mathbb{G}$.

For the CA, we use the same kind of signatures but in the group $\hat{\mathbb{G}}$. We may improve it so that several entities can cooperate to create a collective public key $\hat{pk}_A = \hat{g}^{sk_{A,1} \cdot (\cdots) \cdot sk_{A,n}}$ gathering the private keys of n participating CAs. The verification procedure would therefore still have the exact same complexity. The signature algorithms are thus, with a message $M \in \mathbb{G}$, the credentials $(\hat{pk}_S, sk_S) \in (\hat{\mathbb{G}}, \mathbb{Z}_p)$ of the signer in $\hat{\mathbb{G}}$ and a signature $\sigma \in \mathbb{G}$:

$$\text{SIG.CREATE}(M, sk_S) := M^{sk_S}$$

$$\text{SIG.VERIFY}(\sigma, M, \hat{pk}_S) := \left[e\left(\sigma, \hat{g}\right) = e\left(M, \hat{pk}_S\right)\right]$$

We propose to use this signature scheme for the certificates $cert \in \mathbb{G}$ delivered by the CAs, which will be signatures on the public keys of the clients:

$$\text{CERT.CREATE}(pk_C, sk_A) := \text{SIG.CREATE}(pk_C, sk_A)$$

$$\text{CERT.VERIFY}(cert, pk_C) := \text{SIG.VERIFY}(cert, pk_C, \hat{pk}_A)$$

To check the signature of the key K, we will assume a full-domain hash function $\hat{H} : \{0,1\}^* \rightarrow \hat{\mathbb{G}}$ which we will treat as random. We will therefore place the following check inside the NIZK, where the groups $\mathbb{G}$ and $\hat{\mathbb{G}}$ need to be exchanged for this variation of SIG.VERIFY:

$$\widehat{\text{SIG.VERIFY}}(\hat{sign}, \hat{H}(K), pk_C)$$

Finally, we will use the following identifier creation:

$$\text{TOKENIZE}(\hat{v_{SP}}, sk_C) := \widehat{\text{SIG.CREATE}}(\hat{v_{SP}}, sk_C)$$

with $\hat{v_{SP}} \in \hat{\mathbb{G}}$ a unique identifier for the SP – that may be the result of a map from its name – and $\hat{value} \in \hat{\mathbb{G}}$ the created unique identifier value.

To sum everything up, if we replace the expressions in Equation (1) with their equivalents, we are left with a NIZK that looks like:

$$\text{NIZK}\left\{ pk_C, sk_C, cert : \begin{array}{l} pk_C = g^{sk_C} \\ \bigwedge e\left(cert, \hat{g}\right) = e\left(pk_C, \hat{pk_A}\right) \\ \bigwedge e\left(g, \hat{sign}\right) = e\left(pk_C, \hat{h}\right) \\ \bigwedge \hat{value} = \hat{v_{SP}}^{sk_C} \end{array} \right\}$$

where $\hat{pk_A}$ and $\hat{h} := \hat{H}(K)$ are supposed to be known and $\hat{sign}$ and $\hat{value}$ explicitly contained inside the token, alongside with the NIZK. We should point out that we can further simplify this NIZK: since sk_C is part of the variables, we can replace $\widehat{\text{SIG.VERIFY}}(\hat{sign}, \hat{H}(K), pk_C)$ with $\hat{sign} = \widehat{\text{SIG.CREATE}}(\hat{H}(K), sk_C)$, thus leading to:

$$\text{NIZK}\left\{ pk_C, sk_C, cert : \begin{array}{l} pk_C = g^{sk_C} \\ \bigwedge e\left(cert, \hat{g}\right) = e\left(pk_C, \hat{pk_A}\right) \\ \bigwedge \hat{sign} = \hat{h}^{sk_C} \\ \bigwedge \hat{value} = \hat{v_{SP}}^{sk_C} \end{array} \right\} \tag{2}$$

This NIZK being compatible with the ones supported in the paper by Groth et al. [25], this concludes the implementation part. In the following parts of this paper, we will be referring to the creation of tokens with:

$$\text{CREATETOKEN}(SP, K) := \left(\hat{sign}, \hat{value}, \text{NIZK}\{\cdots\}\right) \tag{3}$$

where "NIZK$\{\cdots\}$" has been made explicit in Equation (2) and SP is the domain name associated with the SP, thus identifying him; $\hat{v_{SP}} := \hat{H}(SP)$. The verification of the zero-knowledge proof inside a token $\mathbf{t}$ will be written as $\text{VERIFYTOKEN}(\mathbf{t}, v_{SP}, K)$.

VI. Description of the protocol

A. Outline

Our protocol involves three kinds of nodes, as listed below and as shown in Figure 2.

- 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.
- 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.

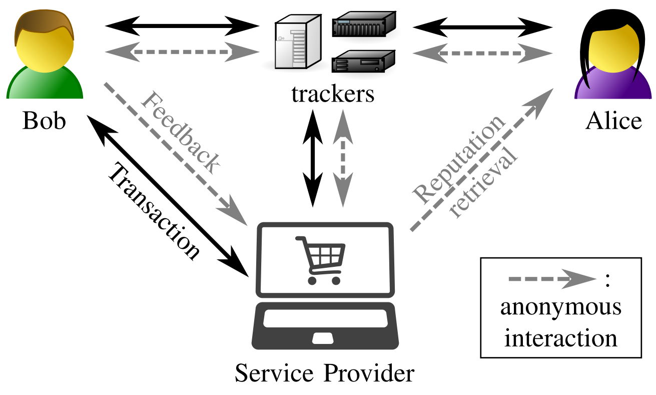

Fig. 2. Overview of the different entities and some primitive operations

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

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

- **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 VI-B3).
- **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.

B. Setup

1) 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 without any protection against Sybil attacks, since the corruption of a majority of these trackers threatens the security of the scheme. The inclusion of a new tracker would need to be carefully checked as well as validated by the previously existing trackers.

Although we are including these supposedly trusted parties, the security is no longer dependent on a single third party but on an aggregate trustworthiness of a large number of trusted parties. Even if a few become corrupted, the security is still upheld as long as a majority of them is honest. One possible way of selecting the trackers would be to ask many different well-established and well-trusted organizations to become trackers. Competition between some of them and the fear of fraud discovery would potentially prevent any colluding malicious majority.

In our model with tokens, these tokens could in fact be used to allow anyone to become a tracker. The assumption is that these tokens are a secure defense (e.g. the previously proposed implementation with a secure Certificate Authority) against Sybil attacks. We would however still need to restrict the number of trackers to a reasonable quantity, not to impinge on efficiency.

A possible outcome of the proofs of bad behavior would be the exclusion of a tracker or SP, once a maximum limit of malicious activities has been reached. We propose to set this limit to 1 for the trackers because of the importance of their honesty.

2) 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 V)
- 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

3) 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 $(d, v_{current}, v_{total}, h, s_3, data)$ where:

- d: Date of publication
- v_T: Aggregated reputation value over the latest period T
- v_{tot}: Aggregated reputation value over the period n_tT
- 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)
- $data$: List of 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). These trees are detailed in Section VI-D2. 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.

4) Updating the list of trackers: A peer will have a possibly outdated list of trackers in memory when it tries to connect for the first time. To update this list in case it is too old, or gets too old, we will ask all the listed trackers for their updated list, as well as the list of proofs they have for each of the old trackers. The application will then decide which trackers are to be trusted and which to eliminate, based on the minority / majority aspect of the answers (see Section VI-D1). All the trackers who have a valid proof of withdrawal or an undeniable proof of malicious behavior will however not be taken into account in this decision process, whether they responded or not. A detailed implementation of this procedure is given in Algorithm 6, Appendix B. This update process will allow us to assume in the following sections that the clients have a list of trackers that can be considered up-to-date.

C. Primitive operations

1) 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 [26]. If he is asked directly, the query is no longer anonymous, but it is faster.

Should a SP refuse to send his signed reputation, any peer may then use the trackers: they can either directly answer the request, or collectively sign a proof of bad behavior if they realize that the SP also refuses to tell them (they might also use an anonymous connection to hide their identity while requesting this information).

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. It is used for instance in the DATRET Algorithm (Algorithm 8 in Appendix B) which collects the data – feedback records – of the block.

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 $d$
    end if
    if $\neg$ConnectTo($SP$) then
        fail with Unable to connect to $SP$
    end if
    $(header, s_4) \leftarrow$ 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
    end if
    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
    end if
    return $(v_{tot}, header, s_4)$
end procedure
```

2) 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 VI-E).
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 5: BLINDSIG to do the financial transaction in itself (see Section VI-E). It also uses the token creation scheme CREATETOKEN (Equation 3) defined in Section V.

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 $(sk, K, s_{SP})$
end procedure
```

3) 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 (see Section VII-B2).
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 V). He then signs the whole 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. This can for instance be achieved using a MIX network such as Tor [26].

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.

4) 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

Algorithm 3 Send a feedback record

```
procedure SendFB($SP, (sk, K, s_{SP}), v, c$)
    Get and verify the root of $SP.\mathcal{T}_3$
    while client not satisfied with the anonymity set do
        Wait for some time and retrieve it again
    end while
    if Feedback about $SP$ already sent today then
        fail with Only one per day, please return tomorrow
    end if
    if Feedback aggregation period then
        Wait for the end of the day
    end if
    $d \leftarrow$ 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($SP$)
    $\alpha$.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
        end if
    end if
    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
    end if
end procedure
```

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

Algorithm 4 Feedback aggregation (tracker side)

```
procedure TFbAggregate($SP, tosign, hinfo, s_6$)
    if $SP \in$ keys($b_t^2$) then
        fail with Malicious behavior (send $b_t^2[SP]$)
                        ▷ Example of "punishment": exclusion
    end if
    $(d, v_T, v_{tot}, h) \leftarrow tosign$
    if $d \neq$ today() $+ 1$ then
        fail with Wrong day
    end if
    if $SP$ already aggregated today then
        fail with Only one per day (send last data)
    end if
    $(h', r_1, r_2, r_3) \leftarrow hinfo$
    if $h \neq$ Hash($SP, h', r_1, r_2, r_3$) then
        fail with Wrong hash
    end if
    if $\neg$CheckSig($SP, s_5, tosign$) then
        fail with Wrong signature
    end if
    Mark $SP$ as aggregated for today
    return CreateSig($tosign$)
end procedure
```

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, as long as it is consistent with the previous prerequisite.

Most systems (at least all the ones with a finite set of feedback values) are compatible with such a requirement. To compute the average of the feedback values for instance, we can save the sum σ of these values as well as their number m in both v_{tot} and v_T. The verification of the new v_{tot} value can easily be done thanks to the following formula, where E evaluates to a quantity that may be σ or m, and S_i is the set of feedback records for day i (mutually exclusive):

$$E\left(\bigcup_{i=d-n_t+1}^{d} S_i\right) = E\left(\bigcup_{i=d-n_t}^{d-1} S_i\right) - E\left(S_{d-n_t}\right) + E\left(S_d\right) \quad (4)$$

Which translates into:

$$v_{tot,d} = v_{tot,d-1} - v_{T,d-n_t} + v_{T,d}$$

This equation is the reason why each SP is responsible for saving $n_t + 1$ blocks of records, and not just n_t: this is for inspectors to be able to retrieve and check $v_{T,d-n_t}$ when verifying $v_{tot,d}$.

Along the same lines, if we use a finite set of possible feedback values (a system with stars for instance, where there are at most 5 or 10 possible values), we can also rely on Equation 4 to aggregate the feedback, by only counting the number of reviews a certain feedback value corresponds to. When displaying the reputation, these totals may then be further aggregated to display a unique mark. The Beta reputation system [27] proposed by Jøsang and Ismail is an example of such an aggregation that could be used when displaying v_{tot}.

An interesting fact is that the feedback aggregation can also take into account the number of clients who went through a transaction but did not rate the SP, during the $n_t T$ period. This quantity can be obtained thanks to the difference between the number of buyers for this period and the number of records for this period. Both can be integrated into v_T and v_{tot} and then updated, thanks to the list of records and thanks to the Merkle tree $\mathcal{T}_3$ (see Section VI-B2).

D. Handling malicious peers

1) Handling malicious trackers: When a peer obtains diverging answers from the trackers, the law of the majority is used: the response given by the majority is accepted considering the assumption that the majority is honest. The diverging minority is considered dishonest and the signed messages received from the minority trackers are considered to be proofs of bad behavior of these trackers. The peer will send them to the honest ones in order for these proofs to be appended to their hash table b_t^1, and for the corresponding trackers to be excluded from the system.

There are three kinds of proofs of bad behavior that can be generated for the trackers:

- The ones inherent to the procedure UPDATETRACKERS whose aim is for clients to update their list of trackers.
- When a tracker refuses without any appropriate reason to sign the block header of a SP, this SP can ask the other trackers to communicate themselves with the corrupted tracker. This might lead to a proof of bad behavior if these trackers also face an unexplained refusal.
- A tracker signs several block headers from a single SP for the same day.

2) Handling malicious SPs: Some of the possible bad behaviors of the SP can be detected directly while processing the previous primitive operations. However, other verification procedures – such as checking the correctness of the feedback records included inside a block – are not necessarily included in these operations. Still, thanks to the signatures of the trackers on the blocks, uniqueness of the signed blocks is ensured. That means that anybody can do all the remaining checks over this block of data. We leave it to the implementation to decide who does these security checks. It might be the trackers before signing the blocks, the SP checking the blocks of their competitors or even the client themselves, by activating an option on the application to do some background checks, for instance. Note that the blocks do not need to be checked in entirety. The system of proofs of bad behavior is to be dissuasive enough so that even a slight probability of being caught is sufficient to discourage SPs from deviating from the protocol. This way of controlling the SPs enables anyone to do the verification work and maybe construct proofs of bad behavior that will be sent to the trackers. We will call *inspectors* the peers who run verification procedures, whoever they are and whatever verification they do.

The proofs of bad behavior may be generated in two ways. The first case happens when a SP refuses to deliver some information or gives information with a bad signature. Then the inspector can ask the trackers for this information. If a majority of trackers fails to retrieve the information, they generate and sign a proof of bad behavior that says so. On the other hand, if the trackers get the information, then they just forward it to the inspector. This first kind of proof of bad behavior ensures the availability of the SPs, meaning that they are forced to provide an answer to every information demand, with a valid signature if required.

The second kind of proofs of bad behavior is more straightforward: when an inspector receives some signed information that is erroneous, this constitutes in itself a proof of bad behavior, that can be sent to the trackers who are to directly accept it – provided that it is valid.

Our scheme is designed so that this system of proofs of bad behavior deters the SPs from doing any malicious activity.

For example, to prevent any ballot-stuffing attack that uses the same blind signature several times, we make sure that the one-time public keys K that are in the feedback records are not used twice. To do this check efficiently, each SP maintains a Merkle tree $\mathcal{T}_1$ that contains all the one-time public keys that have been used thus far.

A second Merkle tree $\mathcal{T}_2$ containing the currently-used token values along with their publication date d is also updated in the same manner. In a similar way, this is to efficiently prevent two feedback records having the same token value $\hat{value}$ from being both aggregated. In order to be able to quickly check a specific identity, the nodes of this tree are to be sorted based on the token value $\hat{value}$ itself.

In addition to the trackers' proofs that ensure the availability of the SPs, inspectors may generate the following types of proof of bad behavior for a specific SP:

- REPRET: Wrong trackers' signature on the block header.
- SENDFB: The feedback record has not been included in the data block despite the commitment C.
- DATRET: Wrong hash h' for the block data $data$.
- A record has been added with an already used identity ($\mathcal{T}_2$), but the new aggregated feedback does not take into account the removal of the previous feedback value (i.e. wrong aggregation v_T).
- Added record with an already used public key K ($\mathcal{T}_1$).
- Wrong aggregation v_{tot}.
- Invalid feedback record in a data block (wrong date, value, token or signature).
- Same public key K or same token value $\hat{value}$ twice in the same block.
- Malformed Merkle tree: $\mathcal{T}_1$, $\mathcal{T}_2$ or $\mathcal{T}_3$ is malformed or incoherent with the corresponding expected root r_i.
- A record has been added, but not its public key in $\mathcal{T}_1$.
- A public key has been added in $\mathcal{T}_1$ without any corresponding feedback record.

- A public key has disappeared from $\mathcal{T}_1$.
- A record has been added, but not its token value in $\mathcal{T}_2$.
- An identity has been added in $\mathcal{T}_2$ without any corresponding record.
- An identity has not been removed from $\mathcal{T}_2$ whereas the air date for this identity has expired.
- An identity has been removed from $\mathcal{T}_2$ whereas the air date for this identity has not expired.
- A record has been added with an already used identity, but the publication date for the corresponding identity has not been updated in $\mathcal{T}_2$.
- The publication date of an identity in $\mathcal{T}_2$ has been altered without this identity being in a new feedback record.
- A token disappeared in $\mathcal{T}_3$ (verification is not really necessary since SP have an incentive to keep them).
- $\mathcal{T}_3$ contains an invalid token.

E. Blind signatures

Many algorithms exist for blind signatures: from the most well-known and simple one based on RSA cryptography [28] to other more complex ones [29], [30]. 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. [31], 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.

1) Payment server as a TTP: Although this solution does not have its place in a fully-decentralized scheme, it is still interesting as long as the payment procedure involves a specific agent referred to as the bank. In this case, and with most blind signature schemes, the bank can act as a Trusted Third Party (TTP) and relay the blind signature while checking its validity to trigger the payment.

This can for instance be done with RSA blind signatures [28], where the bank can check the signature of the blinded message.

2) Fair Exchange Signature Scheme (FESS): We can fairly assume that most payment schemes can be triggered with an electronic signature by the client on the financial transaction, which may then be transferred to the SP via the bank or network. Many e-cash systems already use such signatures to validate transactions (e.g. Bitcoin, Dashcoin, Zerocash).

Besides, it is most likely that a chosen blind signature scheme will end with a final message from the signer to the enquirer, thus activating the blind signature.

From these two assumptions, we can design a more generic scheme that may work with any blind signature protocol. This sub-protocol is described in Appendix A. It can nonetheless be simplified for the cases when a third party may check the blind signature without having to know any sensitive information. This is the type of signatures that we consider most appropriate for our implementation.

3) Simplification for special cases: As mentioned previously, although generic as it is, the last protocol can be greatly simplified if it is adapted to some particular blind signature schemes. It can actually be adapted to all the blind signatures schemes in which a third party can verify the validity of the last message enabling the blind signature, without having learned any hidden information.

This can be applied for RSA blind signatures [28]; the protocol is then reduced to the following steps, later formalized in the Algorithm 5:

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 – Mr^e.
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 (see Section VI-D2).

Algorithm 5 RSA Blind signature (client side)

```
procedure BlindSig($SP, K, context$)
    $T \leftarrow SP.\textsc{GetTransactionToSign}(context)$
    $(e, n) \leftarrow$ public key of $SP$ for the blind signatures
    $r \leftarrow \text{random}(), k \leftarrow \textsc{Hash}(K)$
    $m \leftarrow kr^e \mod n$
    $\tilde{T} \leftarrow SP.\textsc{GetCommitment}(T, m)$
    if $\neg\textsc{CheckSig}(SP, \tilde{T}, (\textsc{BlindCommit}, m, T))$ then
        fail with Wrong commitment from $SP$
    end if
    $s_T \leftarrow \textsc{SignTransaction}(T)$
    $\bar{m} \leftarrow SP.\textsc{AskBS}(s_T)$
    if $\bar{m}^e \mod n \neq m$ then
        Send $(T, m, \tilde{T}, s_T)$ to a few trackers
        if They don't send back some valid $\bar{m}$ then
            fail with Bad behavior
        end if
    end if
    return $\bar{m}r^{-1}$
end procedure
```

VII. Analysis

We will analyze our scheme by following each one of the objectives we set in Section IV.

A. *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 (even if we use multi-signatures [19] for s_3 in the feedback data block). As a matter of fact, the network usage only consists in a query to the SP that is concerned, and his answer. That is if everything goes well but it should be the case since it is in the own interest of the SP (see Section VII-D3). Also, the client only has to check two electronic signatures, which is still very quickly computable.

Besides, this efficiency of the reputation retrieval procedure is not detrimental to the efficiency of the other procedures. These other procedures are indeed also doable in constant time for the clients, as well as for the trackers if we don't handle the verification procedures to them. Finally, 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 fame and presumably to their computational capacity.

B. *User anonymity preservation*

1) Assumptions: For user anonymity to be preserved, we have to make the following assumptions:

Assumption 1. *The anonymizing network interface that is used (e.g. Tor [26]) is not breached.*

Assumption 2. *The blind signature scheme that is used is indeed unlinkable.*

Assumption 3. *The NIZK proofs and the BLS signatures are secure, and they do not reveal any hidden information. Also, the hashes that are used throughout the paper are considered resistant to collisions.*

Assumption 4. *The correctness of the Merkle tree $\mathcal{T}_3$ is ensured (see Section VII-D3).*

Assumption 5. *The number of different identities (i.e. people owning different certificates) colluding with a specific SP can be upper bounded by some constant number Ω_{SP}. The number of different identities declaring that they went through a transaction with a SP while they did not can be upper bounded by some constant number Δ_{SP}.*

The anonymizing method that is proposed, Tor, comes with some limitations of its own ([32], [33]), but the establishment of such an anonymous connection is out of scope of this paper.

On the other hand, most of the blind signature schemes (at least [28] and [31]) have been proven to be unlinkable. Even if they are breached in the future, meaning that everybody is able to sign messages on behalf of a SP, the unlinkability property will still be preserved, thus ensuring the anonymity of the past signatures.

2) Anonymity set: Although feedback records and transactions are unlinkable, a specific SP knows which transactions he made, and his current feedback records are necessarily coming from clients who were involved in these transactions. We can therefore only ensure anonymity among a subset of the whole population. This subset of possible clients for a given feedback record will be called the anonymity set of the client who posted this record.

For major SPs, this will not be a problem since they have a lot of buying clients and each one of them takes more or less time to submit a feedback – assuming that they do send one. For small SPs, one may have to wait a while before sending his feedback, in order to better preserve his privacy and increase the size of his anonymity set. This waiting parameter may be adjusted by each user to better fit their own needs.

An important point though is that no other anonymity-preserving reputation system is capable of avoiding this issue. However, our system will ensure that each client gets an anonymity set suited to his or her needs. The role of the Merkle tree $\mathcal{T}_3$ maintained by the SPs is for the clients to be able to calculate their anonymity set, and decide whether to wait or not before engaging in a transaction. This tree contains the identities of all the users who went through a transaction, including the ones who did not send any feedback. This information, coupled with the feedback records themselves, will give enough knowledge for the clients to compute a lower bound of their anonymity set.

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

Proof. Thanks to the Assumption 4, the client can retrieve the number N of different identities having declared that they went through a transaction with the SP. At most $\Omega_{SP}+\Delta_{SP}$ of these identities can be ruled out by the SP as buyers for a transaction that they did not initiate, because of the Assumption 5. Let M be the number of feedback records so far. Then if the client waits until $N - \Omega_{SP} - \Delta_{SP} - M \geq A$, he gets an anonymity set of at least A buyers. $\square$

Further analysis could even get a weaker condition for the same anonymity set, but this condition is sufficient to demonstrate our property.

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

Proof. The client / feedback unlinkability could only be threatened by the following items:

- The transition and relation between the transaction phase and the subsequent feedback submission steps
- The communication when the client sends the feedback record to the SP (or the trackers)
- The feedback record itself

These three points do not alter the unlinkability because of the following reasons:

- The blind signature (only link between the financial transaction and the future events that might occur due to

feedback submission) is unlinkable: Assumption 2. The anonymity set is controlled by the client: Property 1.
- The connection is anonymous: Assumption 1.
- The date d is not an issue thanks to the statement of Property 1. We do not consider the feedback itself as an issue against unlinkability here, since the clients are well aware of the implications of any specific content they might publish – this is the very goal of a reputation system. The token is unlinkable because of the Assumption 3. K has been randomly chosen so it is unlinkable too. s_1 and s_2 are signatures by publicly known entities on the previous data: they are also unlinkable.

□

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

Proof. This type of unlinkability is based on these two facts:
- The two tokens of two feedback records from the same client to two different SPs are unlinkable because of the Assumption 3.
- The remaining threat to the unlinkability of those two feedback records is the linkability of both feedbacks to the same client. But, once again, this unlinkability is within the anonymity set defined by the client, according to the Property 2.

□

C. 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, as it was discussed in the corresponding section, this CA could be comprised of several entities and thus decentralized, using some multi-signature scheme such as [19].

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.

The presence of some TTP and the requirement that it should be non-malicious are nonetheless often essential in network protocols, in particular for the bootstrap or peer discovery phase. Many well known decentralized protocols – BitTorrent, Bitcoin, etc. – rely on singular servers in order to retrieve the list of peers (and/or other information): these are respectively trackers, and the servers corresponding to hard-coded IP addresses and DNS records such as *bitseed.xf2.org*[1]. The key point is for these servers to have a minimal role in the protocol and/or to be discovered and proven wrong in case of fraudulent behavior. A significant improvement still is to rely on a group of several servers instead of a unique one; these servers correspond to our network of trackers. This way, we can still afford to have a minority of malicious servers, as long as the majority behaves correctly.

[1]See https://en.bitcoin.it/wiki/Satoshi_Client_Node_Discovery

D. Robustness

1) Assumptions: This scheme is robust against colluding malicious peers – meaning that it remains correct and functional – given that the following two assumptions hold:

Assumption 6. *If the peers have a list of N running trackers, at least $\lfloor \frac{N}{2} \rfloor + 1$ of them are honest.*

Assumption 7. *The CA only delivers certificates to individual and unique users.*

2) Correctness:

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.*

We will see below that the SPs have no interest in behaving maliciously when they can be detected. This condition is therefore not a real issue.

Proof. Assumption 3 implies that only the clients who have certificates can create valid tokens, since the zero-knowledge proof verifies these certificates. Moreover, the token value that is disclosed is blindly linked to the corresponding identity. This means that one cannot create more valid tokens about some SP with different values $\hat{value}$ than the number of certificates he owns, which should be one according to Assumption 7. In addition to this, only one feedback record per token value may be accepted in the current reputation blocks. This is ensured with the Merkle tree $\mathcal{T}_2$, and would be detectable otherwise.

The signature on the one-time public key K in the NIZK proof prevents anyone from reusing the feedback record after it has been integrated inside a block, because each new feedback record should contain a new K. This is verified thanks to $\mathcal{T}_1$, and would be detectable otherwise. Also, the feedback record cannot be changed by someone other than the one who created it, because of the signature s_2 on the whole record. The blind signature scheme also prevent this person from creating several valid feedback records from a single transaction.

We have seen that these feedback records cannot be forged. The Assumption 7 restricts the number of valid feedback records without transaction to Ω_{SP} (i.e. the people to whom the SP accepts to sign keys K without any transaction).

The only remaining threat would be the refusal of the SP to include some feedback records inside the next reputation block. This is not possible because of the verification of the clients who submit feedback (see Section VI-C3).

□

3) Preventing and detecting malicious behavior: Malicious behavior from the SPs is to be prevented in a dissuasive way. As we already mentioned, trackers, SPs, clients, everybody can participate in the verification of each SP, and potentially report a malicious behavior. The implications of a proof of such behavior should be undesirable for the SP: a decrease of his reputation score or even an expulsion from the reputation system are conceivable outcomes.

Even though all the data does not need to be checked, the amount of verification and threat should guarantee the statistical non-profitability of any misbehavior of a SP. This way, we may assume for correctness that they are not misbehaving. To prevent important fraud from remaining undiscovered, we can also do some targeted verification. Should the daily aggregated feedback of a specific SP suspiciously increase one day, for instance, it should be all the more verified.

VIII. 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 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. The protocol remains secure as long as a majority of the trackers behaves correctly. We introduce the construct of tokens implemented with Non-Interactive Zero Knowledge proofs to prevent Sybil attacks. For future work, some improvements can be considered to further minimize the role of the trackers. Using our system of tokens to allow any peer to become a tracker while still preventing Sybil attacks would be a conceivable approach.

References

[1] "Bittorrent protocol." [Online]. Available: http://www.bittorrent.com/

[2] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008. [Online]. Available: https://bitcoin.org/bitcoin.pdf

[3] M. Barbaro and T. Zeller Jr, "A face is exposed for aol searcher no. 4417749," August 2006. [Online]. Available: http://query.nytimes.com/gst/abstract.html?res=9E0CE3DD1F3FF93AA3575BC0A9609C8B63

[4] P. Resnick and R. Zeckhauser, *Trust among strangers in internet transactions: Empirical analysis of eBay' s reputation system*, ch. 6, pp. 127–157. [Online]. Available: http://www.emeraldinsight.com/doi/abs/10.1016/S0278-0984%2802%2911030-3

[5] J. R. Douceur, "The sybil attack," in *Proceedings of 1st International Workshop on Peer-to-Peer Systems (IPTPS)*, 2002. [Online]. Available: http://research.microsoft.com/apps/pubs/default.aspx?id=74220

[6] R. Merkle, "A certified digital signature," in *Advances in Cryptology CRYPTO 89 Proceedings*, ser. Lecture Notes in Computer Science, G. Brassard, Ed. Springer New York, 1990, vol. 435, pp. 218–238. [Online]. Available: http://dx.doi.org/10.1007/0-387-34805-0_21

[7] D. Chaum, "Blind signatures for untraceable payments," in *Advances in Cryptology*, D. Chaum, R. Rivest, and A. Sherman, Eds. Springer US, 1983, pp. 199–203. [Online]. Available: http://dx.doi.org/10.1007/978-1-4757-0602-4_18

[8] E. Pavlov, J. Rosenschein, and Z. Topol, "Supporting privacy in decentralized additive reputation systems," in *Trust Management*, ser. Lecture Notes in Computer Science, C. Jensen, S. Poslad, and T. Dimitrakos, Eds. Springer Berlin Heidelberg, 2004, vol. 2995, pp. 108–119. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-24747-0_9

[9] S. Dolev, N. Gilboa, and M. Kopeetsky, "Efficient private multi-party computations of trust in the presence of curious and malicious users," *Journal of Trust Management*, vol. 1, no. 1, p. 8, 2014. [Online]. Available: http://www.journaloftrustmanagement.com/content/1/1/8

[10] T. Dimitriou and A. Michalas, "Multi-party trust computation in decentralized environments in the presence of malicious adversaries," *Ad Hoc Netw.*, vol. 15, pp. 53–66, Apr. 2014. [Online]. Available: http://dx.doi.org/10.1016/j.adhoc.2013.04.013

[11] O. Hasan, L. Brunie, E. Bertino, and N. Shang, "A decentralized privacy preserving reputation protocol for the malicious adversarial model," *IEEE Transactions on Information Forensics and Security*, vol. 8, no. 6, pp. 949–962, 2013. [Online]. Available: http://dx.doi.org/10.1109/TIFS.2013.2258914

[12] E. Androulaki, S. G. Choi, S. M. Bellovin, and T. Malkin, "Reputation systems for anonymous networks," in *Proceedings of the 8th International Symposium on Privacy Enhancing Technologies*, ser. PETS '08. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 202–218. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-70630-4_13

[13] R. Petrlic, S. Lutters, and C. Sorge, "Privacy-preserving reputation management," in *Proceedings of the 29th Annual ACM Symposium on Applied Computing*, ser. SAC '14. New York, NY, USA: ACM, 2014, pp. 1712–1718. [Online]. Available: http://doi.acm.org/10.1145/2554850.2554881

[14] E. Anceaume, G. Guette, P. Lajoie Mazenc, N. Prigent, and V. Viet Triem Tong, "A Privacy Preserving Distributed Reputation Mechanism," Oct. 2012. [Online]. Available: https://hal.archives-ouvertes.fr/hal-00763212

[15] P. Lajoie-Mazenc, E. Anceaume, G. Guette, T. Sirvent, and V. Viet Triem Tong, "Efficient Distributed Privacy-Preserving Reputation Mechanism Handling Non-Monotonic Ratings," Jan. 2015. [Online]. Available: https://hal.archives-ouvertes.fr/hal-01104837

[16] A. Schaub, R. Bazin, O. Hasan, and L. Brunie, "A trustless privacy-preserving reputation system," IFIP SEC - Privacy, 2016.

[17] J. Bethencourt, E. Shi, and D. Song, "Signatures of reputation," in *Proceedings of the 14th International Conference on Financial Cryptography and Data Security*, ser. FC'10. Berlin, Heidelberg: Springer-Verlag, 2010, pp. 400–407. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-14577-3_35

[18] A. Narayanan and V. Shmatikov, "Robust de-anonymization of large sparse datasets," in *Security and Privacy, 2008. SP 2008. IEEE Symposium on*, May 2008, pp. 111–125.

[19] A. Boldyreva, "Threshold signatures, multisignatures and blind signatures based on the gap-diffie-hellman-group signature scheme," in *Public Key Cryptography PKC 2003*, ser. Lecture Notes in Computer Science, Y. Desmedt, Ed. Springer Berlin Heidelberg, 2002, vol. 2567, pp. 31–46. [Online]. Available: http://dx.doi.org/10.1007/3-540-36288-6_3

[20] E. Bursztein, M. Martin, and J. Mitchell, "Text-based captcha strengths and weaknesses," in *Proceedings of the 18th ACM Conference on Computer and Communications Security*, ser. CCS '11. New York, NY, USA: ACM, 2011, pp. 125–138. [Online]. Available: http://doi.acm.org/10.1145/2046707.2046724

[21] E. Duffield and D. Diaz, "Dash : A privacy-centric crypto-currency," 2014. [Online]. Available: https://www.dashpay.io/wp-content/uploads/2015/04/Dash-WhitepaperV1.pdf

[22] 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*, May 2014, pp. 459–474.

[23] J. Camenisch and M. Stadler, "Proof systems for general statements about discrete logarithms," Institute for Theoretical Computer Science, ETH Zurich, Tech. Rep. 260, Mar. 1997.

[24] D. Boneh, B. Lynn, and H. Shacham, "Short signatures from the weil pairing," in *Advances in Cryptology ASIACRYPT 2001*, ser. Lecture Notes in Computer Science, C. Boyd, Ed. Springer Berlin Heidelberg, 2001, vol. 2248, pp. 514–532. [Online]. Available: http://dx.doi.org/10.1007/3-540-45682-1_30

[25] J. Groth and A. Sahai, "Efficient non-interactive proof systems for bilinear groups," in *Advances in Cryptology EUROCRYPT 2008*, ser. Lecture Notes in Computer Science, N. Smart, Ed. Springer Berlin Heidelberg, 2008, vol. 4965, pp. 415–432. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-78967-3_24

[26] R. Dingledine, N. Mathewson, and P. Syverson, "Tor: The second-generation onion router," in *Proceedings of the 13th Conference on*

USENIX Security Symposium - Volume 13, ser. SSYM'04. Berkeley, CA, USA: USENIX Association, 2004, pp. 21–21. [Online]. Available: http://dl.acm.org/citation.cfm?id=1251375.1251396

[27] A. Jøsang and R. Ismail, "The beta reputation system," in *In Proceedings of the 15th Bled Conference on Electronic Commerce*, 2002.

[28] S. Goldwasser and M. Bellare, "Lecture notes on cryptography," 2001, page 235. [Online]. Available: http://cseweb.ucsd.edu/~mihir/papers/gb.html

[29] J. Camenisch, M. Koprowski, and B. Warinschi, "Efficient blind signatures without random oracles," in *Security in Communication Networks*, ser. Lecture Notes in Computer Science, C. Blundo and S. Cimato, Eds. Springer Berlin Heidelberg, 2005, vol. 3352, pp. 134–148. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-30598-9_10

[30] T. Okamoto, "Efficient blind and partially blind signatures without random oracles," in *Theory of Cryptography*, ser. Lecture Notes in Computer Science, S. Halevi and T. Rabin, Eds. Springer Berlin Heidelberg, 2006, vol. 3876, pp. 80–99. [Online]. Available: http://dx.doi.org/10.1007/11681878_5

[31] D. Chaum, A. Fiat, and M. Naor, "Untraceable electronic cash," in *Advances in Cryptology CRYPTO 88*, ser. Lecture Notes in Computer Science, S. Goldwasser, Ed. Springer New York, 1990, vol. 403, pp. 319–327. [Online]. Available: http://dx.doi.org/10.1007/0-387-34799-2_25

[32] D. Kesdogan, D. Agrawal, V. Pham, and D. Rautenbach, "Fundamental limits on the anonymity provided by the mix technique," in *Security and Privacy, 2006 IEEE Symposium on*, May 2006, pp. 14 pp.–99.

[33] M. Reed, P. Syverson, and D. Goldschlag, "Anonymous connections and onion routing," *Selected Areas in Communications, IEEE Journal on*, vol. 16, no. 4, pp. 482–494, May 1998.

[34] J. Liu, R. Sun, and K. Kwak, "Fair exchange signature schemes," *Science China Information Sciences*, vol. 53, no. 5, pp. 945–953, 2010. [Online]. Available: http://dx.doi.org/10.1007/s11432-010-0065-1

Appendix A
Generic Fair Exchange Scheme

As stated in Section VI-E2, we describe here a generic way of implementing a fair exchange with any blind signature scheme, under the two fair assumptions previously mentioned.

Let M be the (valid) message to sign, and I an invalid message that is fixed, public and always the same. The scheme is described as follows, and illustrated in Figure 3:

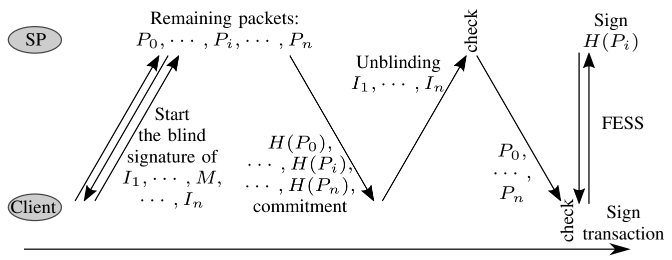
SP
Client
Remaining packets:
$P_0, \cdots, P_i, \cdots, P_n$
Start the blind signature of $I_1, \cdots, M, \cdots, I_n$
$H(P_0), \cdots, H(P_i), \cdots, H(P_n)$, commitment
Unblinding $I_1, \cdots, I_n$
check
$P_0, \cdots, P_n$
check
Sign $H(P_i)$
FESS
Sign transaction
time

Fig. 3. Blind signature scheme using FESS (H: hash function)

1) The client asks the SP to blindly sign the message M as well as some number of invalid messages I (a few ones should be enough, since the SP wants to avoid proofs of bad behavior at all costs – see Section VII-D3). The messages are to be shuffled, for the SP not to be able to know which is the real message. The two sides interact until the last step of the signature scheme.
2) The server sends the hashes of the last data packets it has to send to make the signatures valid, as well as a signed commitment on the current situation.
3) The client saves this data, tells which is the real message – without revealing it – and also shares the information that will prove to the SP that all the other messages are indeed I.
4) After checking this information, the SP sends back the last packets it was holding for all the signatures of I. Should he fail to do so, send wrong messages or ones that do not correspond with the previous hashes, the client is free to use the SP's last commitment as well as his unblinding messages in order to ask for the trackers' help (see the first kind of proof of bad behavior in Section VI-D2).
5) If everything worked fine so far, the two peers shall proceed to the Fair Exchange Signature Scheme (FESS) as described in the paper by J. Liu et al. [34]. The SP is to sign a commitment on the previously disclosed hash of the remaining packet that the client should receive. The client on the other hand is to sign the financial transaction so that it takes effect. The SP should be the one to disclose his signature (*keystone* in [34]), thus making the one of the client valid.
6) The SP is then expected to deliver the last packet to the client. Should he fail to do so, the signature on its hash that has been disclosed by the FESS protocol is a proof that can be transmitted to the trackers to either indirectly get the packet or generate a proof of bad behavior. With the last packet received, the client has his blind signature of M.

Appendix B
Other algorithms

This first algorithm (Algorithm 6) helps the clients update their list of trackers. It takes a possibly outdated list of trackers l as only argument, and returns the updated list named r_v. The information that the client uses to update the list is the tuple (d', l_t, b_t^1, s_t) returned by the trackers that have been contacted. As introduced in Section VI-B1, l_t is supposed to be the current list of trackers, b_t^1 the hash table containing proofs of bad behavior of intentional withdrawal for each of the former trackers. s_t is the signature of the sender over the rest of the message.

Algorithm 7 is used to check the hash in the block headers that the SPs provide. It is called in Algorithm 3 and it result is an input of the following algorithm.

Algorithm 8 is used to retrieve the block data and is also called in Algorithm 3. While $hinfo$ comes from the previous algorithm, $header$ and s_4 come from Algorithm 1.

Algorithm 9 checks for the validity of a single feedback record that is to be part of a data block. See Section VI-B2 for the description of the feedback record itself.

Algorithm 6 Update the list of trackers

```
procedure UpdateTrackers($l$)                         ▷ Update the list $l$
    $M, P \leftarrow \text{EmptyMap}(), d \leftarrow \text{today}()$
    for all $t \in l$ do
        if $t \in \text{keys}(P)$ then                        ▷ $t$ is corrupted
            continue
        end if
        $(d', l_t, b^1_t, s_t) \leftarrow t.\textsc{GetTrackers}(d)$    ▷ see VI-B1
        if $d' \neq d \vee \neg\textsc{CheckSig}(t, s_t, (d', l_t, b_t))$ then
            continue                          ▷ Wrong date or signature
        end if
        if $l_t \cap \text{keys}(b^1_t) \neq \varnothing$ then
            $P[t] \leftarrow (\textsc{TInconsistent}, (d', l_t, b^1_t, s_t), \varnothing)$
            continue
        end if
        for all $t' \in \text{keys}(b^1_t)$ do
            if $\textsc{CheckProof}(b^1_t[t'])$ then
                $P[t'] \leftarrow b^1_t[t']$
            else
                $P[t] \leftarrow (\textsc{WrongTProof}, (d', l_t, b^1_t, s_t), t')$
            end if
        end for
        if $l_t \in \text{keys}(M)$ is defined then
            $M[l_t] \leftarrow M[l_t] \cup \{t\}$
        else
            $M[l_t] \leftarrow \{t\}$
        end if
    end for
    if $\|\text{keys}(P) \cap l\| = \|l\|$ then
        fail with No more valid trackers
    end if
    $r_c \leftarrow 0, r_v \leftarrow \varnothing$
    for all $M[\alpha]$ do
        $M[\alpha] \leftarrow M[\alpha] \backslash S$
        if $\|M[\alpha]\| > r_c$ then
            $r_c \leftarrow \|M[\alpha]\|, r_v \leftarrow \alpha$
        end if
    end for
    if $r_c < \lfloor \frac{\|l\| - \|S\|}{2} \rfloor + 1$ then
        fail with Too few honest trackers
    end if
    for all $t \in r_v \cap \text{keys}(P)$ do
        Send proof $(t, P[t])$ to all trackers in $r_v \backslash t$
    end for
    for all $t \in r_v \backslash M[r_v]$ do
        $p \leftarrow (\textsc{WrongTList}, (d', l_t, b^1_t, s_t), \varnothing)$
        Send proof $(t, p)$ to all trackers in $r_v \backslash t$
    end for
    return $r_v$                                ▷ Updated list of trackers
end procedure
```

Algorithm 7 Check the hash in the block header

```
procedure CheckHash($SP, header, s_4$)
    $(d, v_T, v_{tot}, h, s_3) \leftarrow header$
    $hinfo \leftarrow \textsc{AskHashInfo}(SP, d)$
    $(h', r_1, r_2, r_3) \leftarrow hinfo$
    if $h \neq \textsc{Hash}(SP, h', r_1, r_2, r_3)$ then
        Ask a valid $hinfo$ for $header$ to a few trackers
        if They did not obtain one then
            fail with Bad behavior
        end if
    end if
    return $hinfo$
end procedure
```

Algorithm 8 Retrieve the block data

```
procedure DatRet($SP, header, s_4, hinfo$)
    $(d, v_T, v_{tot}, h, s_3) \leftarrow header$
    $(h', r_1, r_2, r_3) \leftarrow hinfo$               ▷ see CheckHash
    $(d', data, s_5) \leftarrow \textsc{AskBlockData}(SP, d)$
    if $(d \neq d') \vee \neg\textsc{CheckSig}(SP, s_5, (d', data))$ then
        Ask a few trackers to retrieve valid block data
        if They did not obtain one then
            fail with Bad behavior
        end if
    end if
    if $h' \neq \textsc{Hash}(data)$ then
        $p \leftarrow (\textsc{DatRet}, (header, s_4, hinfo, data, s_5), \varnothing)$
        Send $(SP, p)$ to the trackers
        fail with Bad behavior
    end if
    return $(data, s_5)$
end procedure
```

Algorithm 9 Verify a feedback record

```
procedure CheckRecord($SP, rec, d$)
    $(d', v, c, \mathbf{t}, K, s_1, s_2) \leftarrow rec$         ▷ See Section VI-B2
    if $d \neq d'$ then
        return False                                  ▷ Wrong date
    end if
    if $v$ is out of range then
        return False                  ▷ Incorrect feedback value
    end if
    if $\neg\textsc{VerifyToken}(\mathbf{t}, v_{SP}, K)$ then
        return False                                 ▷ Wrong token
    end if
    if $\neg\textsc{CheckSig}(SP, s_1, K)$ then
        return False                       ▷ Wrong blind signature
    end if
    if $\neg\textsc{CheckSig}(K, s_2, (d', v, c, t, K, s_1))$ then
        return False                     ▷ Wrong record signature
    end if
    return True                          ▷ Valid feedback record
end procedure
```