Uncheatable Reputation for Distributed Computation Markets

Bogdan Carbunar[1] and Radu Sion[2]

[1] Computer Science, Purdue University (carbunar@cs.purdue.edu)
[2] Computer Science, Stony Brook University (sion@cs.stonybrook.edu)

Abstract. Reputation systems aggregate mutual feedback of interacting peers into a "reputation" metric for each participant. This is then available to prospective service "requesters" (clients) for the purpose of evaluation and subsequent selection of potential service "providers" (servers). For a reputation framework to be effective, it is paramount for both the individual feedback and the reputation storage mechanisms to be trusted and able to deal with faulty behavior of participants such as "ballot stuffing" (un-earned positive feedback) and "bad-mouthing" (incorrect negative feedback). While, in human-driven (e.g. Ebay) environments, these issues are dealt with by hired personnel, on a case by case basis, in automated environments, this ad-hoc manner of handling is likely not acceptable. Stronger, secure mechanisms of trust are required.

In this paper we propose a solution for securing reputation mechanisms in computing markets and grids where servers offer and clients demand compute services. We introduce *threshold witnessing*, a mechanism in which a minimal set of "witnesses" provide service interaction feedback *and* sign associated ratings for the interacting parties. This endows traditional feedback rating with trust while handling both "ballot-stuffing" and "bad-mouthing" attacks. Witnessing relies on a challenge-response protocol in which servers provide verifiable computation execution proofs. An added benefit of this solution is that it naturally offers assurances of computation result correctness.

Keywords: Trust, Reputation Systems, Electronic Commerce

1 Introduction

In a reputation system, satisfaction feedback provided by interacting entities is aggregated and used in the construction of a "reputation" metric of each participant. This metric is then to be used by prospective service clients in evaluating and selecting among potential servers. One example of a reputation system is eBay. In a typical scenario, following a sale, the buyer provides a satisfaction rating which is then stored in a publicly available reputation profile of the seller which can be used by prospective buyers to decide whether to buy from this particular seller.

In the case of eBay, interacting entities are human. One could envision leveraging a paradigm of interaction feedback reputation in fully automated digital interactions, for example in distributed servicing systems. The promise of such reputation frameworks [3, 5, 6, 16, 17, 21, 28] in distributed computing environments

is to offer a low cost, scalable method for assessing reliability (and possibly level of trust) of connected system entities.

A naturally scalable system of reciprocal peer interaction ratings allows for a gradual build of a "reliability" metric for each connected entity. In a simple, traditional paradigm, every time two entities interact, they rate each other's performance and "attach" that rating to a history corresponding to the particularly rated party. This history is usually preserved by an (often specialized) authoritative, trusted party (e.g. ebay.com). In fully-connected (possibly financial) incentive based systems (such as ebay.com) this method seems to be suited well, due to its low overhead and simplifying assumptions.

However, in environments with less defined incentive models, possibly functioning under more loosely connected communication assumptions (e.g., ad-hoc and peer to peer communication patterns) centralized trust is a costly, often unrealistic proposal. A distributed trusted alternative for reputation management is required. Moreover, in hostile settings, malicious behavior can interfere significantly with the ability to provide and manage interaction and service ratings (possibly with the purpose of inflicting reputation damage to the competition or the system itself). Nevertheless, these are the type of frameworks where a reputation paradigm would yield the most benefits, not only because of its scalability and virtually zero-cost, but mainly because of its potential to provide feedback to security and resource managers, essential in such ad-hoc and faulty (possibly malicious) settings.

This is why, for a reputation framework to be effective, it is paramount for both the individual feedback and the reputation storage mechanisms to be trusted and able to deal with faulty behavior of participants such as "ballot stuffing" (un-earned positive feedback) and "bad-mouthing" (incorrect negative feedback). While, in human-driven (e.g. Ebay) environments, these issues are dealt with by an army of hired individuals, on a case by case basis, in automated environments, this ad-hoc manner of handling is likely not acceptable. Stronger, secure mechanisms of trust are required. Additionally, especially in open, dynamic, possibly hostile environments, of concern are truly malicious system entities (e.g., intruders), mounting denial of service attacks.

Introducing the ability to deal with faulty behavior in distributed computing economies, in which parties are paid for services is especially challenging. In traditional paradigms reputation in such frameworks is enforced by an economic incentive mechanism and non-incentive driven (possibly malicious) behavior is often ignored. Incentives for competing parties to rate each-other truthfully for service interactions are often hard to identify and enforce without a strong security mechanism. Moreover, due to the competitive nature of these systems, malicious parties would actually benefit from offering negative ratings if asked about the competition. This can effectively result in denial of service attacks (to the rated parties) and/or disrupted flow of business (e.g., paid requests for computing cycles).

In other words, incentive-enforced reputation, makes sense mostly when the incentive model is limited to the system itself. It cannot account for "external" (non-considered) incentives that could provide enough "fuel" for a destructive behavior, counter to the internal system-incentive structure. A party could be

outright malicious with the one and only purpose to succeed in a denial of service (DOS) attack. It becomes important to be able to deal with such hostile scenarios as part of normal system operation when managing reputation.

In this paper we introduce a solution for secure reputation management in a distributed computing environment. We believe this to be a first required step in the integration of reputation as a trusted automated assessment mechanism in distributed computing environments. Our solution is composed of two major elements: a proof of computation method (an extension of the "ringer" concept first introduced in [13]) and a "threshold witnessing" mechanism. In the witnessing protocol, a set of sufficient "witnesses" are gathered to witness service interactions and subsequently sign a document certifying a new associated rating. The witnessing is coupled with a mechanism of computation proofs which provides an (arbitrary high) confidence level that a particular set of computations was indeed performed by a given party. This is required in witnessing to make sure that ratings are in fact correlated to the quality of the result. The main contributions of this paper include: (i) the proposal and definition of the problem of securing rating correctness and their associated semantics (computational result correctness) in distributed computing markets, (ii) a solution proposing the use of *threshold witnessing* and *computation proofs* to produce securely signed ratings and (iii) the evaluation thereof, (iv) an extension to the *ringers* concept for arbitrary computations and an analysis of its applicability to various classes of functions.

This paper is structured as follows. Section 2 introduces the main system and adversary models as well as some of the basic construction primitives. Section 3 overviews, details and analyzes our solution and its building blocks. Section 4 surveys related work and Section 5 concludes.

2 Model and Tools

2.1 Communication and System Model

Let $\mathtt{n}$ be the average total number of uniquely identified (e.g., through a numeric identifier $\mathtt{Id(X)}$)) processes or participants in the system at any given point in time. Due to the potentially dynamic nature of real systems (with participants joining and leaving continuously), defining $\mathtt{n}$ precisely is inherently hard.

The purpose of the system is to provide a market for computations. Participants can export CPU cycles that others can use in exchange for payment. We assume that there exists a finite set of computation "services", $\{\mathtt{f_1}, ..., \mathtt{f_s}\}$ and each participant has the ability to perform them, albeit at different costs. Let Alice be a service provider and Bob a service requester. In such a computing market, as part of a service request, Bob specifies a certain amount he is willing to pay for it as well as some additional quality of result constraints, e.g. time bounds. Bob's aim is to maximize the investment, given a set of computations it needs to perform.

For any interaction between Alice ($\mathtt{Id(Alice)} = \mathtt{A}$) and Bob ($\mathtt{Id(Bob)} = \mathtt{B}$), let there be a unique session identifier, e.g., a composition of the current time and the identities of the two parties, $\mathtt{sid(A, B, time)} = \mathtt{H(A; B; time)}$. We will use the

notation $\mathtt{sid}$ when there is no ambiguity. Let $\mathtt{f}$ be a service provided, $\mathtt{f} : \mathbb{D} \rightarrow \mathbb{I}$, and let $(\mathtt{x_i})_{\mathtt{i} \in [1,\mathtt{a}]} \in \mathbb{D}$ be the computation inputs.

As it is not central to our contribution, to model costs and execution times we are proposing the following intuitive model: (i) for a computation $\mathtt{f}$ and a given input data set $\{\mathtt{x_1}, ..., \mathtt{x_a}\}$ the amount of associated CPU instructions $\mathtt{NI}(\mathtt{f})$ required to compute it, can be determined easily [3]; for every system participant $\mathtt{X}$ both (ii) the execution time per CPU instruction $\mathtt{TPI}(\mathtt{X})$ and (iii) the charged cost per time unit $\mathtt{CPT}(\mathtt{X})$ are publicly known and easily accessible to every other participant [4]. While this model can be made more complex, we believe it fits best the current scope. Thus, for a given function or computation load, any entity can determine every other's entity associated cost and execution time. This is an important element in the process of matching a particular service request (including time and cost constraints) to the most appropriate service provider. For more details see Section 3.2.

There exists a universal time authority that provides the current absolute time value (e.g., UTC) with a certain precision ϵ_t to every participating entity. There exists a PKI [22], that can distribute certified public keys to all participants. This enables them to sign and check signatures of others. We propose to use this same infrastructure to also provide verifiable threshold signatures [25]. More precisely, we use it to distribute to each participant a certified *master secret key share* and the *master public key*. The secret key shares are set up such that any $\mathtt{c} + \mathtt{1}$ participants can sign an arbitrary message with the master secret key, and the correctness of any signature share can be verified by anyone in a non-interactive manner ($\mathtt{c}$ or less participants cannot perform the same operation, see Section 2.4).

We assume that the underlying distributed communication layer offers the following types of communication channels: (i) secure point to point between two entities (cost: ψ_{pp}), (ii) secure multicast within a group (cost per multicast: ψ_{mcast}) and (iii) broadcast (cost per broadcast: ψ_{bcast}). The multicast channel allows group creation and message delivery to group members. Additionally, as the main focus of this paper is not on the communication layer we assume that there exist join/leave protocols for participating entities that enable them to both become aware of and communicate with the other entities in the system. The only related extension that our solution proposes is to the join protocol so as to enable a particular entity to gain knowledge of existing reputation values in the system (see Section 3.2).

Let the ratings be numeric in our system. We say that a reputation is a *trustworthy enclosure* of a rating. More precisely, the reputation of participant $\mathtt{X}$ is $\mathtt{rep}(\mathtt{X}) = \mathtt{S}_{\mathtt{MK}}(\mathtt{Id}(\mathtt{X}), \mathtt{rating}(\mathtt{X}), \mathtt{T})$, where $\mathtt{S}_{\mathtt{MK}}(\mathtt{M})$ denotes message $\mathtt{M}$ signed with the secret master key, $\mathtt{rating}(\mathtt{X}) \in [0, 1]$ is $\mathtt{X}$'s rating (a higher value is a better rating), and $\mathtt{T}$ is the creation time of the reputation. Additionally, upon receiving the results $\mathtt{r}$ of an interaction of $\mathtt{X}$ performed in time $\Delta\mathtt{T}$, let $\rho()$ be any function

[3] Also, for illustration simplicity of we also assume that for each x_i, this amount is the same.

[4] We could enter into more technical details on how this is accomplished (e.g., this value could be distributed together with the party's public key) however this would bring nothing new.

that aggregates $\mathtt{r}$ and $\Delta\mathtt{T}$ with the previous rating of $\mathtt{X}$, to create the new rating of $\mathtt{X}$; the new rating value of $\mathtt{X}$ becomes $\mathtt{rating}_{\mathtt{new}}(\mathtt{X}) = \rho(\mathtt{rating}_{\mathtt{old}}(\mathtt{X}), \mathtt{r}, \Delta\mathtt{T})$.

2.2 The Adversary

Let $\mathtt{c}$ be the upper bound on the number of active faulty participants at any point in time (e.g., no more than $\mathtt{c}$ participants can collude, crash or act dishonestly). The mechanisms proposed here are always secure but most efficient when the size of the input data sets $\mathtt{a}$ is truly large. More specifically, when on average, $\mathtt{a} > (2\mathtt{c} + 1)$ holds for (most of) the computation jobs in the system. To bring efficiency even for smaller input sets, one could envision a mechanism for gathering multiple inputs over a time period, until a critical mass is attained and only then submit them for execution.

The role of reputation ratings is then to allow service requesters to choose service providers with a good history. Of concern here are scenarios of *ballot stuffing* and *bad-mouthing* [7] in which participants collude in order to build fake pasts. In ballot stuffing un-earned good reputation ratings are provided to service providers by colluding clients. The main purpose of bad-mouthing is to provide incorrect negative ratings, possibly for the competition. Additional attacks include Sybil attacks, where malicious participants create fake identities in order to obtain more than $\mathtt{c}$ key shares and mobile virus attacks, where discovered configuration weaknesses are repeatedly exploited to corrupt more than $\mathtt{c}$ hosts. Section 3.4 shows how our solution secures also against such attacks.

2.3 Ringers

The *ringers* concept was first introduced in [13]. The main idea behind it is to (cheaply) provide computation proofs for the evaluation of a certain hypothesis over a large input set where only certain items will match (e.g., interesting patterns in [1]). Here we propose to use ringers in a distributed computing market context where a computation needs to be performed against *all* the data items in the input set.

In their initial version ringers work as follows. The first underlying assumption is that the computations in the system are non-invertible one-way. A service client wishes to get one of these computations $\mathtt{h}$ computed for a set of inputs, $\{\mathtt{x}_1, ..., \mathtt{x}_\mathtt{a}\}$ by a service provider. To perform the computation it first computes a challenge ("ringer") to be submitted along with the inputs to the service provider. This challenge is exactly the result of applying $\mathtt{h}$ to one of the inputs $\mathtt{h}(\mathtt{x}_\mathtt{t})$, where $\mathtt{t} \in [1, \mathtt{a}]$ is not known to the service provider. The implicit assumption here is that computing $\mathtt{h}$ for the entire input set is significantly more expensive than for a single item in the input (e.g, if $\mathtt{a}$ is large enough). The client then submits $\{\mathtt{x}_1, ..., \mathtt{x}_\mathtt{a}\}$ *and* $\mathtt{h}(\mathtt{x}_\mathtt{t})$ to the service provider. In addition to the normal computation results $\{\mathtt{h}(\mathtt{x}_1), ..., \mathtt{h}(\mathtt{x}_\mathtt{a})\}$ the service provider is expected to return also (as a computation proof) the correct value for $\mathtt{t}$. Due to the non-invertible nature of $\mathtt{h}$, a correct return provides a confidence of actual computation over the set of inputs.

The main power of the ringers lies in the assumed non-invertibility of the performed computations. To directly fake a proof (and produce a "valid" $\mathtt{t}$), the

service provider would have to either: (i) act honest and perform a computations or (ii) cheat and perform only $0 \leq w < a$ computations hoping it finds the ringer in the process and, if not, guess. The probability to succeed in cheating is positively correlated to the amount of work performed; over the course of multiple interactions it can be forced to arbitrary small values. For more details see Section 3.4. The ringer construct (for arbitrary computations) in this paper is obtained by "wrapping" results in one-way, random crypto-hash functions. In other words, we lift the assumption of one-way non-invertibility for the computations in the system; h can be any function. The ringer challenge submitted to the service provider becomes now $H(h(x_t))$ where H() is a one-way non-invertible cryptographic hash function. Thus, instead of the assumed one-wayness of computations, our extension puts the main power of ringers in the non-invertibility and randomness of the cryptographic hash deployed. Additionally, we extend the adversary model to also consider "guessing" (see Section 3.3).

2.4 Verifiable Threshold Signatures

The model of verifiable threshold signatures [25] consists of a set of n participants and a trusted dealer. Since we already assume the existence of a decentralized trusted infrastructure providing public key distribution, see Section 2.1, we can use it to play the part of the trusted dealer. Initially, the trusted infrastructure needs to generate a master public key PK, a verification key VK, n shares of a master secret key $SK_{i:1..n}$ and n verification keys $VK_{i:1..n}$. Each participant P_i receives PK, VK and its shares SK_i and VK_i, each certified by the trusted infrastructure. Additional secret and verification key shares can easily be generated later on, for the use of new participants that join the system. This is not generating a high overhead, requiring only the computation of a polynomial and an exponentiation [25]. The signature verification algorithm takes a message, its signature and the master key PK and determines if the signature is correct. The signature share verification algorithm takes a message, PK, VK, the signature share of process P_i, and VK_i and determines if the signature share is valid. The share combination algorithm takes as input a message, $c + 1$ valid signature shares for the message and PK and produces a valid signature of the message. Any c processes cannot collude and produce a valid signature of a message.

3 Solution

3.1 Overview

At an overview level our initial solution proceeds as follows (Figure 1 (a)). Bob wishes to get a given computation f executed over a set of input data items $\{x_1, ..., x_a\}$, in exchange for payment. Both the payment and the amount of time he is willing to wait for service completion are upper-bounded. In an initial *witness selection* phase (**step 1**, Section 3.2), Bob selects a set of $2c + 1$ computation "witnesses" W_i (this provides a threshold-secure way of avoiding illicit ratings). He then sends to all of them (via multicast) a service request including f, the inputs, the payment and target execution time upper bounds (**step 2**). The wit-

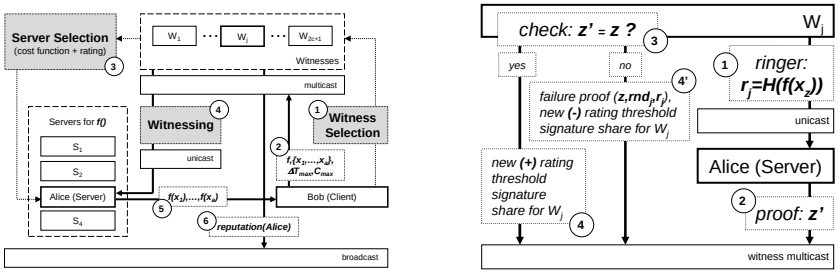

Fig. 1. (a) Solution Overview (b) Building Block: Witnessing Protocol

nesses then perform a distributed *server selection* process (**step 3**, Section 3.2), at the end of which the least-costly, best-reputation, available server is selected to perform `f` for Bob. As the adversary model in Section 2.2 guarantees a majority of the witnesses are honest and non-colluding, this process is to complete successfully. Let the selected server be "Alice". Note that the selection of Alice is not under the control of Bob. Alice is provided `f` and the input data set and the witnesses then initiate the process of *threshold witnessing* (**step 4**, Section 3.2) by sending (each in turn) a set of challenge ringers to Alice. Upon executing the computation Alice completes the witnessing process by returning the execution proofs associated with the challenge ringers to the witnesses, as well as the actual computation results back to Bob (**step 5**). Finally, depending on the proof correctness, the witnesses sign (using verifiable threshold signatures, Section 2.4) a new rating (a combination of the previous rating and "good" if correct proofs or "bad" otherwise) for Alice and distribute (broadcast) it (**step 6**, Section 3.2). If the rating does not change it is not distributed.

3.2 Building Blocks

Rating Storage Management. Let us start by first exploring the way the actual reputation information is stored. As the rating of each participant is ultimately a numeric value, in itself it can be easily altered. Our solution for introducing trust in reputation values is to allow their creation only if a certain number of participants agree on that reputation. For this, the reputation of participant `X` is stored signed with the secret master key, as $\mathtt{S_{MK}(Id(X), rating(X), T)}$. Then, we use verifiable threshold signatures (see Section 2.4) to allow no less than $\mathtt{c+1}$ participants to sign a reputation with the secret master key. Thus, the reputation of a participant can be created or changed only if at least $\mathtt{c+1}$ participants agree on the participant's new rating.

Every participant stores the most recent reputation for every other participant together with: the time it takes the participant to execute an instruction, *time per instruction (TPI)*, and the amount charged by the participant per unit of its processor time, *cost per time (CPT)* (see Section 2.1). Both values are constant and signed with the participant's private key before distribution.

In order for participants to store consistent reputations, we need each joining participant to acquire the current view of the system, and a change in one's reputation to be advertised to all. It is thus straightforward to see that at most $\mathtt{c}$ participants may have an incorrect view of the system, since we assume that at most $\mathtt{c}$ participants are faulty. A joining participant has then to make its presence known through a broadcast message, followed by the transfer of reputations knowledge of at least $2\mathtt{c}+1$ participants, already part of the system. Specifically, when a participant $\mathtt{X}$ receives the broadcast message of a new participant, $\mathtt{J}$, it stores $\mathtt{J}$'s identity under an initial, pre-agreed upon rating. $\mathtt{X}$ then retrieves the current time, $\mathtt{T}$, and if the selection test is positive, that is, $\mathtt{H(Id(X),T)\ mod\ \lceil n/(2c+1+e)\rceil = 0}$, $\mathtt{X}$ sends back to $\mathtt{J}$ its collection of reputations. $\mathtt{J}$ waits to receive $\mathtt{c}+1$ replies and then for each participant only stores the most recent reputation. Since we assume that the current time can be retrieved with error ϵ_t, each participant uses only the most significant bits of $\mathtt{T}$ in order to perform the selection test with the same value of $\mathtt{T}$. $\mathtt{e}$ is a positive integer, used to ensure that at least $2\mathtt{c}+1$ of the participants will be selected.

Witness Selection. Before exporting a job, service client $\mathtt{B}$ first needs to select $2\mathtt{c}+1$ witnesses, ensuring that even if $\mathtt{c}$ participants are faulty, a majority, at least $\mathtt{c}+1$, will be honest and alive for the duration of the protocol. Since $\mathtt{B}$ already stores the reputations of all the participants, the witnesses can be elected randomly among them. This corresponds to **step 1** in Figure 1 (a). In **step 2**, $\mathtt{B}$ creates a multicast channel for the witnesses and sends the (signed) job description: $\mathtt{f}$, the set of input values $(\mathtt{x_i})_{\mathtt{i}\in[1,a]}$, the maximum time $\mathtt{B}$ is willing to wait for job completion, $\Delta\mathtt{T_{max}}$, and the maximum amount $\mathtt{B}$ is willing to pay for the computation, $\mathtt{C_{max}}$, a signed digest $\mathtt{S_B(H(f,(x_i)_{i\in[1,a]},\Delta T_{max},C_{max}))}$, along with a certificate containing $\mathtt{B}$'s public key, meant to prevent integrity attacks.

Server Selection. The $2\mathtt{c}+1$ witnesses need to first select the most suitable service provider (see **step 3** in Figure 1 (a)). This is performed subject to the following constraints. First, all participants $\mathtt{X}$ for which $\mathtt{TPI(X)\times NI(f)\times a}$ is greater than $\Delta\mathtt{T_{max}}$, or $\mathtt{TPI(X)\times CPT(X)\times NI(f)\times a}$ is greater than $\mathtt{C_{max}}$, are not further considered. Secondly, the participant with the best reputation among the remaining ones is selected. Let that participant be Alice ($\mathtt{A}$). Even with $\mathtt{c}$ faulty witnesses, no less than $\mathtt{c}+1$ witnesses will select $\mathtt{A}$.

Next, $\mathtt{A}$ is added to the witness multicast group. The first witness in a certain order [4] ("the leader") multicasts the job description received from $\mathtt{B}$. If the other witnesses receive this message, they know that $\mathtt{A}$ also received it, and stop; else, after waiting a small amount of time, the next witness in the ordering assumes leadership and sends this multicast. This continues until the witnesses hear the expected job description. In Section 3.2 we show that in fact the number of expected multicasts is exactly one.

Threshold Witnessing. The 4th step in Figure 1 (a), detailed in Figure 1 (b), depicts the service witnessing operation. This operation requires the $2\mathtt{c}+1$ witnesses to first export $\mathtt{B}$'s computation to $\mathtt{A}$, then verify the accuracy of the

computation performed by $\mathtt{A}$, and based on the quality of the service performed, compute and sign the new rating of $\mathtt{A}$. The essence of the service witnessing operation is the usage of ringers (see Section 2.3). We now show how the threshold witnessing operation is performed securely by $\mathtt{2c+1}$ external witnesses.

Ringer Generation. Each witness $\mathtt{W_{j:1..2c+1}}$ selects one (or a small random number of values – we illustrate here the case with one single value for clarity) random value $\mathtt{x_z}$ from the input set $(\mathtt{x_i})_{\mathtt{i}\in[1,a]}$ specified by $\mathtt{B}$ in the job description and computes a ringer $\mathtt{r_j = H(f(x_z))}$. Based on the identities of $\mathtt{A}$ and $\mathtt{B}$ and the current time, $\mathtt{W_j}$ generates a unique session identifier, $\mathtt{sid}$, (see Section 2.1). The purpose of $\mathtt{sid}$ is to prevent replay attacks by introducing a freshness element. Then $\mathtt{W_j}$ computes $\mathtt{S_{W_j}(H(Id(W_j), sid, r_j))}$, and sends its identifier, $\mathtt{Id(W_j)}$, $\mathtt{sid}$, the ringer $\mathtt{r_j}$, together with the signed digest and $\mathtt{W_j}$'s public key certificate to $\mathtt{A}$ (**step 1** in Figure 1 (b)). When $\mathtt{A}$ receives such a message, it verifies $\mathtt{W_j}$'s signature. Note that even though $\mathtt{A}$ knows $\mathtt{r_j}$ and $\mathtt{A}$ may collude with a subset of the witnesses, none of them actually knows the $\mathtt{x_z}$ value generated by an honest witness. Moreover, since at most $\mathtt{c}$ witnesses can be malicious, at least $\mathtt{c+1}$ witnesses will be honest and generate good ones; these are enough to ensure $\mathtt{A}$'s cooperation.

$\mathtt{A}$ waits to receive $\mathtt{2c+1}$ valid messages for the same session identifier, $\mathtt{sid}$. If, within a given time frame, starting with the receipt of the first ringer, $\mathtt{A}$ receives less than $\mathtt{c+1}$ such messages, it ignores the job received. Otherwise, $\mathtt{A}$ sends a multicast message to all the witnesses that participated. The message contains a concatenation of all the signed ringers received. The witnesses that receive this message, inquire the remaining witnesses for their ringers. If the remaining witnesses, less than $\mathtt{c+1}$ of them, show that they chose a different service provider, $\mathtt{A}$ should perform the job with only the initial ringers (necessarily from honest witnesses). If the remaining witnesses reply with ringers, $\mathtt{A}$ should perform the job using all ringers. This mechanism is required to avoid a case of malicious witnesses mounting a denial of service attack in which they don't send out ringers but claim Alice to be malicious. It also handles the case of a malicious Alice claiming to have not received all the ringers.

Revealing the Ringers. Next, $\mathtt{A}$ performs the computation and reveals the input values $\mathtt{x_z}$ hidden in the $\mathtt{2c+1}$ ringers. It creates a single message containing $\mathtt{S_{w_j}(H(Id(W_j), sid, r_j))}$ and $\mathtt{S_A(H(Id(A), sid, z))}$, for $\mathtt{j = 1..2c+1}$. The message also contains the results of the computation, $\mathtt{f(x_1), .., f(x_a)}$, along with its signed digest. Note that the first signed digest was sent by $\mathtt{W_j}$, and is used to prove the value of the ringer $\mathtt{r_j}$. $\mathtt{A}$ then sends this message on the witness multicast channel (**step 2** in Figure 1 (b)). Each witness $\mathtt{W_j}$ verifies the correctness of only its own ringer, that is, $\mathtt{r_j = H(f(x_z))}$. The multicast of A is meant to prevent a witness from falsely claiming that $\mathtt{A}$ did not send back a correct answer.

If any witness $\mathtt{W_j}$ discovers that $\mathtt{A}$ did not send back $\mathtt{x_z}$ or that $\mathtt{r_j \neq H(f(x_z))}$, $\mathtt{W_j}$ sends a multicast message to all the other witnesses revealing this fact. The other witnesses are able to verify the claim by computing the correct answer to $\mathtt{W_j}$'s ringer and compare it with the answer sent back by Alice (received during $\mathtt{A}$'s previous multicast, **step 3** in Figure 1 (b)). This acts as the proof that $\mathtt{A}$ did not perform the entire computation. A negative rating is then issued.

Signature Generation. Based on $\mathtt{A}$'s current rating, $\mathtt{rating(A)}$, the returned results of the current computation, $\mathtt{r}$, and the time elapsed since $\mathtt{A}$ received the job description, $\Delta\mathtt{T}$, each witness $\mathtt{W_j}$ is able to compute $\mathtt{A}$'s new rating using the ρ function (see Section 2.1). In general, if $\mathtt{A}$ is caught cheating, either by not performing the entire computation or performing it slower than advertised, its rating will decrease, otherwise, it will increase. Each $\mathtt{W_j}$ then generates a verifiable signature share of $\mathtt{A}$'s new reputation, $\mathtt{S}^{\mathtt{shr}}_{\mathtt{W_j}}(\mathtt{Id(A)}, \rho(\mathtt{rating(A)}, \mathtt{r}, \Delta\mathtt{T}), \mathtt{T})$, where $\mathtt{T}$ is the current time and $\mathtt{S}^{\mathtt{shr}}_{\mathtt{W_j}}(\mathtt{M})$ denotes message $\mathtt{M}$ signed with $\mathtt{W_j}$'s share of the secret master key. Then $\mathtt{W_j}$ sends this value, along with its certified verification key $\mathtt{VK_j}$ (see Section 2.4) and $\mathtt{A}$'s new rating in clear, to all the other witnesses, using the group's multicast channel. Each witness waits to receive $\mathtt{c}$ correct signature shares for the same new reputation of $\mathtt{A}$ as the one generated by itself. As mentioned in Section 2.4, any participant can verify the validity of a signature share by using the master public key, master verification key and the verification key of the share, $\mathtt{VK_j}$. Additionally, since no more than $\mathtt{c}$ witnesses are malicious, an honest witness will receive at least $\mathtt{c}$ such correct signature shares, ensuring progress. Since $\mathtt{c} + \mathtt{1}$ different and correct signature shares are enough to generate a valid signature (see Section 2.4), each witness is able to generate the signed new rating of $\mathtt{A}$ locally.

Reputation Distribution. In the last stage of the protocol, depicted in **steps 5** and **6** in Figure 1 (a), the results of the computation are returned to $\mathtt{B}$ and the new reputation of $\mathtt{A}$ is distributed. Since we assume that $\mathtt{A}$ can only be lazy, if $\mathtt{A}$ performed the computation, it will send the correct results to $\mathtt{B}$. The witnesses know each other's identities and a global ordering of the group members is assumed [4]. The first witness is in charge of sending the new reputation of $\mathtt{A}$ on the broadcast channel to all the participants in the system. If during this broadcast the remaining witnesses hear the expected reputation, they stop. However, if the next witness (in the given group order) does not hear the expected reputation in a given time frame, it will itself send $\mathtt{A}$'s new reputation on the broadcast channel. This process goes on until all the witnesses are satisfied with the distributed reputation. Note that a witness cannot simply send an incorrect reputation since it will be easily detected, as it would need a fresh timestamp and to be signed with the master key, that is, by at least $\mathtt{c} + \mathtt{1}$ honest witnesses.

An optimization to this solution is to simply *ignore un-changing ratings*. In other words, if, because of the current interaction, the rating of $\mathtt{A}$ doesn't change no additional action is performed (no rating distribution). This reduces the number of broadcasts to a fraction of $\frac{\mathtt{c}}{\mathtt{n}}$ (Section 3.4).

Punishing Malicious Witnesses. If a witness acts maliciously at any stage (e.g., as part of rating signing or distribution) due to its nature, the witnessing scheme allows for immediate counter-measures upon detection. One such measure could be simply flagging the identity of the particular witness and effectively removing it from the system. Also, because the only actual effect of such malicious behavior is just a likely minor slow-down of the protocol (e.g., waiting for the necessary amount of honest signature shares) *but sure detection* (!), this constitutes a significant cooperation incentive for witnesses. In other words, the likelihood of a

witness committing such "suicidal" action can be considered minor. Thus any rational witness should act honest in the witnessing protocol, for its own survival benefit. As a result, e.g., the expected number of transmissions required for rating distribution is exactly 1.

3.3 Attacks and Improvements

Cheating and Laziness. Is *bad-mouthing* still possible? Due to the nature of the solution, issuing a bad rating requires a secured proof of non-compliance by Alice, that all witnesses agree with. This would only happen if at least one of them will show its ringer $H(f(x_z))$ for which Alice did not respond correctly with z. But, if Alice responded correctly, all messages are signed, ringers are non-invertible, and at most c of the witnesses are malicious, this is not possible.

Furthermore, a straight-forward ballot-stuffing attack, where clients create and duplicate simple compute jobs in order to artificially increase the ratings of preferred servers, is thwarted through the indirection introduced by the witnessing layer. For each job requested by a client, a server is chosen by $2c + 1$ witnesses, containing an honest majority.

Next, we ask what are the chances of malicious entities to succeed in cheating in the witnessing phase? In other words, is lazy behavior (resulting in *ballot-stuffing*) possible and how likely? An analysis on the power of ringers can be found in [13]. Without duplicating these results, here we are exploring a scenario not considered in [13]. Let us start by asking the question: For r ringers, what is the probability of "finding" (i.e., finding the rank of the corresponding input item in the item set) x of them by performing only $w < a$ work? In other words, what is the likelihood of cheating by simply finding the ringers after doing less work than required.

This can be modeled as a classical sampling experiment without replacement (retrieving x black balls out of w draws from a bowl of $(a - r)$ white and r black balls): $P_0(a, w, r, x) = \frac{\binom{r}{x} \times \binom{a-r}{w-x}}{\binom{a}{w}}$ where $x \in [\max(0, w + r - a), \min(r, w)]$. Additionally, we know the success probability of simple guessing of r ringers without performing any work is (choosing r out of a items): $P_1(a, r) = \frac{1}{\binom{a}{r}}$ A rational malicious Alice could deploy the following cheating strategy: do $w < a$ work (compute only w results) and, if not all the ringers are discovered (possible if also $r < w$), simply guess the remaining ones. It can be shown that the success probability of such a strategy is:

$$P(w, r) = \sum_{i=\max(0,w+r-a)}^{\min(r,w)} [P_0(a, w, r, i) \times P_1(a - w, r - i)] = \frac{1}{\binom{a}{r}} \sum_{i=\max(0,w+r-a)}^{\min(r,w)} \binom{w}{i}$$

To better understand what this means we depicted the behavior of $P(w, r)$ in Figure 2 for $b = 20$. It can be seen that (e.g., for $r = 5$) a significant amount of work (e.g., $w > \frac{3}{4}b$) needs to be performed to achieve even a 33% success probability. Figure 2 (b) illustrates the inverse dependency on the number of ringers r for specific values of performed work. The more challenges are presented to Alice, the less its probability of getting away with less work.

Maybe more importantly, Figure 2 (c) illustrates the inverse exponential dependency on the number of ringers r for specific values of performed work. The

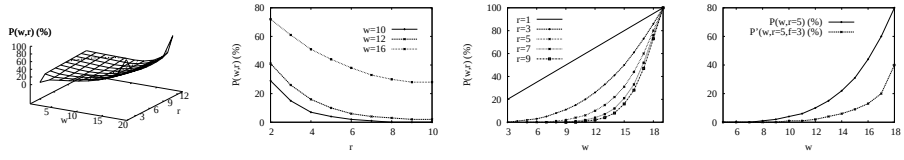

Fig. 2. The behavior of $\mathtt{P}(\mathtt{w},\mathtt{r})$ ($\mathtt{a}=20$). (a) 3-dimensional view, (b) inverse dependency of $\mathtt{r}$ to $\mathtt{w}$ (2-dimensional cut through (a)). (c) The behavior of $\mathtt{P}(\mathtt{w},\mathtt{r})$. A 2-dimensional cut through (a) showing the relationship between $\mathtt{P}(\mathtt{w},\mathtt{r})$ and the amount of performed work $\mathtt{w}$, plotted against the base case with one ringer ($\mathtt{r}=1$). (d) $\mathtt{P}'(\mathtt{w},\mathtt{r},\mathtt{f})$ and $\mathtt{P}(\mathtt{w},\mathtt{r})$ plotted for $\mathtt{r}=5$, $\mathtt{f}=3$.

more ringers are presented to Alice, the less its probability of getting away with less work. Over multiple interactions, lazy behavior is not sustainable as the probability of getting caught increases exponentially and the cheating success probability converges to 0: $\prod_{\mathtt{i}=1}^{\mathtt{v}}(\mathtt{P}(\mathtt{w_i})) \rightarrow 0$, where $\mathtt{w_i} < \mathtt{a}$ is the work performed in each of the interactions.

Fake Ringers. There exists an important issue of concern with the above scheme. Because Alice knows the number of ringers, once she finds all of them she can simply stop working.

One simple yet effective solution to this problem is to add "fake" ringers to the set of submitted ringers. In other words, instead of the witnesses computing and sending a total of $(\mathtt{r}+\mathtt{f})$ *correct* ringers, they will compute just $\mathtt{r}$ correct ones and then simply generate $\mathtt{f} > 0$ random ringer-like items and add them to the set. This has the additional benefit of reducing computation costs. Now Alice has to respond correctly only to the non-fake ones. Because she does not know which ones and how many of the challenges are fake, she is forced to execute all the queries to guarantee a correct answer (she cannot stop after it finds all the correct ones, as it doesn't know which ones and how many they are).

Introducing the fake ringers solves the issue of Alice being able to simply stop after discovering all the ringers. It also offers higher security assurances. Let us explore why [5].

Let us first assess the impact of fake ringers on the success probability of Alice's malicious behavior $\mathtt{P}'(\mathtt{w},\mathtt{r},\mathtt{f})$. To succeed, at each step, she needs to first guess *exactly* what the value of $\mathtt{f}$ is. If she is off even by one and replies with a value to a fake ringer (instead of stating it is fake), the witnesses know that Alice did not compute $f()$ over all the inputs. It can be shown that:

$$\mathtt{P}'(\mathtt{w},\mathtt{r},\mathtt{f}) = \frac{1}{\binom{\mathtt{a}}{\mathtt{r}}} \sum_{\mathtt{i}=\max(0,\mathtt{w}+\mathtt{r}-\mathtt{a})}^{\min(\mathtt{r},\mathtt{w})} \left[\frac{\binom{\mathtt{w}}{\mathtt{i}}}{\min(\mathtt{a}-\mathtt{w},\max(1,\mathtt{r}+\mathtt{f}-\mathtt{i}))}\right]$$

[5] This discussion is a summary of results by Sion et. al. in [26] where this very issue is discussed in a different context (assurances for query execution over outsourced data).

where $\frac{1}{\mathtt{min(a-w,max(1,r+f-i))}}$ is the probability of Alice guessing the value of $\mathtt{r}$ (and $\mathtt{f}$) after performing $\mathtt{w}$ work and discovering $\mathtt{i}$ correct ringers. This is so because Alice knows that clearly $(\mathtt{r}-\mathtt{i}) \leq (\mathtt{a}-\mathtt{w})$ (number of remaining ringers cannot exceed number of remaining un-considered inputs). Then, there are $(\mathtt{r}-\mathtt{i})+\mathtt{f}$ remaining possible values for $\mathtt{f}$, only one of which is correct. The $\mathtt{max}()$ needs to be considered if $\mathtt{f}=0$ and Alice discovers all $\mathtt{r}$ ringers: it knows then that $\mathtt{f}=0$. In Figure 2 (d) the evolution of $\mathtt{P}'(\mathtt{w},\mathtt{r},\mathtt{f})$ is plotted against $\mathtt{P}(\mathtt{w},\mathtt{r})$ for $\mathtt{r}=5$, $\mathtt{f}=3$. It can be seen how 3 additional fake ringers (no cost to the issuer) significantly decrease the cheating success probability of Alice (e.g., for $\mathtt{w}=17$ from 60% to 20%).

To function properly, deploying fake ringers requires the assumption that their number is random for every witnessing procedure and cannot be predicted by Alice. Now we are faced with solving the following problem: the witnesses have to make sure that, through-out time, both $\mathtt{r}$ and $\mathtt{f}$ are secret, randomly chosen and not correlated to each other or with their previous values.

But how can this be achieved in an environment where up to $\mathtt{c}$ of the witnesses could be malicious? If all the witnesses somehow agree upon values for $\mathtt{r}$ and $\mathtt{f}$, nothing is stopping the malicious ones to leak these very values to Alice, thus defeating the advantages of fake ringers all-together. To solve this, we propose the following adjustment to the ringer generation mechanism. Instead of each witness generating exactly a single correct ringer, let it generate a random, secret number of correct and incorrect ringers. As a majority of witnesses are non-malicious, even if the rest of the witnesses are not cooperating, this mechanism will result in a random value for the (total) number of fake and real ringers, neither of which are known to, or under the control of any one party.

If $\mathtt{c}$ is large enough it may warrant the argument that, due to the law of large numbers, this will result, on average, in 50% true ringers and 50% false ones. This might make it easier for Alice to approximate the moment when it can simply stop and guess $\mathtt{f}$. In that case, the following alternative can be deployed: each witness performs a separate witnessing round with Alice, (using its own random numbers of true and fake ringers). After initially performing all the computations (just once) for each such round, Alice will simply respond to the ringers challenges only. Let us also note that this alternative can be put into place at no additional cost, by having Alice not discard the computed results until all the witnesses have been satisfied. No extra computations will be required in each witnessing round.

3.4 Analysis

Communication Overhead. Let us analyze the incurred communication costs. These are composed of: (i) the initial request multicast, in the witnessing stage, (ii) one multicast with the service request (witnesses to Alice), (iii) $\mathtt{2c}+1$ unicasts with ringers (each witness to Alice), (iv) one multicast with proofs (Alice to witnesses), (v) $\mathtt{2c}+1$ multicasts with threshold signature shares (within witnesses groups) and (vi) one final broadcast with the actual signed reputation:

$$\psi_{\mathtt{comm}} = (1+1+1+(\mathtt{2c}+1))\psi_{\mathtt{mcast}} + (\mathtt{2c}+1)\psi_{\mathtt{pp}} + \psi_{\mathtt{bcast}}$$

If we normalize with respect to the cost of the point to point communication, (i.e., $\psi_{\mathtt{pp}}=1$):

$$\psi_{\mathtt{comm}} = (2\mathtt{c}+4)\psi_{\mathtt{mcast}} + \psi_{\mathtt{bcast}} + (2\mathtt{c}+1)$$

To understand this better, let us assume a simple multicast mechanism that yields an average cost (number of messages) of $\psi_{\mathtt{mcast}}(\mathtt{x}) = \beta_{\mathtt{mcast}}\mathtt{x}$ (for a group of $\mathtt{x}$ members) where $\beta_{mcast} \in (0,1)$:

$$\psi_{\mathtt{comm}} = \beta_{\mathtt{mcast}}(2\mathtt{c}+2)(2\mathtt{c}+4) + (2\mathtt{c}+1) + \psi_{\mathtt{bcast}}$$

Now, if we consider that a traditional scenario deploying reputation ratings (without witnessing and computation proofs) would only pay the communication cost of distributing the ratings ($\psi_{\mathtt{bcast}}$), the actual total incurred *overhead* for securing the rating mechanism is

$$\Delta\psi_{\mathtt{comm}} = \beta_{\mathtt{mcast}}(2\mathtt{c}+2)(2\mathtt{c}+4) + (2\mathtt{c}+1)$$

Thus, the communication overhead is of an $\mathtt{O}(\mathtt{c}^2)$ complexity order.

Let us now consider the optimization proposed in Section 3.2, namely, to not distribute un-changing ratings. Because we assume a maximum of $\mathtt{c}$ faulty parties, the ratio of negatively rated interactions is roughly $\frac{\mathtt{c}}{\mathtt{n}}$. Thus, intuitively, on average the ratio of interactions that result in "changing" ratings can also be considered roughly $\frac{\mathtt{c}}{\mathtt{n}}$. This results in an additional reduction of communication costs, as for $(1-\frac{\mathtt{c}}{\mathtt{n}})$ of the interactions, stages (v) and (vi) are not necessary:

$$\Delta\psi_{\mathtt{comm}} = \beta_{\mathtt{mcast}}(2\mathtt{c}+2)(2\mathtt{c}+1)\frac{\mathtt{c}}{\mathtt{n}} + 2\beta_{\mathtt{mcast}}(2\mathtt{c}+2) + (2\mathtt{c}+1)$$

Now the communication overhead is reduced to an order of $\mathtt{O}(\frac{\mathtt{c}^3}{\mathtt{n}})$.

Computation Overhead. The computation overhead includes: (i) the generation of $2\mathtt{c}+1$ ringers by the witnesses, (ii) the computation of $\mathtt{a}$ hashes over each function output, by Alice and (iii) the generation of $(2\mathtt{c}+1)$ threshold signature shares for the new ratings by the witnesses. Let ω_f be the cost of computing $\mathtt{f}$ for one of the inputs. We have

$$\Delta\omega_{\mathtt{computation}} = (2\mathtt{c}+1+\mathtt{a})(\omega_{\mathtt{hash}} + \omega_{\mathtt{f}}) + (2\mathtt{c}+1)\omega_{\mathtt{s}}$$

where $\omega_{\mathtt{hash}}$ is the cost of hashing a function output (when generating a ringer) and $\omega_{\mathtt{s}}$ is the cost of generating a threshold signature share.

Let us assess the complexity of computations as a function of $\mathtt{a}$, the number of input items in the request data set. For this purpose $\omega_f = 1$. Also, because ratings are numeric and of small size, and the hashes are likely computed over finite amounts of data, in the current scope, to assess overhead dependency of $\mathtt{a}$, we are considering both $\omega_{\mathtt{s}}$ and $\omega_{\mathtt{hash}}$ to be constants. The computation overheads are thus of an order of $\mathtt{O}(\mathtt{c}+\mathtt{a})$; because $\mathtt{a} > \mathtt{c}$, this becomes $\mathtt{O}(\mathtt{a})$. If we apply the optimization proposed in Section 3.2 (i.e., not to distribute un-changing ratings) these costs are further reduced with $2\mathtt{c}+1$. This would still leave the computation complexity at $\mathtt{O}(\mathtt{a})$, with smaller constants however.

Sybil Attacks. Apparently, a simple yet efficient attack, first introduced for peer to peer environments [10] could be deployed also in our framework. The attack consists in the creation of large numbers of fake identities by a single malicious entity (hence the name "Sybil attacks"). The fake identities could then be used to "help" the malicious entity in supporting its opinion, effectively implementing ballot stuffing or bad-mouthing attacks. Fortunately, since any fake participant can own at most a non-unique master secret key share (of the malicious entity that created it), it will be rendered unsuitable to act as an independent witness [6].

Mobile Virus Attacks. In scenarios where participants are similarly configured, it is possible for attackers to repeatedly exploit discovered weaknesses and rapidly corrupt a number of hosts in excess of the bound $\mathtt{c}$. Such mobile virus attacks [20], may enable malicious participants to eventually corrupt the entire network. An elegant solution for mobile viruses [20, 15] that can easily be applied in our setting, combines proactive re-computation of the verifiable secret shares of participants with the rebooting of infected hosts. The assumption is that mobile viruses can only infect a bounded, $\mathtt{c}$, number of participants within a certain time frame. Then, periodically, the share of each participant is renewed, by adding to it verifiable shares of a random polynomial, computed by other participants. Similar to the old shares, the new shares are able to reveal the initial secret, however, old and new shares cannot be combined.

Periodic share renewal provides also a simple solution to the key revocation problem. Participants that leave or are eliminated from the system, loose their ability of acting as witnesses at the next share renewal period.

4 Related Work

Reputation ratings, as an integrated mechanism of trust and quality assessment, initially emerged in social, human-driven market environments. Auction systems, such as eBay.com, allow buyers and sellers to rate each other based on the results of their transactions. The reputation of a participant is then a function of the ratings over a time period. Online community sites, such as Epinions.com, provide not only reviews for products but also ratings for the reviewers. This is achieved by allowing participants to indicate trusted or untrusted users and also rate reviews written by others. The system offers rewards (money) for highly rated reviews.

A majority of the work on providing reputations [5, 21, 28], focuses on understanding and integrating rating metrics with costs and incentive models, e.g., in service frameworks. Xiong and Liu [28] model reputations based on the transaction context and the community context factor, in addition to the total number of transactions performed by the service provider and the credibility of the feedback sources. They provide incentives for rating, by slightly increasing the reputation of a participant each time it provides feedback to others. Papaioannou and Stamoulis [21] argue that the usage of reputation values has to be complemented by

[6] It would not be able to correctly participate in the threshold signature process – at least $\mathtt{2c} + \mathtt{1}$ unique shares are required to reconstruct the threshold signature.

reputation-based policies, defining the peers eligible to interact with each other. Chen and Singh [5] propose a rating aggregation system that takes into account not only the advertised result of the transaction, but also the reputation of the entities providing the rating, and the confidence level of ratings. Their method is related to the weight propagation or transfer of endorsement mechanism used to rank pages in search engines. In [9] Dingleding et.al. explore the interaction of reputation mechanisms with anonymity frameworks. They make the valid argument that ordinary (un-secured) statistics suffice only in extremely low-threat environments; here we make the complementary argument that indeed stronger security is essential for more dynamic, automated environments with higher fault rates.

Due to the initial "social" dimension associated with reputation research, the essential issue of securing reputation mechanisms in automated environments has only recently [2, 6–8, 17, 23, 24] been identified. Resnick et al [23] provide an excellent overview of the problems of providing trust in such systems. Damiani et al. [6] extend Gnutella to include reputations not only for participants but also for individual resources. The paper describes several interesting attacks, such as pseudospoofing, ID stealth or shilling, and proposes a protocol that is secure against them. An extension of the security analysis for such attacks performed by multiple, different, colluding participants would make for interesting future work.

Dellarocas [7] uses anonymity coupled with cluster filtering to separate fair and unfair ratings, to weight ratings, and prevent attacks initiated both by service providers and service users. The paper focuses on a centralized marketplace, hence, a distributed implementation of anonymity, of computation and storage of reputations remains to be investigated. Kamvar et al. [17] propose the use of eigenvalues to estimate the trust between any two participants. The solution distributes the computation and storage of the trust values onto third parties. Since the input values for the trust computation process can be on untrusted hosts, correctness verification mechanism for the stored information are required. Otherwise, the whole computation may be compromised.

Aberer and Despotovic [2] construct reputations based on the (often reasonable) assumption that most participants are honest and therefore only negative feedback needs to be made public. In addition, the complaints are stored in a decentralized manner. It is important however to handle bad-mouthing attacks based on the creation of fake complaints for competition. Selcuk et al. [24] use trust vectors, distrust ratings and credibility ratings to identify malicious participants and content. These efforts lack an exploration of an adversary model that includes powerful, synchronized attackers.

In related research, Naor and Pinkas [19] introduce a mechanism providing secure interaction "counts" between servers and clients in web interactions; the main goal is to allow servers to securely count client interactions, while preventing count inflation or lack of client cooperation. This is an important problem to consider when web hosting services are provided for payment.

Maybe closest to our research in terms of the actual strong security goals is the research by Dewan and Dasgupta [8]. There they propose a mechanism that allows each participant to store its own reputation, as a signed chain of past transactions. Drawbacks of such an approach include the inability to deal directly

with attacks such as ballot stuffing or bad mouthing, nor with the issue of rating semantics and execution correctness. Their solution however is certainly elegant and space efficient when applicable. An additional difficulty however also resides in *securely* storing the record of the last transaction in a reputation chain.

In the area of verifiable distributed computations we already discussed work by Golle and Mironov [13]. Szada et al. [27] extend their solution to optimization functions, Monte Carlo simulations and sequential function applications. In the work of Du et al. [11], the service provider commits to the computed values using a Merkle tree. Then, the service provider is queried on the values computed for several sample inputs. The commitment prevents the service provider from changing the output of its computations.

5 Conclusions

In this work we have studied the problem of providing a secure reputation infrastructure for distributed computations. Our solution uses ringers [13] to construct computation correctness proofs. We also constrain the generation and modification of reputations, by requiring at least one non-faulty external observer to agree with the new reputation. We achieve this by employing a novel threshold witnessing mechanism, coupled with threshold signatures. We analyze the communication and computation overheads, as well as the level of security achieved. We believe that in a significant number of scenarios the goal of achieving secure trust among participants is as important as the applicability of the system, and thus, well worth the overheads. In future work we plan on building a proof of concept of the proposed mechanisms and exploring their capability in bootstrapping and maintaining trust. We also plan to increase the level of security provided by our solution against actively malicious service providers.

Acknowledgements

We would like to thank Philippe Golle for helpful discussions.

References

1. SETI @ Home. Online at `http://setiathome.ssl.berkeley.edu`.
2. Karl Aberer and Zoran Despotovic. Managing trust in a peer-2-peer information system. In *Proceedings of the tenth international conference on Information and knowledge management*, pages 310–317. ACM Press, 2001.
3. Beulah Alunkal, Ivana Veljkovic, Gregor von Laszewski, and Kaizar Aminand. Reputation-based Grid Resource Selection. In *Proceedings of the Workshop on Adaptive Grid Middleware*, New Orleans, LA, September 2003.
4. Y. Amira, G. Ateniese, D. Hasse, Y. Kim, C. Nita-Rotaru, T. Schlossnagle andJ. Schultz, J. Stanton, and G. Tsudik. Secure group communication in asynchronous networks with failures: Integration and experiments. In *The 20th IEEE International Conference on Distributed Computing Systems*, pages 330–343, 2000.
5. Mao Chen and Jaswinder Pal Singh. Computing and using reputations for internet ratings. In *Proceedings of the 3rd ACM conference on Electronic Commerce*, pages 154–162. ACM Press, 2001.

6. Ernesto Damiani, De Capitani di Vimercati, Stefano Paraboschi, Pierangela Samarati, and Fabio Violante. A reputation-based approach for choosing reliable resources in peer-to-peer networks. In *Proceedings of the 9th ACM conference on Computer and communications security*, pages 207–216. ACM Press, 2002.
7. Chrysanthos Dellarocas. Immunizing online reputation reporting systems against unfair ratings and discriminatory behavior. In *Proceedings of the 2nd ACM conference on Electronic commerce*, pages 150–157. ACM Press, 2000.
8. Prashant Dewan and Partha Dasgupta. Securing reputation data in peer-to-peer networks. In *Proceedings of International Conference on Parallel and Distributed Computing and Systems PDCS*, 2004.
9. Roger Dingledine, Nick Mathewson, and Paul Syverson. Reputation in p2p anonymity systems, 2003.
10. John R. Douceur. The sybil attack. In *First International Workshop on Peer-to-Peer Systems (IPTPS)*, pages 251–260, 2002.
11. Wenliang Du, Jing Jia, Manish Mangal, and Mummoorthy Murugesan. Uncheatable grid computing. In *Proceedings of the 24th International Conference on Distributed Computing Systems (ICDCS'04)*, pages 4–11. IEEE Computer Society, 2004.
12. David K. Gifford. Weighted voting for replicated data. In *Proceedings of the seventh ACM symposium on Operating systems principles*, pages 150–162. ACM Press, 1979.
13. Philippe Golle and Ilya Mironov. Uncheatable distributed computations. In *Proceedings of the 2001 Conference on Topics in Cryptology*, pages 425–440. Springer-Verlag, 2001.
14. J. Hassan and S. Jha. Optimizing expanding ring search for multi-hop wireless networks. In *Proceedings of IEEE GlobeCom*, 2004.
15. Amir Herzberg, Stanislaw Jarecki, Hugo Krawczyk, and Moti Yung. Proactive secret sharing or: How to cope with perpetual leakage. In *CRYPTO '95: Proceedings of the 15th Annual International Cryptology Conference on Advances in Cryptology*, pages 339–352. Springer-Verlag, 1995.
16. Audun Josang and Roslan Ismail. The beta reputation system. In *Proceedings of the 15th Bled Electronic Commerce Conference*, 2002.
17. Sepandar D. Kamvar, Mario T. Schlosser, and Hector Garcia-Molina. The eigentrust algorithm for reputation management in p2p networks. In *WWW*, 2003.
18. Dahlia Malkhi and Michael Reiter. Byzantine quorum systems. In *The ACM Symposium on Theory of Computing*, pages 569–578. ACM Press, 1997.
19. Moni Naor and Benny Pinkas. Secure and efficient metering. *Lecture Notes in Computer Science*, 1403:576–576, 1998.
20. Rafail Ostrovsky and Moti Yung. How to withstand mobile virus attacks (extended abstract). In *PODC '91: Proceedings of the tenth annual ACM symposium on Principles of distributed computing*, pages 51–59. ACM Press, 1991.
21. T.G. Papaioannou and G.D. Stamoulis. Effective use of reputation in peer-to-peer environments. In *Proceedings of IEEE International Symposium on Cluster Computing and the Grid CCGrid*, pages 259–268, 2004.
22. Michael K. Reiter, Matthew K. Franklin, John B. Lacy, and Rebecca N. Wright. The Ω key management service. In *Proceedings of the 3rd ACM conference on Computer and communications security*, pages 38–47. ACM Press, 1996.
23. Paul Resnick, Richard Zeckhauser, Eric Friedman, and Ko Kuwabara. Reputation systems. *Communications of the ACM*, 2000.
24. A.A. Selcuk, E. Uzun, and M.R. Pariente. A reputation-based trust management system for p2p networks. In *Proceedings of the IEEE International Symposium on Cluster Computing and the Grid CCGrid*, pages 251–258, 2004.
25. Victor Shoup. Practical threshold signatures. In *Proceedings of Eurocrypt*, 2000.

26. Radu Sion. Query execution assurance for outsourced databases. In *Proceedings of the Very Large Databases Conference VLDB*, 2005.
27. D. Szajda, B. Lawson, and J. Owen. Hardening functions for large-scale distributed computations. In *Proceedings of IEEE Symposium on Security and Privacy*, pages 216–224, 2003.
28. Li Xiong and Ling Liu. A reputation-based trust model for peer-to-peer ecommerce communities [extended abstract]. In *Proceedings of the 4th ACM conference on Electronic commerce*, pages 228–229. ACM Press, 2003.

Appendix

Centralized storage. Another way to store ratings is to only use $2c + 1$ dedicated, system-wide rating storers. Each participant needs to build and maintain routes to the rating storers. The initial routing information can be built using flooding, and the subsequent maintenance is only required to find alternate paths as participants crash. When a participant B needs to find the rating of A, the rating storers are contacted. Each rating storer signs the rating of A and sends it to B. B then waits to receive $c + 1$ answers whose signature verifies. The most recent valid rating for A is then considered.

Byzantine Quorum Systems. An alternative for the distributed storing of ratings is to use quorum systems. A quorum system [12] is a collection of sets of participants (quorums), such that the intersection of any two quorums contains at least one participant. Such systems are used to store timestamped information and provide the guarantee that a read operation performed for a piece of data on all the participants of a quorum will return the latest value of the data, even if previously written on a different quorum. Byzantine quorum systems [18] extend the above idea to work in the case of c failing participants. That is, any two quorums intersect in at least $2c + 1$ participants. It remains to be seen how byzantine quorums can be created and updated in dynamic scenarios. This is subject to future work.

Forcing Server Selection. An issue of some concern is to be found in the deterministic nature of the server selection process. Because computation costs are directly derived from the TPI and CPT values and reputation ratings are public, Bob could possibly generate a job choosing its input size a and required ΔT_{max} and C_{max} such that the server selection cost calculus would result in the selection of a given desired party. If Bob were to be able to select Alice, it could simply submit bogus jobs (possibly with agreed-upon inputs/outputs) and force artificial increases in Alice's rating.

Let us now analyze what it would entail to force the selection of a certain party X. X is characterized by its rating, CPT(X) and TPI(X). To be able to "distinguish" X (to force its selection) among other service providers, it could feature "extreme" CPT and TPI values. However, due to the assumption that they are relatively fixed and publicly known, it would be hard for a party to adjust them often just for this purpose and remain undetected. Upon such detection, a ban or negative rating can be enforced. Thus it is reasonable to assume that, if Alice is to maintain

an active status in the system, its CPT and TPI values cannot be altered for the purpose of matching certain service requests.

One remaining possibility for Bob to force X's selection is to adjust the actual *job parameters* to suit the given values for CPT(X) and TPI(X): produce a job description that includes f, an input data set of size a, ΔT_{max} and C_{max} values, such that for $\Delta T = TPI(X) \times NI(f) \times a$, the following holds *only for* X (!): $\Delta T < \Delta T_{max}$ and $\Delta T \times CPT(X) < C_{max}$. But is this possible?

For given CPT and TPI values, it is likely that there exists an entire set of other service providers that offer close costs and execution times. Because the CPT(X) and TPI(X) values are not easily altered for this purpose (see above argument), Bob simply doesn't have enough leverage to force these equations to be satisfied *only for* X.

There are scenarios however, where Bob could still mount this ballot-stuffing attack. For example in the case of un-even, sparse clustering in the euclidean (CPT, TPI) space, which could result in some parties being quite "unique" (e.g., no "neighbors"). An obvious solution would be to store separate rating values for each (A, B) pair of parties, i.e., group all the ratings that B offers for A and make the final rating for A a weighted sum of all the ratings from all the parties. This would reduce the impact of one particular party's attempts to stuff the ballot. This solution presents however the drawback of requiring additional storage ($O(n^2)$ total). A "middle way" fix would be to enable weighted ratings only in the case of parties rated less-than perfect (potentially malicious), reducing required storage to $O(cn)$.

Service Workflows. In the case of sequential operations, where the output of a computation performed by service provider B is used as input for a computation to be performed by another service provider A, malicious behavior of B will propagate to subsequent computations. A mechanism that allows A to be cleared of any accusation of malicious behavior, is simply requiring A to store the results of B's computation, signed by B.

Multi-level Ratings and Clustering. In scenarios where the assumption that every party can perform any computation (albeit at different costs) is too restrictive, we propose to relax it by clustering existing parties based on their ability to provide a certain service. Then, a party has different rating values for each service provided. This makes sense because trust is not an all-or-nothing proposition. Trusting a certain party to perform linear regression analysis correctly should not necessary imply it can also perform association rule mining (even if advertised) or that any of these service are performed in a timely manner.

A simple way to support reputations in such an environment could be to allow ratings for a party X to be issued only by non-competing parties, that is, parties that do not offer any of the services exported by X. However, this approach can be subject to circular collusions. We propose a solution where each service cluster is responsible for membership management operations and for maintaining the reputations of the members. Hence, each party should only store mappings between service names and cluster identifiers. An interesting option, that we plan to explore in future work, is to offer additional services of witnessing and reputation

storing. Then, the retrieval of reputations and the selection of witnesses could also be trust-based.

Rating Decay and Challenges. Another issue of significant interest here is ensuring the "comeback" ability of once un-trusted or un-reliable parties. In the absence of identity changes, a certain party can be "stuck" with a negative rating for a long time. If this rating also results in a lack of further interaction, e.g. a grid scheduler never decides to schedule jobs on a certain back-end CPU again, a closed "isolation" loop is complete. It might be of benefit to weigh ratings based on their "age" (i.e. when their associated transaction occurred) such that older ratings count increasingly less. This would "break" the loop and allow the party to redeem itself.

Additionally we propose the introduction of an explicit redemption mechanism in which the given party (aware of its bad rating) broadcasts a claim message in which it *challenges* the rating and invites the rating parties to use its services and issue new associated ratings. This new claim could of course be false. However (in both cases) this issue can be solved by introducing an exponential back-off delay mechanism (in which exponentially-increasing delays between allowed redemptions are introduced after every deceiving claim) that would curb truly malicious parties fast. Moreover, the existence of challenges, along with the assignment of the lowest rating level to joining parties, could be used to provide a mechanism for avoiding cold starts as well as counter-incentives for identity spoofing.

Alternate Communication Patterns. We have assumed until now the existence of a broadcast channel, allowing hosts to communicate efficiently and learn of each other's ratings. However, there exist distributed environments not provided with such a communication infrastructure. In a significant set of scenarios, there is an ad-hoc, possibly peer-to-peer nature to the communication infrastructure. Mobile scenarios (e.g. emergency response units, military battle-field operations) constitute such examples. Assumptions of interactive centralized trusted parties (i.e., that would manage interaction/service ratings) are difficult to sustain and simply not reasonable in many such scenarios. Moreover, often, an outright hostile nature of the environment (e.g., war-zone) requires the ability to quickly and efficiently assess and maintain a correct operation state of available resources without generalized trust assumption (e.g., enemy has infiltrated some of the infrastructure).

In the following, we propose a simple solution that requires each rating to be stored by only $2\mathtt{c}+1$ participants, called *rating storers*. We then show how the solution in Section 3.1 can be extended to fit the communication requirements of such a loosely connected network. We remind that the term reputation is used to denote the rating of a participant, signed with the master key of the system, that is, agreed upon by at least $\mathtt{c}+1$ participants.

When a new rating is created for a participant, `A`, its rating is distributed on $2\mathtt{c}+1$ rating storers located in the vicinity of `A`. The initial rating storers are found using an expanding ring search strategy [14]. If the number of neighbors of `A` is greater than or equal to $2\mathtt{c}+1$, `A`'s neighbors are used to store `A`'s rating. Otherwise, a flooding message with a TTL value set to 2 is sent by `A`. If the number

of replies received from all the contacted participants is still less than $2c + 1$, the TTL value is increased and the flooding is repeated. This process allows A to also build an efficient multicast tree for contacting its rating storers.

When participant B needs A's rating, B sends a challenge, R_B, to A. A sends this challenge to its rating storers which in turn sign A's rating and R_B, and send the signature, along with their public key certificate to A. A collects $c + 1$ such replies and forwards them to B. Even if up to c rating storers fail, A will be able to correctly perform this step. B first checks that A's message does not contain duplicates, then verifies the signatures of the rating storers and the system signatures, and considers only the most recent rating. The goal of the challenge is to ensure that A will not send a batch of old signed ratings, stored locally.

The witness selection is done using an expanding ring search. The server selection without flooding is done by using the information collected from the $2c + 1$ witnesses. That is, each witness reports the ratings it knows and the best is selected. If none of the ratings or capabilities are sufficient, continue the expanding ring search until a suitable server is found. The multicast witness communication is most likely done through B. A multicast message can simply be sent to B, that in turn has built a multicast tree to all its witnesses. Thus, all messages issued by witnesses need to be signed, to prevent B from changing their content. At the end of the service witnessing, the witnesses will act as the new rating storers.