SureRepute: Reputation System for Crowdsourced Location Witnesses

Rafael Figueiredo, Samih Eisa, Miguel L. Pardal
{rafael.figueiredo, miguel.pardal}@tecnico.ulisboa.pt, samih.eisa@inesc-id.pt
INESC-ID, Instituto Superior Técnico, Universidade de Lisboa, Portugal

***Abstract*—Location is an important attribute for many mobile applications but it needs to be verified. For example, a user of a tourism application that gives out rewards can falsify his location to pretend that he has visited many attractions and thus receive benefits without deserving them. To counter these attacks, the system asks users to prove their location through *witnesses*, i.e., other devices that happen to be at the location at the same time and that can be partially trusted. However, for this approach to be effective, it is important to keep track of the witness behavior over time. Many crowdsourcing applications, like Waze, build up reputations for their users, and rely on user co-location and redundant inputs for data verification.**

In this work, we present SureRepute, a reputation system capable of withstanding reputation attacks while still maintaining user privacy. The results show that the system is able to protect itself and its configuration is flexible, allowing different trade-offs between security and usability, as required in real-world applications. The experiments show how the reputation system can be easily integrated into existing applications without producing a significant overhead in response times.

I. Introduction

Nowadays, information systems are depending more on *geographical location*, as determined by the mobile device of the user [1]. The most common use case involves a user sharing its location with a server, and the server performs a computation and returns data or provides a service to the user. It may be important to be sure that the user was in fact in that physical location at that specific time, otherwise a user could just lie about its location and get undeserved benefits. For example, in a scenario where users receive rewards for going to visit certain locations, a user could just say that he was at any location to get rewards without really needing to be there.

Location certification systems are able to verify if a location claim is valid or not through *evidence*, usually collected by or resulting from interactions with other devices, called *witnesses*. For this approach to be effective, many and diverse witnesses are necessary. One of the ways to have witnesses without a big investment is to use crowdsourcing [2], i.e., rely on the devices of other users. The verification of location still needs to be performed, because there is the possibility of users performing tasks incorrectly by mistake or intentionally. To identify and prevent this problem, it is important to have some way of knowing if a user is trustworthy, i.e., if it has an history of correct behavior. This can be achieved by keeping track of a *reputation* for each user [3].

The SureThing project [4] addresses the need for creating and validating location certificates so that devices can make proof of their location. The project provides a framework with data formats and utility libraries available to multiple applications, such as advertising [4], smart tourism [5], smart vehicle inspections [6], and medical appointment verifications [7]. A reputation system [3] can help these systems by providing an up-to-date reputation value when they are deciding to accept a location claim.

Reputation systems gather feedback that reflects user previous behavior, aggregate that feedback and map it into a reputation score that reflects the user behavior. Reputation mechanisms can encourage trustworthy behavior and discourage participation by those who are unskilled and dishonest [3]. Despite their potential, reputation systems have their own intrinsic problems related with privacy and reputation attacks that need to be addressed.

The main goal of this work was to create a reputation system, called SureRepute, that can be integrated into crowdsourced applications, such as the SureThing domain applications. SureRepute can provide reputation values and is capable of defending against reputation attacks while still ensuring privacy to its users.

II. Background and Related Work

According to Mousa et al. [8], *reputation* is the collective assessment of how much an individual or an entity can be trusted. It is possible to compute reputation scores using feedback that members provide of each other, resulting in a reputation system that allows members to be accounted for their actions. Hoffman et al. [9] further explains that reputation can be a source to build trust in the participants, by allowing parties to decide how much they trust a participant in a given context and encouraging trustworthy behavior while discouraging dishonest participation

Abdel [10] and Mousa et al. [8] present some methods used in reputation models, that allow to create scores that reflect the users behavior. The most relevant model is the Bayesian model [8] [10] [11].

Jøsang and Ismail [11] presented a system called *beta reputation system*, that uses a Bayesian model which combines feedback by using *beta* distributions to allow for the derivation of reputation ratings. A beta distribution is a family of continuous probability density functions defined on the interval [0,

1], and indexed by two parameters α and β. The probability expectation value of the beta distribution is given by:

$$E(\alpha, \beta) = \frac{\alpha}{(\alpha + \beta)} \tag{1}$$

In the beta reputation model $\alpha = r + 1$ and $\beta = s + 1$, where r is the collective amount of positive rating and s is the collective amount of negative rating, allowing to represent the reputation score of a user. The reputation score can then be calculated based on Equation 1. and is expressed as:

$$ReputationScore(i) = E(r_i, s_i) = \frac{r_i + 1}{r_i + s_i + 2} \tag{2}$$

Where the reputation is in the range of [0,1] and the value 0.5 represents a neutral rating.

Old feedback may not always be as relevant to the reputation score, because the agent may change its behavior over time. It is important to have a model, where old feedback is given less weight than more recent one. This can be achieved by introducing a *forgetting weight*, λ, to r and s:

$$r = r_{i-1} * \lambda + r_i \quad \text{and} \quad s = s_{i-1} * \lambda + s_i \tag{3}$$

A. Reputation Attacks and Defense Techniques

Entities can bias the credibility of a reputation system in diverse ways. It can be done deliberately or not and isolated or in collusion with others. Koutrouli and Tsalgatidou [12], divide *reputation attacks* into three main categories: *Unfair recommendations* are recommendations that do not share honest feedback, which can be done by individual users or in collusion, where a group(s) of malicious users tries to destabilize the system; *Inconsistent behavior* is done by users to create a false impression of themselves, divided into two types: *Traitor*, where the user behaves properly until a strong positive reputation is gathered and then deceives others. *Discriminators*, where the user behaves properly with most entities and misbehave towards a subset of them; *Identity Management attacks* exploit the type of identity management in used and the anonymity level provided by the system. There are four types of attacks: *Whitewashing*, where a user discards its identity and enters the system with a new one to avoid bad reputation. *Sybil Attack*, where a user create multiple identities to provide large amount of false feedback. *Impersonation*, where a user portrays itself as another; *Man-In-The-Middle-Attack*, where a user tamper with the messages of others.

Koutrouli and Tsalgatidou [12] also present defenses techniques that can be deployed to reduce or prevent these attacks. The most relevant are:

- *Incorporate time in reputation estimation*: to reduce the effect of inconsistent behavior by a Traitor;
- *Penalize oscillatory behavior*: making negative ratings have more weight than positive ones in the score to reduce the effect of inconsistent behavior done by a Traitor;
- *Controlled Anonymity*: conceal the identities between users (but not from the administrator) can reduce the effect of Unfair recommendations and Discriminators;
- *Cryptographic signatures*: to reduce the affect of Unfair recommendations, Impersonation and Man-in-the-middle-attack;
- *Barriers to create and change identity* by implementing entry fees, requiring proof-of-work, not allowing the reputation of a user to fall behind the one of a newcomer, and give a low reputation value to newcomers, to discourage Unfair recommendations, Sybil attack and Whitewashing.

B. Privacy-Preserving Techniques

The privacy of a user has to do with its ability of being anonymous and not let other users record its transactions and recommendations [12]. However, a reputation system only works if the reputation is linked in some way to the identity of the user. The more information is linked to the user the less anonymity and privacy there is, but also there is an higher level of accountability, increasing the credibility of the reputation estimation.

The use of pseudonyms is a common mechanism used to protect the anonymity and privacy of the participants [13]. Instead of transmitting a real identifier of the user, all interaction with the application is performed under an alias. Pseudonyms are a good trade-off between trust and privacy, which allow correlating a pseudonym with different transactions, while still maintaining the user's real identity hidden [14].

Calado and Pardal [15] present a reward system based on points where you can create tasks for other users to do in exchange for points. To assign points to the users it uses pseudonyms as a privacy mechanism which allows for the replacement of real names to identifiers with non-related values.

Ma et al. [16] propose a decentralized privacy-preserving reputation management system for mobile crowd-sensing, in which edge nodes are deployed regionally and are responsible for collecting data as well as maintaining a consortium blockchain. *Homomorphic encryption*, which is a cryptographic technique that allows for performing computations on encrypted data without decryption, is used to update and maintain its reputation values.

C. Location Certification Systems

Location certification systems provide ways to attest and verify that a user is where s/he is saying s/he is.

Nosoushi et al. [17] presents a distributed location proof scheme called PASPORT, where mobile users can act as provers or witnesses, allowing for the generation of location proofs for each other. PASPORT integrates a form of reputation that is used for witness selection, where the prover calculates entropy-based trust score for the witnesses based on its location proof generation history. The score is high when a witness provided a few location proofs to that prover, and its low when it has issued many location proofs to that same prover. Only users that have a score above a specific threshold can be selected as witnesses.

SureThing [4] framework for location certification that provides data formats and utility libraries, used by multiple applications:

- STOP [6] - a cargo vehicle inspection system with tamper-proof records to prevent location spoofing attacks;
- SurePresence [7] - allows a user to prove his location when interacting with a kiosk. The *kiosk* represents a witness of the system that endorses the prover claims; and the *server* acts as the verifier and is responsible for storing verified location certificates;
- CROSS [5] - provides tourist itinerary verification. The application logs Wi-Fi scans during visits to locations. At the end of the trip, the application sends the collected data to the server for verification. A new strategy was also recently added by Grade et al. [18] that allows peer-to-peer witnessing between nearby users in specific stops in the itinerary.

III. SureRepute: Design

The design of SureRepute includes the attack model, the assumptions and requirements identified, the general architecture of the system, as well as the privacy protections and the score calculation technique.

A. Attack Model

We created an attack model that describes what attacks are a problem in our solution and how they will be dealt with based on what was presented in Section II-A.

SureRepute assumes that the recommendations, i.e., the user behavior reports, are always submitted by a trusted entity, that belongs to the infrastructure of the system. For example, in CROSS [5], the server is a trusted entity that can send recommendations to SureRepute based respectively on the rejection and acceptance of the collected data of the trip and suspicious behavior. We protect the confidentiality and authenticity of all exchanged messages with robust cryptography, namely, TLS with mutual authentication.

The recommenders are always trusted, so unfair recommendations are not considered a threat as the recommendations are never considered malicious. The attack model will focus on inconsistent behavior and identity management related attacks.

The reputation attacks on SureRepute and the strategies that are going to be employed by the system to prevent or reduce their impact are: have a malicious action affect much more the score than a good one; use pseudonyms but keep a global reputation that is shared among different domains, making bad behavior in a specific domain affect the reputation score in general; reduce incentive to create new identities by giving the smallest score possible to newcomers and have a slow build-up of reputation.

B. Requirements

- **R1** - Provides scores that reflect the users behavior;
- **R2** - Capable of being resilient to inconsistent behavior and identity management attacks;
- **R3** - The same user in different domains maintains a shared reputation that reflects the user general behavior;
- **R4** - Use pseudonyms to guarantee privacy to the users.

C. General Architecture

The solution was implemented using a client-server model that can be incorporated into each of the SureThing application domains represented in Figure 1, namely, CROSS, SurePresence, and STOP. The client is called SureRepute-Client, and it will serve as a way for an application domain to communicate with its own server, which is called SureRepute-Server. There is one instance of the SureRepute-Server inside each application domain. Servers will communicate with each other to maintain a shared reputation of users that are present in different application domains. The clients also interact with an identity provider that is responsible for making the identity-pseudonym translations and with the CA whenever it needs a new valid certificate.

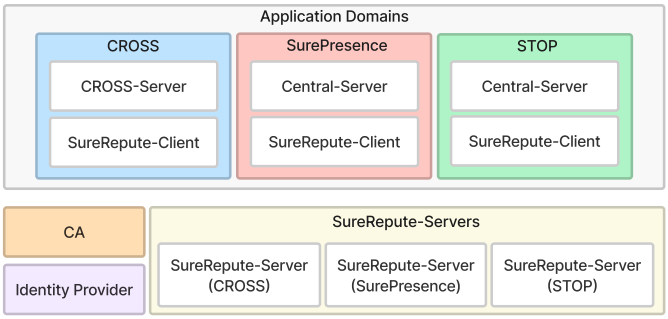


Fig. 1. SureRepute integration with SureThing Framework.

D. Assumptions

- SureRepute-Server always updates the score of pseudonyms based on the behavior reports received;
- Entities that make recommendations of user behavior always submit accurate reports;
- Accounts with the same email in different application domains are always the same person;
- All entities can always communicate with other entities and no entity ever permanently fails;
- All messages that pass the Confidentiality, Integrity, Authenticity and Freshness tests are considered secure;
- Only trusted entities can interact with the Certificate Authority (CA).

E. Privacy Protection

The privacy of the users about their identity and their behavior is ensured by separating which information each entity has. The identity provider is used as a highly trusted entity that stores the relationship between real identities and pseudonyms and only responds to requests from trusted application domains. It returns the pseudonym encrypted with the appropriate SureRepute-Server public key to ensure that a client cannot associate the pseudonyms with the user's real identity. The SureRepute-Server also has no access to the real identity of the user as it only uses the pseudonyms not the real identity of the users. Furthermore, it also does not store the history of all reported behaviors individually, instead it keeps

two variables that represent the *cumulative* value of good and bad behavior.

F. Score Calculation Technique

Our reputation score calculation will follow mainly the binomial Bayesian model, with some modifications. We will use Equation 2 to calculate the reputation score, and use a modified version of Equation 3 to calculate r and s, which are respectively the collective amount of positive and negative rating, to introduce different levels of behavior reports and different forgetting weights for the positive and negative behavior (λ_r and λ_s). SureRepute-Server can receive four types of behavior reports, and each will make a different change to r and s. The four types are:

- *Well Behaved*: where the user behaved as intended. In this case r and s will be updated with: $s = s_{i-1} * \lambda_s$ and $r = r_{i-1} * \lambda_r + 1$.
- *Accidentally Malicious*: where the user behaved maliciously, but could be due to factors outside of the user control. In this case s and r will be updated with: $s = s_{i-1} * \lambda_s + 0.5$ and $r = r_{i-1} * \lambda_r$
- *Intentionally Malicious*: where the user purposely behaved maliciously. In this case s and r will be updated with: $s = s_{i-1} * \lambda_s + 1$ and $r = r_{i-1} * \lambda_r$
- *Critically Malicious*: where the user purposely behaved maliciously in a critical situation. In this case s and r will be updated with: $s = s_{i-1} * \lambda_s + 2$ and $r = r_{i-1} * \lambda_r$.

The value of the forgetting weighs for r and s will be different, making good reports be forgotten much more quicker than bad reports, which can make bad behavior affect more the score while also making recent behavior matter more than old behavior to provide defense against the *traitor* attack.

The initial values given to s and r will also provide the initial setback before the user interactions with the system start to reflect its behavior. The higher the initial s, the more a newcomer needs to interact with the system before starting to be trusted, and thus it better defends against Sybil and Whitewashing attacks. However, it also discourages newcomers, reducing usability.

IV. SureRepute: Implementation

All entities of SureRepute were developed in Java, which runs in most operating systems and hardware platforms. We used REST software architectural style for web services. For the interface description language and canonical data format we decided to use *protocol buffers*, a data representation language and platform neutral, extensible mechanism for serializing structured data, proposed originally by Google, but now open-source. To assure data persistence and avoid data corruption, we decided to use PostgreSQL as the database language for the Identity Provider and SureRepute-Server.

We also used OpenAPI, which is a language-agnostic interface description language for REST APIs that allows users to discover and understand the capabilities of the services through the generated documentation.

Let us now look into each SureRepute entity in more detail. The *SureRepute-Client* works like a library to make remote calls to SureRepute:

- `Get Score`: obtains the score from SureRepute of one or more users (a value in the range $[0, 1]$);
- `Report Behavior`: submit behavior reports for a given user. The behavior reports can be: Well Behaved Report; Accidentally Malicious Report; Intentionally Malicious Report; Critically Malicious Report.

To do these actions, the first thing SureRepute-Client does is to request to the Identity Provider the associated pseudonym encrypted using the public key of the SureRepute-Server. The SureRepute-Client also stores in a cache the recent translations of user identity:encrypted pseudonym, and so, if the encrypted pseudonym is already in the cache the interaction with the Identity Provider does not happen. After having the encrypted pseudonym, it will then send the action to the associated SureRepute-Server of that domain. The server will return back the score of the user. If a connection cannot be made to any entity of SureRepute, SureRepute-Client returns a default score of 0.5, which represents a neutral score.

There is one *SureRepute-Server* for each application domain. The server is responsible for maintaining a reputation score for the pseudonyms that are interacting in their domain. This is done based on reports received from SureRepute-Clients that are used by each application domain. Furthermore, if the same user is interacting with multiple application domains, we maintain a shared reputation, that accounts the behavior of the user in all domains.

To avoid inconsistencies, we ensure that only one server is maintaining updates on the reputation of a user, which is called the *leader* of that pseudonym. The other servers that also receive requests for that pseudonym are considered as its *followers*. When a new pseudonym is received by a SureRepute-Server, it will need to first handle if the pseudonym is already known by any other servers. To do that, the server broadcasts the pseudonym to all other servers. When a client requests the score of a pseudonym that is already known to either the leader or the follower of that pseudonym, the server just needs to return the current stored value for that pseudonym.

The *identity provider* is a highly trusted entity that is responsible for maintaining the translations between the user identity and the respective pseudonym. It initially interacts with all SureRepute-Servers in order to get their public keys. When a SureRepute-Client requests an encrypted pseudonym for a new user, the identity provider generates a new pseudonym for them and stores this relation in the database, if the pseudonym is already known then it just gets it from the database, in both cases the pseudonym will then be encrypted using the public key of the SureRepute-Server associated with that application domain and returned back to the SureRepute-Client.

The *CA (Certificate Authority)* is responsible for validating the identity of all other entities of SureRepute and provide them a certificate that is necessary to establish TLS connections with all other entities. For that, an entity must provide a certificate signing request as well as the type of entity it is,

which is then validated. If valid, it creates a certificate that explicitly requires that the identities use specific dns names of our solution for communication.

V. Use Case: CROSS

In order to attest how SureRepute can be easily added to different application domains of the SureThing project, as well as to be able to evaluate the solution in terms of overhead and if it helps make better decisions, we decided to use CROSS [5].

The CROSS API Server component was reimplemented from Go to the Java programming language.

A new location proof strategy was added by Grade et al. [18] to use other travellers at the same location as witnesses. Now, when a traveler submits a visit, the confidence is calculated based on:

- *displacementConfidenceMultiplier*: Calculates a confidence level of the time it took to go from the previous visit to this visit based on the distance between them.
- *wiFiAPsConfidence*: Percentage of networks found by the client for this visit, compared to the total number of access point registered in the server for that location.
- *peerEndorsementsConfidence*: A visit can now provide endorsements to its location (witnesses), which are then validated. The *weight* of every valid endorsement is calculated as:

$$endorsementWeight(p, w_i) = \frac{Rw_i}{Npw_i + 1} \quad (4)$$

where p is the prover, w_i is the specific witness, Rw_i is the reputation of the witness, which is in range $[0, 1]$, which can be provided by SureRepute, and Npw_i is the number of visits in different trips that where completed in the past by the prover, which the witness w has already testified to. The *confidence* is then calculated using:

$$min\left(\frac{\sum_{i=0}^{n} endorsementWeight(p, w_i)}{endorsementWeightTarget}, 1\right) \quad (5)$$

where the endorsementWeightTarget is the weights sum value of all witnesses weight that it is intended to be achieved so that the strategy confidence is 100%, which is further defined in Grade et al. [18] work.

Based on these values, the confidence assigned to each visit is calculated as:

$$\begin{aligned} min(displacementConfidenceMultiplier \times \\ (wiFiAPsConfidence + \\ peerEndorsementsConfidence), 1) \end{aligned} \quad (6)$$

making the confidence of a visit to be the sum of the confidence gathered by the strategies the user used (capture of Wi-Fi access points and/or witness endorsements), multiplied by the confidence multiplier. The minimum is used because a traveller can use more than one strategy, and each strategy can reach 100% of confidence.

The visit confidence is then compared with the confidence threshold which is a predefined percentage that need to be met by a specific visit. If $confidence \geq confidenceThreshold$, then the visit is accepted otherwise it is rejected. If all visits of a route are accepted then the trip is considered completed and a reward can be given by the CROSS application.

To use SureRepute, CROSS-Server now has to submit reports that allows to create scores that reflects users behavior and that can be used by CROSS-Server when verifying if a visit is accepted or not. The scores of both the witnesses and the traveller are requested to SureRepute synchronously using the SureRepute-Client, when a visit is submitted, as we are dependent on their values, on the other hand report submission is done asynchronously.

Let us now analyse how the scores are used and when reports are submitted:

1) Score of Witnesses: The scores of the witnesses can be really helpful for calculating the endorsement weight that each witness provides, as detailed in Equation 4. This endorsement weight is then used to calculate the *peerEndorsementsConfidence*, which is done using the Equation 5. Using the reputation of SureRepute can be really helpful as the weight that each endorsement has will be based on the past behavior of the witness, which is represented by the reputation value.

2) Score of Traveller: After calculating the visit confidence, we need to compare it to the confidence threshold. The confidence threshold is the confidence level that needs to be achieved by the traveller on that location, which as a default value that is assigned when the location is created. But the score can increase the level of confidence that needs to be achieved based on the traveller score, making users that have a reputation ≤ 0.5, to need to achieve a higher confidence level, as they have have not proven themselves as good users yet. If $proverReputation \geq 0.5$, then the prover has a well-behaved track record and the confidence threshold is not adjusted; otherwise, we use a linear function to calculate the confidence threshold:

$$\frac{1 - confidenceThreshold}{0.5} * proverReputation \quad (7)$$

.

3) Report Submission: As explained in section III-F, there are four types of reports we can submit, and here we adapted the conditions in which each report is submitted to this specific application: Critically Malicious: submitted for the traveler when the trip is not coherent; Intentionally Malicious: submitted for the traveller whenever we are certain that the traveller acted malicious; Accidentally Malicious: submitted for the traveller when the confidence does not reach the confidence threshold; Well Behaved: submitted whenever the confidence threshold is reached. The reports are sent for the traveller and all for all witnesses that made good endorsements.

VI. Evaluation

We want to show that the reputation system is capable of creating scores that reflects user' behavior while also maintaining resistance against Traitor attacks as well as the Sybil/Whitewashing attacks. Although the score must reflect on the behavior of the reports submitted, there are still some

parameters that need to be further studied to understand what their impact is on the score given. These parameters are: the *initial score* details, i.e., the initial values for s and r, which represent the cumulative amount of bad and good behavior respectively, and that we are representing as: Details(s, r); the *forgetting weights*, i.e., λ_s, λ_r, which are represented as Forgetting(λ_s, λ_r).

A. Score Details

The initial score details can be used to choose what score to give to a newcomer when no reports were submitted yet, and it can also give the initial setback before starting to trust the behavior submitted for the user. To demonstrate this, we provide some tests where we submitted 5 reports to the same user, and we also change the number of intentionally malicious behavior reports and good behavior reports submitted. For these tests we used different initial score details: Details(1, 0), Details(5, 0), Details(10, 0), Details(100, 0), and , Details(10, 5).

The results are presented in Figure 2. You can see, with Details(1, 0) or Details(5, 0), when the user reports 5 reports of only good behavior it already gathers a positive rating, however looking at the Details(10, 0) or Details(100, 0), we can see that it still has a score below 0.5. This is because of the initial number of malicious behavior reports we give to a newcomers which serves as the initial setback before starting to trust the user real interaction with the system. After surpassing the initial setback, it is possible to verify that the user score reflects the behavior of the user, i.e., if more than 50% of reports submitted are of malicious behavior, then the score of the user is bellow 0.5.

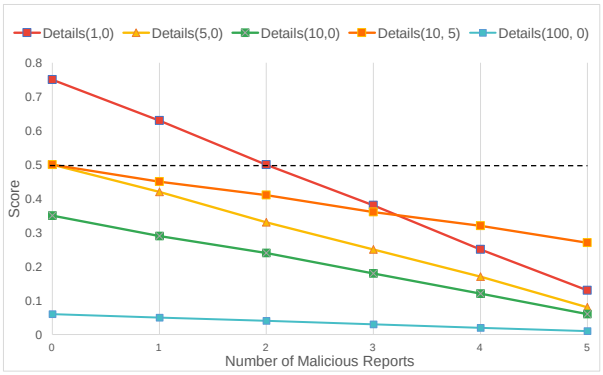


Fig. 2. 5 reports submitted using different initial score details Details(s, r) and changing the number of intentionally malicious reports submitted.

Looking into Details(10,0) and Details(10,5), the major difference is that we can modify the default score given to a newcomer. For example for Details(10,0) we have $score = (0+1)/(10+0+2) = 0.09$ and with Details(10,5) we have $score = 0.35$, which is important when we want to give an initial setback for the user but we still want them to have a default value far from 0.

This initial setback and the default score given to newcomers is really important to defend against Sybil/Whitewashing attacks, as it allows to remove the motivation for a user to get new identities by giving a low initial score to the new identity and it provides a slow build of reputation. The specific score details to use depends on the system and how much we want to prevent these attacks, in a scenario where the user needs to be highly trusted, we need to use a high value for s in the initial score details, which can be chosen based on the average amount of report submissions that are made for the user whenever it uses the system, but of course this is a trade-off between security and usability as by increasing this defense, a newcomer is more penalized by not being trusted for a long time, which can demotivate all new users. On the other hand if we want to prioritize usability we can use a lower value of s, which will make the system more susceptible to these attacks but make it easier for newcomer to be trusted.

B. Forgetting Weights

The forgetting weights allows to make recent behavior matter more than old behavior and can also be used to make malicious behavior to be forgotten slower than good behavior.

As mentioned before, the initial score details given to a user depend on the system that is being used, so to attest how the score reflects the user behavior, lets set a specific scenario: "*Considering a system for tourism trips, where for each trip a user does, an average 3 reports are made and there is a limit of 3 trips per day. The intention is that a new user has a slow build of reputation and the system is capable of defending against reputation attacks, but the initial score cannot be too low, as it would make it impossible for trips to be accepted to newcomers.*".

As on average at most 9 reports are submitted a day, using score details as Details(10, 0) or Details(10, 5) seems like a reasonable initial setback, as it needs at least a day of interaction before starting to trust the reports that are being submitted. As the system needs a default value not too low choosing Details(10,5) for the next tests based on this scenario seems the most appropriate, as it will provide an initial score of 0.35 instead of 0.09.

After setting the initial scores details as Details(10,5), we need to choose the forgetting weights for s and r. By introducing forgetting weights we make the order of when malicious behavior is submitted matter, for this reason, we defined tests where a lot of reports were already submitted, i.e., 100 reports are submitted, and did a similar test as before, but now besides changing the number of malicious reports that are submitted we also change the order that they are submitted, which can be at the start or at the end. We did these tests using different forgetting weights: Forgetting(1,1), Forgetting(0.98, 0.95), Forgetting(0.98, 0.92), Forgetting(0.98, 0.90), Forgetting(0.95, 0.92). The Forgetting(1,1) serves as the control test, because when the forgotten weights are 1 it means that no report is ever forgotten, and so submitting malicious behavior at beginning or at the end is the same.

The results are present in Figure 3 and by looking at the graph we can see that whenever λ_s is bellow 0.98, the malicious behavior starts to be forgotten too fast, as it produces

a score with a value higher than 0.5 even if 80% of the behavior is malicious. On the other hand using λ_r bellow 0.92 is also a bad idea as introduces a drastic difference of when malicious reports are submitted at the end vs at the start, looking at the graph, using Forgetting(0.98, 0.90) as the forgetting weights, the submission of 10 malicious reports already makes the score of the user to be 0.28, in comparison with 0,74 when submitted at the start, decreasing these value would increase the difference between them. Using Forgetting(0.98, 0.95) still maintains the score above 0.5 with 10 (1 day) malicious reports submitted at the end, which is not desirable as we want malicious behavior to drastically impact the score. The forgetting weights that don't forget malicious behavior too quickly but still ensure the desirable penalization on the score is Forgetting(0.98, 0.92), and so these are the values that we would choose for this scenario.

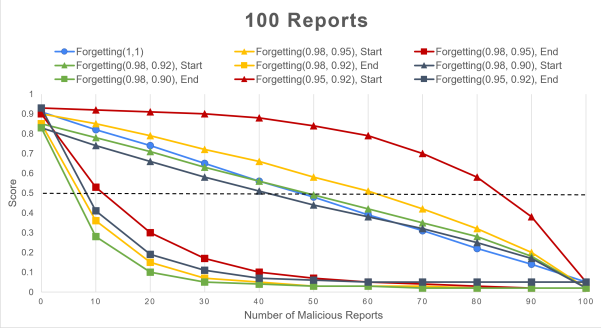


Fig. 3. 100 Reports submitted with Details(10,5), using different forgetting weights Forgetting(λ_s, λ_r) and changing the number of intentionally malicious reports submitted and its order, which can be at the start or at the end.

C. Shared Reputation

To verify that the system maintains a shared reputation whenever a user is present in multiple application domains, we created a scenario where a client submits 80 reports sequentially for the same user, 40 of intentionally malicious behavior and 40 of good behavior, first all to a single server and then dividing the report submission between two servers and then for four servers. We also did another scenario were we submit 40 reports using threads of always good behavior, with the same number of servers. The results of both scenarios showed that all servers contain the same score at the end, which proves that servers are able to create a shared reputation.

D. CROSS Visit Submission

To evaluate the overhead introduced by the calls to SureRepute from an application domain or if it helps making better decisions, we did several tests using CROSS integrated with SureRepute. For these tests we defined two environments:

- *Local environment*: CROSS-client, CROSS-Server, SureRepute-Server and Identity-Provider are running in the same computer.
- *Deployed environment*: CROSS-client is running on a local computer with the same specification as before and CROSS-Server, SureRepute-Server and Identity-Provider are deployed in the cloud in the same cluster with logical isolation.

Normally, CROSS-Client is the application that captures endorsements and/or Wi-Fi access points and submit visits to CROSS-Server during a trip. However, in this case, it is a testing client that makes the requests needed for the tests, in an easy and replayable way.

To attest the overhead introduced when multiple travellers are submitting visits, we setup a scenario where we increase the number of simultaneous submissions to verify the overhead introduced, where each submission is using 15 witnesses. The results are present in figure 4. We can see that when submitting multiple visits at the same time in any of the environments with 15 witnesses, SureRepute introduces a higher overhead than with a single visit, which is normal and can be explained by the fact that multiple threads of CROSS-Server are making requests to SureRepute. But, at most, the overhead introduced by SureRepute is of 250ms, in any of the environments. This value is low when compared with the overall times that are being done by CROSS without SureRepute. The high delay introduced when we change from the local environment to the cloud environment is unrelated with SureRepute, as the delay is much higher than the overhead introduced by SureRepute. If the delay is considered too high for a given user when submitting a trip, we can upgrade the cloud environment by changing the nodes type to process more rapidly the requests or make visit submission on the client app of the users to submit the visits asynchronously.

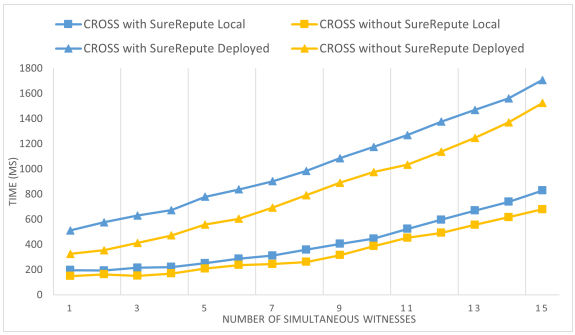
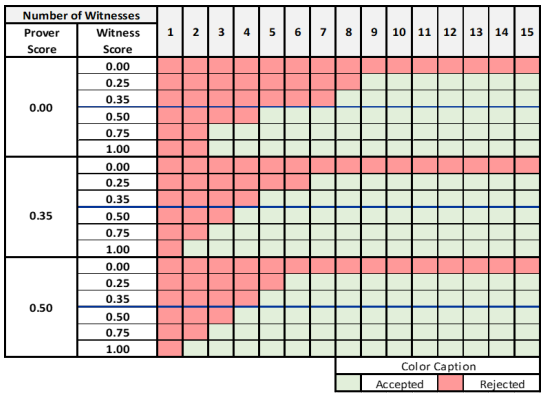


Fig. 4. CROSS Visit validation time with and without SureRepute in a local and deployed environment, varying the number of simultaneous visit submissions

To attest the benefits that SureRepute introduces to CROSS, we developed some tests, where we verify when a visit is accepted or not based only on the witness strategy, as it is where scores are used.

Consider a scenario where a visit has a default confidence threshold of 75%, and travelers submit visits with at most 15 witnesses. In these tests we change the traveler score to change the confidence threshold. When the travelers score is 0, it needs 100% ($100 - \frac{100-75}{0.5} * 0 = 0$) confidence, with 0.35, it needs 82.5% confidence and finally 0.5, where it needs 75% confidence. We also vary the Witness score from 0, 0.25, 0.35, 0.5, 0.75, 1. The number of witnesses goes from [1, 15]. We consider that using score of 0.5 for both the prover

TABLE I
CROSS ACCEPTANCE RATE OF VISITS BASED ON: PROVER SCORE, NUMBER OF WITNESSES AND WITNESS SCORE.



and witness is what simulates a scenario where SureRepute is not used. The results are shown in Table I, and as you can see, maintaining users behavior, makes the need for visits to be submitted by travellers with more or less endorsements in order to be accepted. The number of endorsements needed depend on the score of the witnesses and the traveller, when we have travellers that properly behave it removes the need for the users to have a lot of witnesses. On the other hand having a traveller and witnesses that misbehave makes the need for more witnesses. If SureRepute was not used, then all witnesses and prover would have a score of 0.5 and the user would always need at least 4 endorsements, this would benefit malicious users as they could take advantage of the system more easily.

VII. Conclusion

In this document, we studied how to build a reputation system as well as how the reputation score of its participants can be calculated. We looked into what are the attacks and defenses as well as what are the most used privacy-preservation techniques on reputation systems. We also gathered information about location certification systems that already employ reputation and presented the SureThing application domains, where we intend to insert a reputation system.

Then, we proposed SureRepute, a reputation system that can be integrated into SureThing, which allows entities inside each application domain to submit reports of the users behavior and also to get their reputation. SureRepute provides privacy protection to its users, by using pseudonyms to segregate what information each entity has access to. The score calculation technique used is based on the binomial Bayesian model, where we made possible that different types of behavior can be submitted. We also added the use of forgetting weights and the change of the initial values of good and bad behavior, in order to ensure protections against reputation attacks.

We evaluated that the system is capable of meeting its requirements and integrated SureRepute into a real use case: CROSS. The experiments showed that the integration does not introduce a significant overhead on the application domain, and it helps to make better decisions regarding witnesses, which provides added security to crowdsourced applications.

Acknowledgements

This work was supported by national funds through Fundação para a Ciência e a Tecnologia (FCT) with reference UIDB/50021/2020 (INESC-ID) and through project with reference PTDC/CCI-COM/31440/2017 (SureThing).

References

[1] A. Kushwaha and V. Kushwaha, "Location based services using android mobile operating system," *International Journal of Advances in Engineering & Technology*, vol. 1, no. 1, p. 14, 2011.

[2] J. Yang, L. A. Adamic, and M. S. Ackerman, "Crowdsourcing and knowledge sharing: strategic user behavior on taskcn," in *Proceedings of the 9th ACM conference on Electronic commerce*, 2008, pp. 246–255.

[3] P. Resnick, K. Kuwabara, R. Zeckhauser, and E. Friedman, "Reputation systems," *Communications of the ACM*, vol. 43, no. 12, pp. 45–48, 2000.

[4] J. Ferreira and M. L. Pardal, "Witness-based location proofs for mobile devices," in *IEEE 17th International Symposium on Network Computing and Applications (NCA)*. IEEE, 2018, pp. 1–4.

[5] G. A. Maia, R. L. Claro, and M. L. Pardal, "Cross city: Wi-fi location proofs for smart tourism," in *International Conference on Ad-Hoc Networks and Wireless*. Springer, 2020, pp. 241–253.

[6] H. F. Santos, R. L. Claro, L. S. Rocha, and M. L. Pardal, "Stop: A location spoofing resistant vehicle inspection system," in *International Conference on Ad-Hoc Networks and Wireless*. Springer, 2020, pp. 100–113.

[7] M. C. Francisco, "Surepresence: Location proofs for wearable and kiosk devices," Master's thesis, Instituto Superior Técnico, Universidade de Lisboa, 2021.

[8] H. Mousa, S. B. Mokhtar, O. Hasan, O. Younes, M. Hadhoud, and L. Brunie, "Trust management and reputation systems in mobile participatory sensing applications: A survey," *Computer Networks*, vol. 90, pp. 49–73, 2015.

[9] K. Hoffman, D. Zage, and C. Nita-Rotaru, "A survey of attack and defense techniques for reputation systems," *ACM Computing Surveys (CSUR)*, vol. 42, no. 1, pp. 1–31, 2009.

[10] A. J. Abdel-Hafez, "Reputation model based on rating data and application in recommender systems," Ph.D. dissertation, Queensland University of Technology, 2016.

[11] A. Josang and R. Ismail, "The beta reputation system," in *Proceedings of the 15th BLED Electronic Commerce Conference*, vol. 5, 2002, pp. 2502–2511.

[12] E. Koutrouli and A. Tsalgatidou, "Taxonomy of attacks and defense mechanisms in p2p reputation systems—lessons for reputation system designers," *Computer Science Review*, vol. 6, no. 2-3, pp. 47–70, 2012.

[13] D. Christin, A. Reinhardt, S. S. Kanhere, and M. Hollick, "A survey on privacy in mobile participatory sensing applications," *Journal of systems and software*, vol. 84, no. 11, pp. 1928–1946, 2011.

[14] J.-M. Seigneur and C. D. Jensen, "Trading privacy for trust," in *International Conference on Trust Management*. Springer, 2004, pp. 93–107.

[15] D. Calado and M. L. Pardal, "Tamper-proof incentive scheme for mobile crowdsensing systems," in *IEEE 17th International Symposium on Network Computing and Applications (NCA)*. IEEE, 2018, pp. 1–8.

[16] L. Ma, Q. Pei, Y. Qu, K. Fan, and X. Lai, "Decentralized privacy-preserving reputation management for mobile crowdsensing," in *International Conference on Security and Privacy in Communication Systems*. Springer International Publishing, 2019, pp. 532–548.

[17] M. R. Nosouhi, K. Sood, S. Yu, M. Grobler, and J. Zhang, "Pasport: A secure and private location proof generation and verification framework," *IEEE Transactions on Computational Social Systems*, vol. 7, no. 2, pp. 293–307, 2020.

[18] R. Grade, M. Pardal, and S. Eisa, "Cross city mobile application: Gamified peer-based location certification strategy," Master's thesis, Instituto Superior Técnico, Universidade de Lisboa, Portugal, 2022, currently in progress.