TÉCNICO LISBOA

SureRepute: User Reputation in Location Certification Systems

Rafael Alexandre Roberto Figueiredo

Thesis to obtain the Master of Science Degree in

Computer Science and Engineering

Supervisor(s): Prof. Dr. Miguel Filipe Leitão Pardal
Dr. Samih Eisa Suliman Abdalla

Examination Committee

Chairperson: Prof. Maria Luísa Torres Ribeiro Marques da Silva Coheur
Supervisor: Prof. Miguel Filipe Leitão Pardal
Member of the Committee: Prof. Kevin Christopher Gallagher

November 2022

I would like to dedicate this work to my dad.

For the endless support, love and encouragement throughout all my life.

For the opportunity to achieve a higher education in the field that I love.

Acknowledgments

First and foremost, I have to thank my family, for all the constant love and support they gave me throughout my entire life. Thank you so much for giving me strength to reach for the stars and chase my dreams.

I want to give a special thanks to Miguel Pardal and Samih Eisa, my advisors, for the chance to work on the SureThing project and for all their guidance and support during the making of this thesis as well as their constant belief in my capabilities.

Additionally, I also want to express my appreciation to Ricardo Grade and Lucas Vicente, my colleagues that were also working with me in the SureThing project, they always supported me and gave me really good suggestions.

Lastly, I would also want to express my gratitude to my friends for their unwavering encouragement and support.

This work was supported by national funds through Fundação para a Ciência e a Tecnologia (FCT) with reference UID/CEC/50021/2019 (INESC-ID) and through the project with reference PTDC/CCI-COM/31440/2017 (SureThing).

Resumo

A localização é importante para muitas aplicações móveis, no entanto, para garantir que está correta, precisa de ser validada. Caso contrário, por exemplo, um utilizador de uma aplicação de turismo que dá recompensas, pode falsificar a sua localização para fingir que realizou visitas e assim receber benefícios sem os merecer. Para combater estes ataques, o sistema pede aos utilizadores para provarem a sua localização através de *testemunhas*, i.e., outros dispositivos que estão no local ao mesmo tempo e que podem ser parcialmente confiáveis. O projeto SureThing segue essa abordagem, permitindo que dispositivos obtenham provas da sua localização ao comunicar com infraestrutura existente nos locais ou com outros dispositivos. Para garantir que esta abordagem é eficaz, é importante acompanhar o comportamento das testemunhas. Muitas aplicações que recebem dados fornecidos por utilizadores, como o Waze, validam informação recorrendo a dados redundantes fornecidos por vários utilizadores na mesma localização e ponderam a confiabilidade tendo em conta a reputação dos utilizadores.

Neste projeto, apresentamos SureRepute, um sistema de reputação capaz de resistir a ataques, manter a privacidade dos utilizadores e que pode ser integrado no SureThing ou noutros sistemas. Os resultados mostram que o sistema é capaz de se proteger contra ataques, mas que a configuração é flexível, permitindo diferentes compromissos entre segurança e usabilidade que são exigidos em aplicações reais. Com a integração do sistema num domínio aplicacional específico do SureThing, conseguimos mostrar que um sistema de reputação pode ser facilmente integrado em aplicações sem produzir uma grande sobrecarga no tempo de resposta.

Palavras-chave: Sistema de Reputação, Sistemas de Certificação de Localização, Privacidade, Ataques de Reputação, Defesa Contra Ataques de Reputação, Reputação

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. The SureThing framework follows this approach, allowing constrained devices to obtain proof of their location through interaction with existing infrastructure on the location or with other devices. 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 and that can be integrated into the SureThing framework or other systems.

The results show that the system is able to protect itself from reputation attacks, and the configuration is flexible, allowing different trade-offs between security and usability that are always required in real-world applications. With the integration of the system into a specific domain application, we showed how a reputation system can be easily integrated into existing applications without producing a significant overhead in the response time.

Keywords: Reputation System, Location Certification Systems, Privacy, Reputation Attacks, Defense Against Reputation Attacks, Reputation Score

Contents

List of Tables

List of Figures

Acronyms

API Application Program Interface. 27, 34, 64

CA Certificate Authority. 23–25, 31, 32

DNS Domain Name System. 31, 32

IoT Internet of Things. 65

LBS Location-Based Services. 1

LCT Location Certificate Transparency. 2, 3, 36, 37, 40, 64, 65

MITMA Man-in-the-middle attack. 11, 12

POM Project Object Model. 27

RESTful REpresentational State Transfer. 27, 28, 34

SLCT Signed Location Certificate Timestamp. 37

TLS Transport Layer Security. 25, 31

Chapter 1

Introduction

Nowadays, information systems are depending more on *location*, which led to the emergence of Location-Based Services (LBS), i.e., services that are based on the geographical location of a mobile user as determined by the mobile device [KK11]. The most common use case of a LBS involves a user sharing their location with a server, and in turn the server performs a computation that takes the location as input and returns data/services 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, without any kind of location proofs, a user could just say that he was at any location to get rewards without really needing to be there. The verification of the location can be achieved with a location certification system.

Location certification systems are able to verify if a user location claim is valid or not through the use additional location evidence, usually collected by other devices, called *witnesses*. For example, using the previous scenario where users receive rewards for going to visit certain locations, a user instead of just submitting its location it could also interact with other users in that location in order to complement the location with location proofs given by other users (witnesses), if the location proofs were provided by trustworthy user, it would allow to prove that the user is in fact in that location. 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, i.e., allow other user devices to act as witnesses.

Crowdsourcing is the use of an Internet-scale community of people to outsource a task [YAA08], which can be employed by leveraging the help of a community of users that use a system to do part of the tasks needed. Even when we have a community and many witnesses, the verification of location still needs to be performed, because there is always the possibility of users performing

tasks incorrectly either intentionally or by mistake. To identify and prevent this problem, it is important to have some way of knowing if a user is trustworthy, i.e., if it behaves correctly. This can be achieved by keeping track of the history of the users' activities, which can be done by maintaining a reputation for each user, that reflects their behavior.

The SureThing project 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 that can be leveraged by multiple applications, such as shopping mall advertising [FP18], smart tourism [MCP20], smart vehicle inspections [SCRP20] and presence verification in medical appointments [Fra21]. It also has additional components that are intended to provide Location Certificate Transparency (LCT) and third party verifiability [CERP21]. A reputation system [RKZF00] can further help these systems by allowing the additional use of a reputation value when deciding if a location claim is valid or not.

Reputation systems gather feedback that reflects user previous behavior, aggregate that feedback and map it into a reputation score, i.e., a score that reflects the user behavior, and allow remote inquiries about the reputation score. According to Resnick et al. [RKZF00] reputation mechanisms can encourage trustworthy behavior and discourage participation by those who are unskilled and dishonest. Despite their potential, reputation systems have their own intrinsic problems that need to be addressed. They have problems related to user privacy, as a reputation system only works if the reputation is linked to the users' identity in order to provide accountability, and it has to defend against reputation attacks, where users can for example have inconsistent behavior by creating a false impression of themselves in order to be able to misbehave later.

1.1 Objectives

The goal of this work was to create a reputation system called SureRepute that can be integrated into applications that make use of the SureThing framework and others, to provide reputation to its users. The reputation of witnesses can then help when deciding if a location claim is valid or not by using the score of witnesses as the level of trust we can give to witness endorsements or even to the specific user that is submitting the claim.

The specific objectives for this work can be summarized as follows:

- Develop a reputation system into the SureThing framework that can be used by multiple domains;
- Implement the reputation calculation;

- Implement protections against privacy and reputation attacks;
- Automate the deployment of SureRepute to the cloud;
- Integrate SureRepute with a specific use case: CROSS, a smart tourism application [MCP20];
- Integrate CROSS with components that provide location certificate transparency and third party verifiability [CERP21], that could contribute with behavior reports about the user;
- Evaluate SureRepute using a demo entity built for testing purposes and using a real application domain: CROSS.

1.2 Contributions

Given the objectives, the main contributions of this work are:

- Design of the SureRepute conceptual architecture, having in mind the privacy of the users and that is capable of protect against reputation attacks;
- Introduce a score calculation technique for reputation system that was created based on the binomial Bayesian model;
- Implement SureRepute, including entities such as SureRepute-Server, Identity-Provider and SureRepute-Client, as well as their configurations for automatic deployment in the cloud;
- Improve the security of CROSS, by incorporating user reputation, where a user needs to submit proofs in specific locations during a trip, with the integration with both the LCT [CERP21], which offers third-party-verifiable location certificates and with SureRepute;
- Validation of SureRepute through a detailed evaluation of the score calculation technique, the interactions between the SureRepute entities as well as through concrete tests of SureRepute in the context of the smart tourism use case with a real deployment.

Overall, our work shows that Reputation Systems can be really useful in Location Certification Systems, especially when using witnesses.

1.3 Dissertation Outline

The remainder of the document is structured as follows. In Chapter 2 we present the background and related work, which includes reputation systems, privacy protection, reputation attacks

and location certification systems. Chapter 3 presents SureRepute, its design, which includes the attack model, the requirements and assumptions, privacy techniques and score calculation technique, the implementation, which includes both the architecture and the deployment, and finally a use case. Chapter 4 presents an evaluation of the work done, divided into qualitative and quantitative evaluation. Finally, Chapter 5 presents the conclusion and the future work.

Chapter 2

Background & Related Work

To develop a reputation system, it is important to first understand their key characteristics and how to build them, which are covered in Section 2.1. A reputation system is susceptible to a variety of attacks, so it is important to build a system with protections in mind which are examined in Section 2.2. In Section 2.3 we discuss how to build a reputation system that preserves privacy and talk about what are the trade-offs between trust and privacy. In Section 2.4, we talk about location certification systems, with an emphasis on SureThing. In Section 2.5 we discuss location certificate transparency. Finally in Section 2.6 we provide a summary of what was discussed.

2.1 Reputation Systems

According to Mousa et al. [MMH+15], *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. [HZNR09] 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.

Trust and reputation are often referred as the same term in the literature, but there is a slight difference between them. According to Putra et al. [DDKJ21] trust is a subjective belief of an entity's behavior that grows as more interactions occur, whereas reputation is the aggregated opinion or trust degree of an entity that comes from other entities that it had previously interacted. Hendrikx et al. [HBC15], affirms that "*reputation is not what character someone has, but rather what character others think someone has*" and it defines a reference

model for reputation systems, that has three parties: *Trustor*, *Trustee* and *Recommender*. The *Trustor* wants to trust and interact with a target entity which is the *Trustee*. For a *Trustor* to make a trust decision about a *Trustee*, it will evaluate its own interactions with the *Trustee*, as well as feedback given by *Recommenders* about interactions they had with the Trustee.

2.1.1 Building Steps

Mousa et al. [MMH+15], identifies four main phases for building a reputation system:

- *Information Collection*: in this phase, the information needed to create a reputation is collected. It can be done through: *watchdog module*, where sensing reports that are related to the same task are combined to assess the quality of a participant's input; *users' feedback*, where reports are done by other entities; *community trust*, where a trust server can query its neighbors about their trust on a specific participant; and *history stored*, for example in databases;

- *Information Mapping to Trust Score*: in this phase, the information that is continuously gathered needs to be transformed into a reputation score. A participant can start with a default score, and based on the received feedback it updates the score. It may be done in a centralized or distributed manner. The default score for newcomers as well as the updates on the scores, need to take into account the attacks that will be discussed in Section 2.3;

- *Dissemination*: in this phase the participant score is spread across other entities. Entities can request the updated score of users when needed or the updated score can be automatically sent to entities at a fixed rate or according to some threshold;

- *Decision Making*: in this final phase, a decision based on the trust score assigned to each participant needs to be made. The specific decision depends on the use case but can be used for example to decide if a contribution is accepted or not.

Hoffman et al. [HZNR09], identifies three main phases when building a reputation system: *Formulation*, *Calculation* and *Dissemination*. In the *Formulation* phase, a mathematical scheme is developed to transform collected information into scores. In the *Calculation* phase, the *Formulation* phase is implemented within the constraints of a particular reputation system. These two phases represent a subdivision of the phase *Information Mapping to Trust Score*. The *Dissemination* phase is equivalent to the phase with the same name presented by Mousa et al. [MMH+15].

Hoffman et al. [HZNR09] also discusses important aspects related with reputation systems, which are summarized in Table 2.1. The information source can be: *manual*, which comes

Table 2.1: Important notions about reputation systems.

Notion	**Type**
Information Source	Manual Sources, Automatic Sources
Information Type	Positive, Negative, Both
Reputation Metric	Binary, Discrete, Continuous
Reputation Type	Symmetric Reputation, Asymmetric Reputation
Distribution	Centralized, Distributed

from human feedback or *automatic*, which are obtained automatically. The type of information provided can be positive, negative or both. The type of score that is given can be a binary, discrete or continuous value. The type of reputation, can be: *symmetric*, where each node has a global reputation, or *asymmetric*, where each node has an individual view of each node of the system. The location of reputation can be in a *centralized* entity or *distributed* between entities.

2.1.2 Score Calculation

According to Garcin et al. [GFJ09], a reputation can be calculated based on n ratings r, given by entities through specific aggregating methods such as:

- Arithmetic mean: $Mean = \frac{1}{n}\sum_{i=1}^{n} r_i$;
- Weighted mean: $WeightedMean = \frac{\sum_{i=1}^{n} w(i)r_i}{\sum_{i=1}^{n} .w(i)}$, where w(i) is the weight function;
- Median: Smallest value, r_d such that half the values are $\geq r_d$ and half the values are $\leq r_d$;
- Mode: Smallest value r_0 which occurs most frequently as a rating;

Abdel [AH16] and Mousa et al. [MMH$^+$15] present some methods used in reputation models. We will discuss in detail the two methods we deemed more relevant for building reputation scores: the Bayesian model [MMH$^+$15][AH16][JI02] and the Gompertz function [MMH$^+$15].

Bayesian Model

Jøsang and Ismail [JI02] presented a system called *beta reputation system*, 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:

$$E(\alpha, \beta) = \frac{\alpha}{(\alpha + \beta)} \tag{2.1}$$

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 2.1. and is expressed as:

$$ReputationScore(i) = E(r_i, s_i) = \frac{r_i + 1}{r_i + s_i + 2} \tag{2.2}$$

where the reputation is in the range of [0,1] and the value 0.5 represents a neutral rating. A range such as [-1, +1] can be more intuitive:

$$ReputationScore(i) = (E(r_i, s_i) - 0.5) * 2 = \frac{r_i - s_i}{r_i + s_i + 2} \tag{2.3}$$

where the value 2 makes the range between [-1, 1]. If it was 200 the range would be [-100, 100].

The reputation score reflects the type of behavior of the considered participant. For example, if $r = 19$ and $s = 4$, it has a reputation score of $\frac{19-4}{19+4+2} = 15/25 = 0.6$ which is considered good. If $\alpha = 1$ and $\beta = 9$, it has a reputation score of $\frac{1-9}{1+9+2} = -8/12 = -0.67$ which is considered bad.

Old feedback may not always be 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:

$$r = \sum_{i=1}^{n} r_i * \lambda^{n-i} \quad \text{and} \quad s = \sum_{i=1}^{n} s_i * \lambda^{n-i} \tag{2.4}$$

where $0 \leq \lambda \leq 1$. If $\lambda = 1$, there is no forgetting factor, if $\lambda = 0$, only the last feedback matters.

The problem with Equation 2.4 is that the feedback needs to be kept forever and recalculated from the beginning every time. This can be avoided by using a recursive algorithm for computing the (r,s) parameters:

$$r = r_{i-1} * \lambda + r_i \quad \text{and} \quad s = s_{i-1} * \lambda + s_i \tag{2.5}$$

This statistical distribution could be used as the score calculation technique, by having entities report good or bad behavior of a user to our reputation system, which will allow for the calculation of r and s. Adding a forgetting weight can be used to increase the importance of recent behavior.

Gompertz function

Mousa et al. [MMH+15] presents a way of calculating a reputation score through a sigmoid function, which is defined according to:

$$ReputationScore = a * e^{-b*e^{-c*t}} \tag{2.6}$$

where a is the upper asymptote, b controls the displacement of the output along the x axis, e is the Euler number, c adjusts the growth rate of the function and t can represent a evaluator parameter.

The reputation score reflects the type of behavior of the considered participant. For example, if $a = 1$, $b = 10$, $c = 1.5$, and we have $t = 1$, then the reputation score will be 0.11, and the participant has bad behavior and if we have $t = 3$ it will have 0.89, which is good behavior.

The evaluator parameter can be calculated through:

$$t = \sum_{k'=1}^{k} \lambda^{k-k'} * t_{i-1} \tag{2.7}$$

where k is the total number of tasks that the participant joined, and the summation is used to aggregate historical information. The impact of old data is reduced through $\lambda^{(k-k')}$ with $0 \leq \lambda \leq 1$. The λ can be used as a reward/penalty mechanism by replacing λ with two different values λ_p and λ_s. λ_p is used with participants who have been witnessed to misbehave, such that $\lambda_p > \lambda_s$. This makes a reputation of a participant with λ_s increase more rapidly than one with λ_p.

Incorporating both *aging* and *reward/penalty* mechanisms gives the Gompertz function good capabilities to reflect the immediate behavior of participants. This function can be used for score calculation in our reputation system by having k as the total number of reports received of good and bad behavior of a user, and changing the λ based on the action: If the user misbehaved than the λ must be high, otherwise λ must be low.

2.2 Reputation Attack 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. It depends on the specific application and social setting of the reputation system.

2.2.1 Reputation Attacks

Koutrouli and Tsalgatidou [KT12], divide *reputation attacks* into three main categories: Unfair Recommendations, Inconsistent Behavior and Identity Management attacks.

Unfair Recommendations

Unfair recommendations are recommendations that do not share honest feedback, which are subdivided into:

- *Individual*: Attacks done by individual users, which can be:
  - *Bad mouth*: Happens when a malicious user 'bad-mouth' others in order to unfairly reduce their reputation;
  - *Random opinion*: Selfish users send random opinions because it reduces costs in time and resources;
  - *Unfair praise*: Malicious users give higher recommendations because of fear of reprisals or reciprocity expectation.
- *Collusion*: Group(s) of malicious users that try to destabilize the system, which can be:
  - *Collusive reduction of recommendation*: Badmouthing a subset of users, creating conflicting opinions about the behavior of the victims and the recommendations of the honest recommenders;
  - *Collusive deceit*: Inside a group of malicious users, the users can provide positive feedback to each other, to reduce the effects of their malicious behavior on their reputation.

Inconsistent Behavior

Deliberate use of inconsistent behavior to create a false impression of a user reputation, which allows them to misbehave while maintaining a good reputation. It has the following categories:

- *Traitor*: Users that behave properly for a length of time, in order to establish a strong positive reputation, and be highly trusted, and then deceive others;
- *Discriminator*: Behave properly with most of entities and misbehave towards one of them or a small subset of them.

Identity Management Attacks

These attacks exploit the type of identity management used and the anonymity level provided by the system, and they can be divided into two groups:

- *Registration policy attacks*: Each user, must be registered to the reputation system with a form of identity. The following attacks may occur when pseudonymity is being employed and there is no associated cost with the production of new identities:
  - *Sybil Attack*: A malicious user that can create multiple identities, which allows it to provide large amounts of false feedback about other users without repercussion;

  - *Whitewashing/Pseudospoofing*: A malicious user is able to discard its identity and enter the system with a new one, escaping its bad reputations. This attack is only valuable, if the reputation assigned to a new user is greater that it's own reputation.

- *Authentication policy attacks*: Occurs when there is no way to ensure that users are who they claim they are and if they have the permissions to perform functions in the system. The attacks are:

  - *Impersonation*: Malicious user that portrays itself as another user;
  - *Man-in-the-middle attack (MITMA)*: Malicious user is able to tamper with messages, by listening, changing or dropping messages.

2.2.2 Defense Techniques

Koutrouli and Tsalgatidou [KT12] present defenses techniques that can be deployed to reduce or prevent the reputation attacks:

- *Similarity-based filtering techniques*: Exclude out recommenders, whose feedback has low similarity with the others, which can identify unreliable recommenders. Can reduce the effect of unfair recommendations done by individuals and discriminators;

- *Estimating reputation of the recommender*: Use the reputation of the recommender to decide if the recommendation is trustworthy. Can reduce the effect of unfair recommendations done by individuals;

- *Incentive-based*: Incentivize users by rewarding fair recommendations, and punishing unfair recommendations. Can reduce the effect of unfair recommendations done by individuals;

- *Collaborative filtering techniques*: Clustering techniques that can be used to identify and filter out a group of misbehaving users working together. Can reduce the effect of collusive reduction of recommendation;

- *Incorporating time in reputation estimation*: Give more weight to recent recommendations, making recent behavior matter more than old behavior. Can reduce the effect of inconsistent behavior done by a traitor;

- *Context-based*: Context can be used to assign weights to recommendations. For example use the importance of a transaction to define the amount of increase or decrease in the reputation score. Can reduce the effect of inconsistent behavior done by a Traitor;

- *Penalize oscillatory behavior*: Penalizes sudden changes in behavior by making negative ratings have more weight than positive ones in the score. Can reduce the effect of inconsistent behavior done by a Traitor;

- *Controlled anonymity*: Conceal the identities between users, while only the administrator knows the true identities. Can reduce the effect of personal unfair recommendations, collusive reduction of recommendations and discriminators;

- *Cryptographic mechanisms and unique digital identities*: Use digital signatures which is a mathematical technique used to validate the authenticity and integrity of messages, and use cryptographic mechanisms to ensure confidentiality and freshness of the messages. Reduces the effect of collusive reduction of recommendation, impersonation and MITMA;

- *Difficulty to change/create identities*: Methods to discourage Collusive deceit, Sybil attacks and Whitewashing such as:

  - Restrict identity creation using proof of work, having entry fees, or having a certificate authority that can ensure that each user can only create one identity;
  - Do not allow a reputation of a misbehaving user to fall behind of a newcomer;
  - Assigning a low reputation value to newcomers;
  - Make in every recommendation an amount of the reputation of the recommender to be transferred to the recommended entity. Users with multiple identities can no longer artificially increase the reputation of one of their identities as the total reputation of a user is shared between the identities it possesses. One issue with this approach is that it may be unfair for a recommender to lose an amount of its trustworthiness when recommending another entity.

The defenses presented often clash with each other, as making use of one defense may increase the possibility of another attack. For that reason, Koutrouli and Tsalgatidou [KT12] also analyse the trade-offs between attacks and its defenses:

- *Negative feedback sensitivity versus collusive badmouthing attacks*: When the reputation of a user is significantly decreased with negative recommendations, the reputation system is vulnerable to bad mouthing, and collusive attacks. A trade-off mechanism can be used to check the recommender's credibility, by using filtering techniques to filter out dishonest recommenders or using incentive mechanisms for honest recommendations;

- *Encouraging newcomers vs Preventing Whitewashing*: Initial trustworthiness value assigned to a newcomer should be high enough to encourage new users, but at the same time discourage malicious users to get new identities to escape bad reputation;

- *Resiliency to oscillatory behavior vs Helping reputation repair*: Taking into account the users history and the changes in its behavior mitigates oscillatory behavior but it has as a result that a misbehaving user cannot easily restore its reputation. This may be unfair for honest users which may unintentionally misbehave sometimes.

2.3 Privacy-Preservation Techniques

According to Koutrouli and Tsalgatidou [KT12], the privacy of a user has to do with their ability of being anonymous and not let other users record its transactions and recommendations. 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 it also means there is an higher level of accountability, increasing the credibility of the reputation estimation.

According to Christina et al. [CRKH11], the use of pseudonyms is a common mechanism used to protect the anonymity and privacy of the participants. Instead of transmitting a real identifier of the user, all interaction with the application is performed under an alias. Seigneur and Jensen [SJ04] also consider that 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.

Calado and Pardal [CP18] present This4That, a reward system based on points where you can create tasks for other users to do in exchange for points. A user starts with one thousand points, spends one hundred points every time it creates a task and gains fifty points for answering a task. To assign points to the users, the system uses pseudonyms as a privacy mechanism which allows for the replacement of real names to identifiers with non-related values. Pseudonyms alone still allow correlation between the data and the user by the IP address that comes with the request. Mix Networks are formed by a chain of proxy servers between participants and can be used to prevent the correlation with the IP address, where they make it extremely hard to discover who was the original sender. A rotation of MAC and IP addresses can also decrease the possibility of associating a MAC address to a device. Group signatures can also be used in order to provide a form of authentication through signatures that are made by a group allowing the system to authenticate a member of the group without needing to know which specific member

did the signature.

Ma et al. [MPQ+19] 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[1]. 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. The system starts with a trustor requesting sensing data to an edge node, which will request multiple targets to answer. The targets are responsible for collecting the data and then send back the data encrypted to the edge node. The edge node makes calculations without decrypting the data, which will allow sending back the sensing data without losing privacy as well as update the targets reputation in the blockchain.

Differential privacy is a strategy based on quantifying and limiting an attacker's greatest possible information gain to lessen the risk of privacy breach. It consists on evaluating and sharing data while maintaining individual privacy protection in accordance with existing policies or legal criteria for disclosure limitation or de-identification [WAB+18]. This approach ensures that anyone looking at a collection of differential private analyses will draw the same conclusion about any private information of an individual. The private information is limited and quantified by a privacy loss parameter, which quantifies the maximum possible information gain by the attacker and determines how much noise needs to be introduced during the computation. Costa [dC20] further elaborates on this concept and applies it to geographic location data through *Geo-Indistinguishability*, which secures the location of users reporting their location and assures that any two locations within a specific radius are statistically indistinguishable.

Marti and Garcia-Molina [MGM03] analyse the trade-offs between anonymity and reputation by comparing two distinct types of identity infrastructures for reputation systems: One type uses a central server to tie real world identities of users to pseudo-identities, and the second is completely decentralized, with each user creating and managing its own identity. The first approach is resistant to attacks related to identity management, but the privacy of users may be infringed because they must provide information at a level that they may not feel acceptable. The second approach, is able to protect the privacy and anonymity of users, but is vulnerable to identity management attacks. The authors show that even simple reputation systems can work well in either of the identity infrastructures and that the best approach depends on the specific concerns of the system that is being developed.

[1] https://www.analyticssteps.com/blogs/what-consortium-blockchain

2.4 Location Certification Systems

Location certification systems provide ways to attest and verify that a user is where he is saying he is. These systems are mainly categorized into two groups: *centralized*, where a trusted wireless infrastructure is employed to generate location proofs to mobile users; and *decentralized*, where the users act as witnesses and generate location proofs for each other. They usually have three types of entities: *Prover*, who needs to prove its location and makes a location claim; *Witness*, that may endorse location claims; *Verifier*, who verifies evidence and issues a location certificate.

Nosoushi et al. [NSY+20] presents a distributed location proof scheme called PASPORT (Privacy-Aware and Secure Proof Of pRoximiTy), 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 it is 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 witness.

LINK [TCB10] is another location authentication protocol, that follows a distributed location proof scheme. When a prover wants its location to be certified, it sends a location claim to a certification authority, that acts as the verifier. After that, it requests endorsements of its location from other users (witnesses). These witnesses, will then send a reply to the verifier, which will then decide if the claim is accepted or not based on: *spatio-temporal correlation*, where it rejects a claim if it is not possible to physically move from the prover's previous claimed location to the current location in the time frame between them; and a *reputation score* that is maintained for each user, and is used to discard witnesses' claims, if their score is not above a threshold. The witnesses that remain are divided into two sets, one that agree with the claim versus those who do not agree, if the difference of the sum of scores between the sets is high, then the decision of whether the claim is accepted or not is done based on the set that won. Otherwise it will further analyse the witnesses, by looking into its score trends and location, to remove malicious witnesses. The new sets can then allow for making a decision on the claim. If the difference is still not high enough then the claim is ignored and the prover will need to restart the process. The reputation score of a prover additively increases when its claim is accepted, and multiplicative decreases when it is rejected to discourage bad behavior, making bad behavior much more influential to the score than good behavior.

SureThing project developed a framework for location certification that can be used when developing different applications and with internet of things devices [IFMM10]. SureThing started

with the work of Ferreira and and Pardal [FP18] where they developed a location certification system based on witnesses, for a shopping mall advertising application. This system provided three techniques for location certification: *geographic location*, that verifies if the witness and the prover are close to each other and in an acceptable area; *wi-fi fingerprinting*, which verifies if the prover and witness share a minimum number of common access points; *bluetooth beacons*, that verifies if the beacon values provided by the prover and witness are the same and the beacon exists. It also presented two mechanisms against collusion: *witness decay*, where proofs of the same witness gradually lose their value and *witness redundancy*, where proofs have to be gathered from multiple witnesses.

2.4.1 SureThing Applications

The SureThing [FP18] motivated the development of a framework that provides data formats and utility libraries as well as the development of application domains that apply location certification. We are going to present these domains as well as discuss how a reputation system could be incorporated into them.

STOP: Vehicle Inspection

Santos et al. [SCRP20] presented STOP, a road transportation vehicle inspection support system with tamper-proof records to prevent location spoofing attacks. STOP main components are: *Central Ledger*, which is a central server that receives transportation and inspection records; *Transport mobile applications*, which runs in the vehicles transporting the reported goods. *Inspect mobile application*, used by the inspector at an inspection location. Each transporter periodically reports its location to the central ledger as a chain of locations. To perform an inspection, the inspector chooses a range where to find a car nearby and notifies the respective transporter. When the transporter arrives at the inspection site, it sends a proof request over Bluetooth using a pseudonym and a nonce given by the central ledger. Finally, when the inspection is over it sends a proof to the transporter as well as the central ledger.

A reputation system can be incorporated by making the central ledger send participant reports based on the records it receives to a reputation entity that keeps updating the participants reputation score. The reputation could be used, as a way to identify potentially malicious participants, and give them a higher priority to choose to inspect these transporters first, as they could have something to hide.

CROSS: Tourist Itinerary Verification

Maia et al. [MCP20] presented CROSS, a system that conducts location verification using Android smartphones. It takes a centralized approach, to eliminate the need to be surrounded by sufficient users of the system. The main components of CROSS are *CROSS-Client application*, *CROSS-Server*, *Wi-Fi access point* and *Kiosk*. The system flow starts when the tourist downloads the app. During its use, the application logs Wi-Fi scans during visits to locations. At the end of the trip, the logging stops, and the application sends the collected data to the server. If the Wi-Fi evidence is accepted, then rewards will be given.

There are three strategies that can be employed during a trip for proving that the user is actually attending the locations on the itinerary:

- Capturing which access points are available in specific locations during the itinerary;
- Having an access point in specific locations in the itinerary that dynamically changes name during the day, which allows for a capture of the access points only during a specific period of time, which allows stronger evidence;
- Interacting with the Kiosk by scanning a pair of QR codes. This strategy was implemented in later work.

A new strategy was also added by Grade et al. [GPE22] in a new version of CROSS, that allows for certification based on witnesses, where a prover requests nearby witnesses to endorse its location in specific stops in the itinerary, which provides a decentralized approach to the system.

A reputation system can be incorporated by making the server send participant reports, to a reputation entity based on the current rejection and acceptance of the collected data. The user reputation, could then be used by the server, as an additional parameter to decide the level of trust the user has on the compliance with the tourism itinerary, working along with the strategy used to help decide when rewards should be given. We could also use the accumulated score of the witnesses to decide whether to accept the itinerary or not.

SurePresence: Wearable and Kiosk Devices

Francisco et. al. [Fra21] presented SurePresence, an application that allows a user to prove his location when interacting with a kiosk through the use of location proofs. It has three components: The *client* which represents the prover that makes location claims; The *kiosk*, which represents a witness of the system that endorses the prover claims; The *server*, that acts

as the verifier and is responsible for storing verified location certificates. This client-server model with a single witness presents three location proof techniques: Kiosk-Only technique, which allows a user to prove his location only by using his citizen card and the kiosk; Kiosk-Wearable technique, which generates a location certificate through the interaction of a wearable device with the kiosk; Kiosk-Smartphone technique, which generates a location certificate through the interaction of a smartphone device with the kiosk.

A reputation system can be incorporated by making the server sends participants reports, to a reputation entity. The user reputation could be used as an additional factor to decide if the location proof provided by the kiosk is acceptable.

2.5 Location Certificate transparency (LCT)

Location certificates, allow for any party to independently verify a location claim and digital signatures. However, by themselves, they do not provide trusted storage and verification at a later date. Carvalho et al. [CERP21] proposed a location certificate transparency solution that provides accountability to all entities, by having a tamper-proof ledger based on Merkle trees that provide APIs for storage, retrieval and verification of Location Certificates.

This work is intended to complement CROSS [MCP20], by requiring the verifier to submit the location certificates to a tamper-proof append-only verifiable public log, allowing for the location certificates to be rechecked later when needed. It has three entities:

- *Log Server*: An append-only public log server (ledger) that stores logs of the location certificates so that everyone can audit them when needed;

- *Monitor*: Service that continuously scans for suspicious location certificates and keeps copies of the entire log, to ensure the versions of the log are consistent. It can also be used as a search engine to find location certificates;

- *Auditor*: Service that verifies if the log is consistent and behaves correctly. It sends two Merkle tree hashes (one for an older version of the Merkle tree and another for a newer version, which are given by the monitor) to the log server, which calculates a consistency proof that verifies if two versions of a log are consistent, i.e., if the later version includes everything in the earlier version, in the same order, and all new entries come after the entries in the older version. The result is sent back to the auditor, for verification. It can also determine whether or not a specific certificate is included in the log by asking for a Merkle audit proof, which is a list of missing nodes needed to compute the nodes leading

from a leaf to a root of the three. If the root computed from the audit path and the true root are the same, then the audit path is proof that the leaf exists in the tree.

2.6 Summary

To build a reputation system we need to collect and aggregate behavior data, transform the behavior data into a score, disseminate the score and make decisions based on the score. There are multiple methods that can be used for transforming the behavior data into a score, the most relevant was the Bayesian model. The model calculates the reputation score of a user using Equation 2.2, where r and s represents the cumulative amount of positive and negative ratings respectively.

Reputation systems are susceptible to reputation attacks. These attacks and the defense techniques are respectively summarized in Figure 2.1 and in Table 2.2.

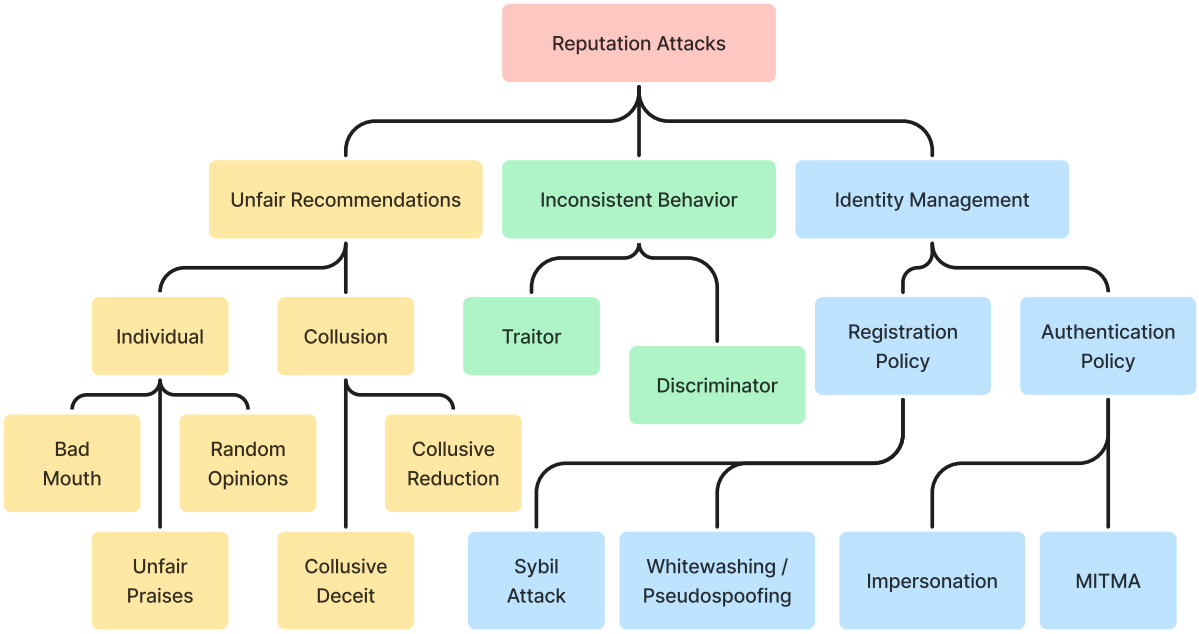

Figure 2.1: Summary of reputation attacks.

Privacy and anonymity are concerns in reputation systems, but it is mandatory to have a way of identifying a user to save its reputation and make changes to it. There are trade-offs, the more privacy and anonymity it has, more difficult it is to build a reputation to an identity. The use of pseudonyms is a promising solution, that may provide a good level of privacy while still permitting the assignment of reputation to users.

The SureThing framework has been implemented in different application domains such as STOP [SCRP20], CROSS [MCP20] and SurePresence [Fra21], which could all be enhanced by a reputation system. A reputation system could be introduced by having entities inside each

Table 2.2: Summary of attacks and defense mechanisms.

Attacks	**Defenses**
Individual unfair recommendations	Similarity-based filtering techniques Estimating reputation of the recommender Incentive-based Controlled anonymity
Collusive reduction of recommendation	Collaborative filtering techniques Controlled anonymity Cryptographic mechanisms and unique digital identities
Collusive deceit	Difficulty to change/create identities
Traitors	Incorporating time in reputation estimation Context Penalize oscillatory behavior
Discriminators	Similarity-based filtering techniques Controlled anonymity
Sybil Attack and Whitewashing	Difficulty to change/create identities
Impersonation MITMA	Cryptographic mechanisms and unique digital identities

domain, that send reports of users behaviors to a reputation entity, that is able to generate and update a reputation score for each user. The reputation value could then be used to support a system when making decisions such as when deciding if a location claim is valid or not by using the score of witnesses as the level of trust we can give to witness endorsements or even to the specific user that is submitting the claim.

Chapter 3

SureRepute

We developed SureRepute, a reputation system that allows application domains of the SureThing framework to maintain a reputation score for each user that reflects their behavior. This is done by allowing separate application domains to submit behavior reports as well as requesting user scores from SureRepute. In Section 3.1 we detail all aspects of the design of the SureRepute system. In Section 3.2 we show the implementation of our solution. Finally in Section 3.3 we detail a specific use case where SureRepute was integrated, which was CROSS [MCP20] for smart tourism.

3.1 Design

In this Section, we explain all aspects regarding the design of SureRepute. We first discuss the attack model in Section 3.1.1, then the assumptions and requirements identified in Section 3.1.2. We present an overview of the architecture of the system in Section 3.1.3. We analyse in detail the privacy protections in Section 3.1.4 and finally we explain the score calculation technique for the reputation score in Section 3.1.5

3.1.1 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 2.2 (Reputation attacks).

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 [MCP20], the CROSS-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.

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 the following (Attack:Defense):

- *Traitor*: Penalize oscillatory behavior by having a malicious action affect much more the score than a good one. Incorporate time when updating a user score, making recent behavior matter more then old behavior;
- *Discriminator*: Controlled anonymity through the use of pseudonyms, i.e., when a user uses different domains it is given the same pseudonym, which leads to a global reputation that is shared among different domains, making bad behavior in a specific domain affect the reputation score in general;
- *Sybil Attack and Whitewashing*: Reduce the motivation for creating or changing identities by giving the smallest score possible to newcomers and have a slow build-up of reputation;
- *Impersonation and Man-in-the-middle-attack*: All messages will ensure Confidentiality, Integrity, Authenticity and Freshness by using cryptographic mechanisms and public-private keys that ensure unique digital identities.

3.1.2 Assumptions and Requirements

The system operators of SureRepute are the organizations that are providing the SureThing applications and that would benefit from having reputation scores of its users. Next, we define the assumptions and the requirements for the reputation system.

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 communicate with the Certificate Authority (CA).

Functional 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 ensure privacy to the users.

Non-Functional Requirements:

- **Adaptability** - The system needs to be easily integrated with new domains;
- **Extensibility** - The system must be flexible for future modifications and extensions;
- **Security** - The system must be able to resist inconsistent behavior and identity management attacks;
- **Privacy** - The system must protect sensitive and private information about the users, such as their behavior.

3.1.3 General Architecture

The solution was implemented using a client-server model that can be incorporated into each of the SureThing application domains as represented in Figure 3.1, 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 for 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.

Our architecture was designed based on the usage of the SureRepute by the SureThing application domains, but the entities that use SureRepute through the SureRepute-Client can be from different systems. We allow the SureRepute-Server to be deployed by a different organization than the one that is using it, and so to ensure anonymization to the users we use pseudonyms.

The entities that constitute the reputation system are:

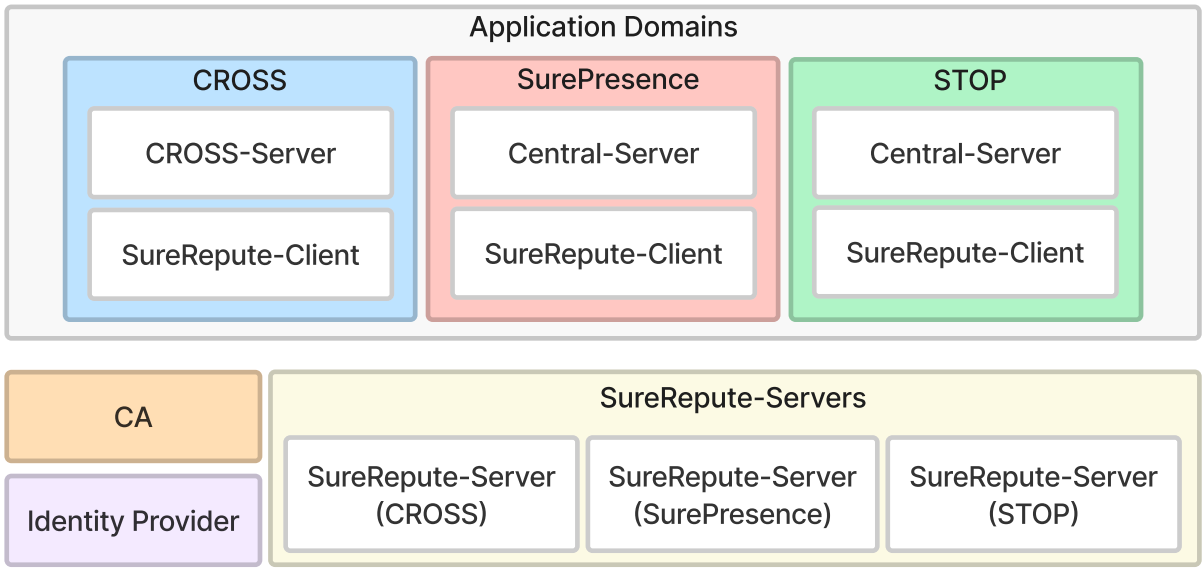

Figure 3.1: SureRepute integration with SureThing Framework.

- *SureRepute-Client*: This entity serves as a client that handles the identity-pseudonym translation and interacts with the SureRepute Server. It provides two actions to the domain servers: report user behavior and get user score. The report submitted must contains a rating value that reflects how the user behaved;

- *SureRepute-Server*: This entity is responsible for handling user scores and keeping the accumulated values of good and bad behavior, which makes a reputation score, for each pseudonym of its users. The server handles requests made from the client. A SureRepute-Server talks with other SureRepute-Servers whenever a new pseudonym appears, as we need to check if they are already handling them. This step allows the system to maintain a shared reputation for the same pseudonym;

- *Identity Provider*: This entity is in charge of doing the identity-pseudonym translations. It registers identities by storing a piece of identity of the user and generating a pseudonym that becomes associated to it. When SureRepute-Client requests the pseudonym of the user, the identity provider will send back the pseudonym encrypted using the public key of the respective SureRepute-Server, in order to make the pseudonym only known to the SureRepute-Server;

- *Certificate Authority (CA)*: This entity is trusted by all other entities and it is responsible for validating the ownership of their public keys.

3.1.4 Privacy Protection

To ensure confidentiality, all communication between SureRepute entities must use TLS[1] with client authentication enabled (except with CA), which demands that both entities are trusted.

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 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. Table 3.1. presents which information related to the pseudonyms each entity has access to and shows how the user's anonymity is ensured in SureRepute. To evaluate what an attacker is able to access, in Section 4.1.3 we will create attack scenarios where a user can have access to the databases of the different entities.

Table 3.1: Entities access rights. X=Has access and Nothing=No access.

	Domain Entity	**Identity Provider**	**SureRepute-Server**
Real Identities	X	X	
Encrypted Pseudonyms	X	X	X
Pseudonyms		X	X
Behavior Details			X
Scores	X		X

3.1.5 Score Calculation Technique

Our reputation score calculation will follow mainly the binomial Bayesian model presented in Section 2.1.2, with some modifications. We will use Equation 2.2 to calculate the reputation score, and use a modified version of Equation 2.5 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:

- *Accidentally Malicious*: where the user behaved maliciously, but could be due to factors

[1] https://www.cloudflare.com/learning/ssl/transport-layer-security-tls/

outside of the user's control. In this case s and r will be updated with:

$$s = s_{i-1} * \lambda_s + 0.5 \quad \text{and} \quad r = r_{i-1} * \lambda_r \tag{3.1}$$

- *Intentionally Malicious*: where the user purposely behaved maliciously. In this case s and r will be updated with:

$$s = s_{i-1} * \lambda_s + 1 \quad \text{and} \quad r = r_{i-1} * \lambda_r \tag{3.2}$$

- *Critically Malicious*: where the user purposely behaved maliciously in a critical situation, which depends on the specific domain we are inserting SureRepute. In this case s and r will be updated with:

$$s = s_{i-1} * \lambda_s + 2 \quad \text{and} \quad r = r_{i-1} * \lambda_r \tag{3.3}$$

- *Well Behaved*: where the user behaved as intended. In this case r and s will be updated with:

$$s = s_{i-1} * \lambda_s \quad \text{and} \quad r = r_{i-1} * \lambda_r + 1 \tag{3.4}$$

The value of the forgetting weighs for r and s will be different, making good reports be forgotten much quicker than bad reports, which can make bad behavior affect more the score while also making recent behavior matter more than old behavior. For that reason they provide a crucial defense against the *traitor* attack. This idea was obtained when learning about how to calculate the reputation score using the Gompertz function in Section 2.1.2.

The score of a user will be calculated using Equation 2.2, which is defined as:

$$ReputationScore = \frac{s+1}{(r+s+2)} \tag{3.5}$$

where $0 \leq ReputationScore \leq 1$.

The initial values given to s and r, will not only give the initial score given to the user, but they 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.

The values for λ_s, λ_r and the initial values for r and s are further evaluated in Section 4.1.

3.2 Implementation

In this Section, we will explain how SureRepute was implemented. In Section 3.2.1, we will detail what each entity of SureRepute does. In Section 3.2.2 we will explain how the automatic deployment of SureRepute was done. Finally, in Section 3.3 we will integrate SureRepute with one specific application domain, to test SureRepute in a real use case and show how easy the integration is.

3.2.1 Detailed Architecture

All entities of SureRepute were developed in Java[2], which allows to run in most operating systems and hardware platforms. We also used Apache Maven[3] as the tool to build, test and execute the code from a central piece of information (POM).

SureRepute follows a client-server model, and each server use GlassFish[4], which is an open-source Jakarta platform application server[5], that provides a set of software components, for developing Java applications. Grizzly[6] is also used, as the HTTP server framework, which has been designed to help build scalable and robust servers, and is utilized as the added web server component for the GlassFish application server.

For communication between entities we need to define an Application Program Interface (API). Upon careful thought we decided to use the REpresentational State Transfer (RESTful) software architectural style for web services, which defines a set of constraints restricting the protocol used in Client-Server architectures with regards to the possible requests and responses. As for the concrete implementation of each entity, we used the Jersey 3.0 reference implementation[7] to support the Jakarta RESTful Web Services 4.0 specification[8] (JAX-RS) which eases and standardizes the development of web services according to the REST architectural pattern with additional Java annotations. Additionally, Jersey helps to expose data in a variety of representation media types and abstracts away the low-level details of the client-server communication. For the interface description language and canonical data format we decided to use protocol buffer[9], a data representation language and platform neutral, extensible mechanism for serializing structured data, proposed originally by Google, but currently open-source.

To assure data persistence and avoid data corruption, we decided to use PostgreSQL[10] as

[2]https://www.oracle.com/java/
[3]https://maven.apache.org/
[4]https://javaee.github.io/glassfish/
[5]https://eclipse-ee4j.github.io/jakartaee-firstcup/jakarta-ee002.html
[6]https://javaee.github.io/grizzly/httpserverframework.html
[7]https://eclipse-ee4j.github.io/jersey/
[8]https://jakarta.ee/specifications/restful-ws/4.0/
[9]https://developers.google.com/protocol-buffers
[10]https://www.postgresql.org

the database language for Identity Provider and SureRepute-Server.

We also used OpenAPI[11], which is a language-agnostic interface for RESTful APIs that allows users to discover and understand the capabilities of the services without access to the source code by allowing to create human-readable documentation about both the data and API specification.

Let us now look into each SureRepute entity in more detail.

SureRepute-Client

SureRepute-Client is a library that allows application domains to make remote calls to SureRepute, abstracting away of specific details about communication with the different entities of SureRepute when integrating with new domains. SureRepute-Client makes available two main actions:

- Get Score: Which gets the score from SureRepute of one or more users. The score is a value in the range $[0, 1]$.
- Report Behavior: Which allows application domains to easily submit behavior reports for a given user. The behavior reports can be: Well Behaved Report, Accidentally Malicious Report, Intentionally Malicious Report and Critically Malicious Report as explained in Section 3.1.5.

The diagram of Figure 3.2 represents how SureRepute-Client does each of these actions. The first thing it does is to request to the Identity Provider (1) the associated pseudonym encrypted using the public key of the SureRepute-Server (2). The SureRepute-Client 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 (3). The server will return back the score of the user (4).

SureRepute-Server

Each application domain has its own SureRepute-Server instance, and each SureRepute-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

[11] https://swagger.io/specification/

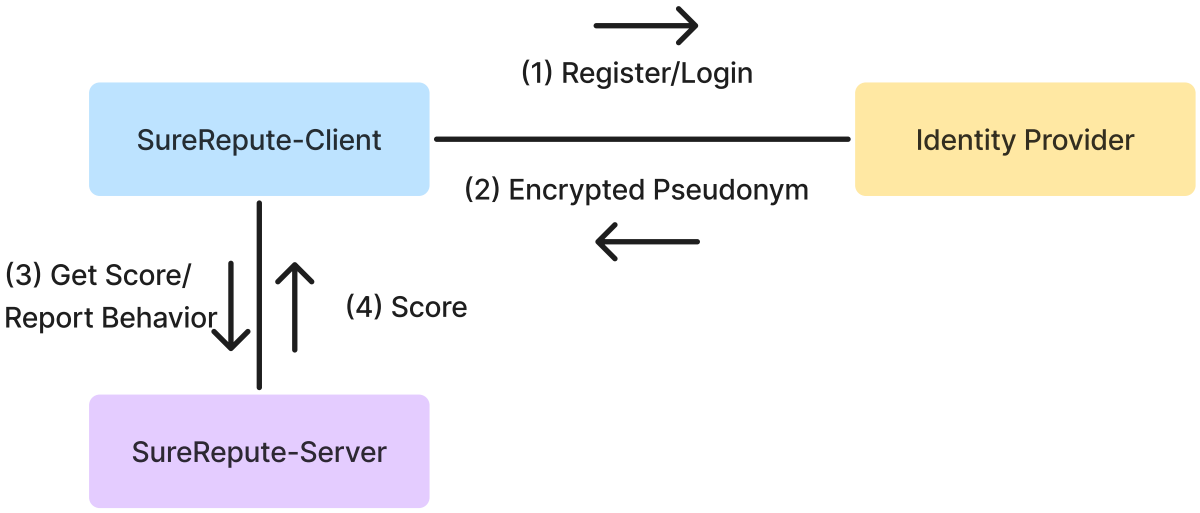

Figure 3.2: Main interaction of SureRepute.

domains, we maintain a shared reputation, i.e a reputation that reflects the behavior of the user in all domains.

In order to maintain a shared reputation for a pseudonym and avoiding inconsistencies, we ensure that only one server is maintaining updates on the reputation of a user, which is called the *leader* of that pseudonym, other servers that also receive requests for that pseudonym are considered as its *followers*.

The diagrams of Figure 3.3 and Figure 3.4 shows respectively how a server becomes a leader or a follower to a pseudonym, which happens in the pseudonym registration phase. When an unknown pseudonym is received in a request (1), the server broadcasts the pseudonym to verify if another server is already the leader for that pseudonym (2), then there are two cases that can happen:

- Case 1: No leader exists yet (3, 4), and so this server becomes the leader for this pseudonym and stores the pseudonym along with the initial Score Details (5).
- Case 2: A leader already exists (3, 4), and so this server becomes a follower for this pseudonym and stores the pseudonym along with the Score Details sent by the leader (5).

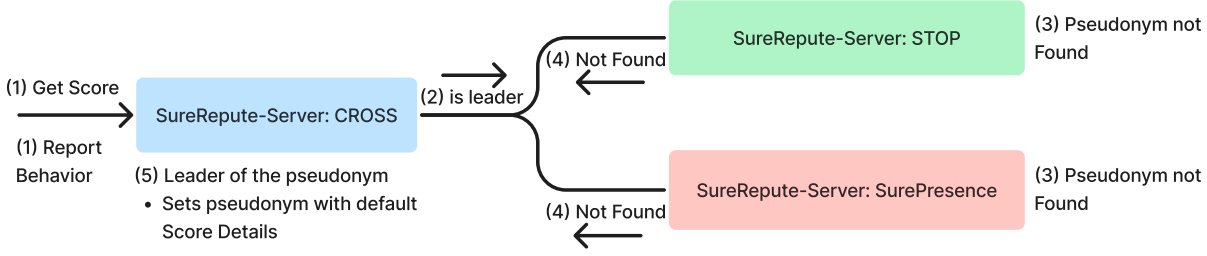

Figure 3.3: Case 1: No leader exists yet for that pseudonym.

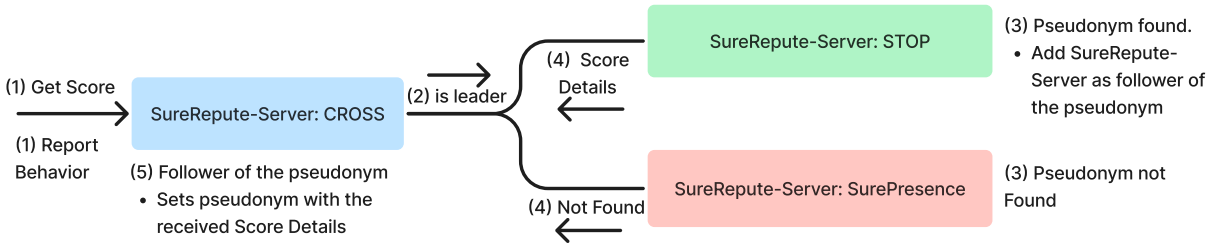

Figure 3.4: Case 2: Leader already exists for that pseudonym - SureRepute-Server: STOP

The diagram in Figure 3.5 and Figure 3.6 shows what happens when a behavior report is received by a follower or a leader of that pseudonym respectively and the pseudonym is already known (1):

- Case 1: If the server is a leader, then it will update the Score Details for that pseudonym (2) and send back to all followers the updated score (3). Each follower will then update the Score Details for that pseudonym (4).
- Case 2: If the server is a follower then it will forward the behavior report to the leader of that pseudonym (2), the leader will then update the Score Details (3) and send back to all followers the updated score (4). Each follower will then update the Score Details for that pseudonym (5).

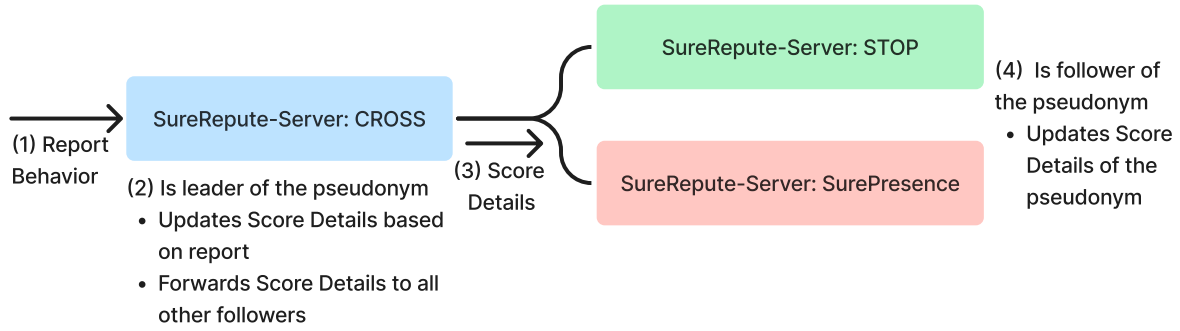

Figure 3.5: Case 1: Report Behavior received by a leader. Leader - SureRepute-Server: CROSS

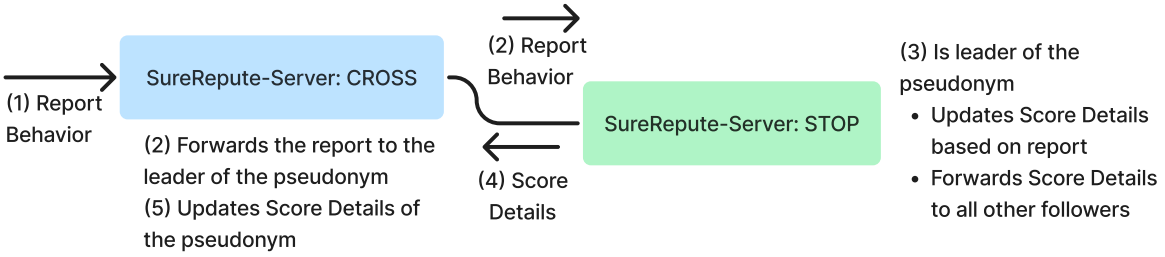

Figure 3.6: Case 2: Report Behavior received by a follower. Leader - SureRepute-Server: STOP.

The diagram in figure 3.7, shows what happens when a client requests the score of a pseudonym that is already known to either the leader or the follower of that pseudonym (1), the server just needs to return the current stored value for that pseudonym (2).

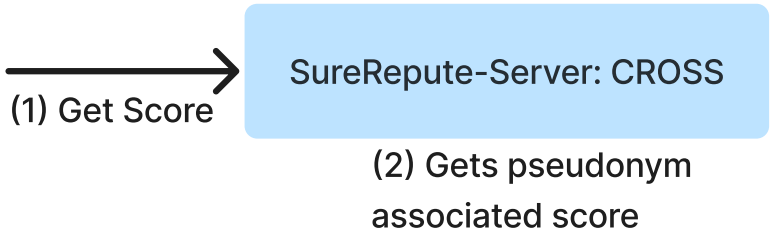

Figure 3.7: Handle of Get Score request received by SureRepute-Server when the pseudonym is already known.

Identity Provider

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.

Certificate Authority

The CA is responsible for validating the identity of all other entities of SureRepute and provide them a public key 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 requires that the identities use specific DNS names of our solution for communication.

3.2.2 Deployment

A key aspect of the project was to also allow SureRepute to be deployed in the cloud, in order to make it very easy to be released if needed. To do that, we decided to deploy the project into Google Cloud[12].

[12]https://cloud.google.com/

To make the deployment we first used Docker to build images for each entity. An image works like a template, which contains the application and all the code dependencies required to run that application. These images can then be pushed to Google Cloud to be used when creating an instance, allowing to deploy the specific entity. We also setup a way of creating and deleting a Kubernetes[13] cluster in Google Cloud, which will contain SureRepute.

To ensure that SureRepute-Servers, the Identity Provider and the CA are isolated from each other we also used logical isolation by using namespaces inside the cluster, where SureRepute-Servers are in 'sure-repute-namespace", IdentityProvider is in 'identity-provider-namespace" and the CA is in 'ca-namespace". This isolation is very important, as in a real scenario the deployment of the SureRepute-Servers and the IdentityProvider needs to be handled by different system operators, responsible for the deployment and management of the system, due to the privacy protections detailed in Section 3.1.4.

A technology that helps you define, install, and upgrade Kubernetes applications is Helm Charts[14]. We used this technology to automatically deploy each entity and its database, as well as ingresses, which is an API object that manages external access to each entity in a cluster and a load balancer. We also used DNS-Kubernetes to automatically create a DNS record with the IP given to that entity by Google Cloud and a DNS name that belongs to the SureRepute domain in GoDaddy[15].

In Figure 3.8, we show a example of a deployment in Google Cloud, with two servers (SureRepute-Server CROSS and SureRepute-Server STOP) an Identity Provider and a CA. The SureRepute-Client is outside of the cluster, as it must be deployed by the system operators of the application domain. This figure is an example of the deployment, the number of servers that are deployed can change, as they are dependent on how many application domains we are working with. All remote calls to entities of the cluster pass through the load balancer, which is responsible of forwarding those calls to the appropriate entity inside the cluster based on the set of rules provided by the ingress. This set of rules allow to forward the request based on the predefined dns names used for the remote call and that are present in the arrows of the figure. These dns names are:

- *client-{domain}.server.surething-surerepute.com*: Used by a SureRepute-Client to interact with its SureRepute-Server. The {domain} in the dns name is replaced by which domain it is interacting, in the case of the diagram it can either be CROSS or the STOP domain;

- *ip.surething-surerepute.com*: Used by a SureRepute-Client to interact with the Identi-

[13]https://kubernetes.io/
[14]https://helm.sh/docs/topics/charts/
[15]https://www.godaddy.com/pt-pt

tyProvider;

- *ca.surething-surerepute.com*: Used by all the entities to interact with the CA;
- *sip-{domain}.server.surething-surerepute.com*: Used by the IdentityProvider to inteact with each of the SureRepute-Servers;
- *ss-{domain}.server.surething-surerepute.com*: Used by SureRepute-Servers, for interactions between each other.

client-cross.server.surething-surerepute.com is used by a CROSS client to interact with SureRepute Server of CROSS.

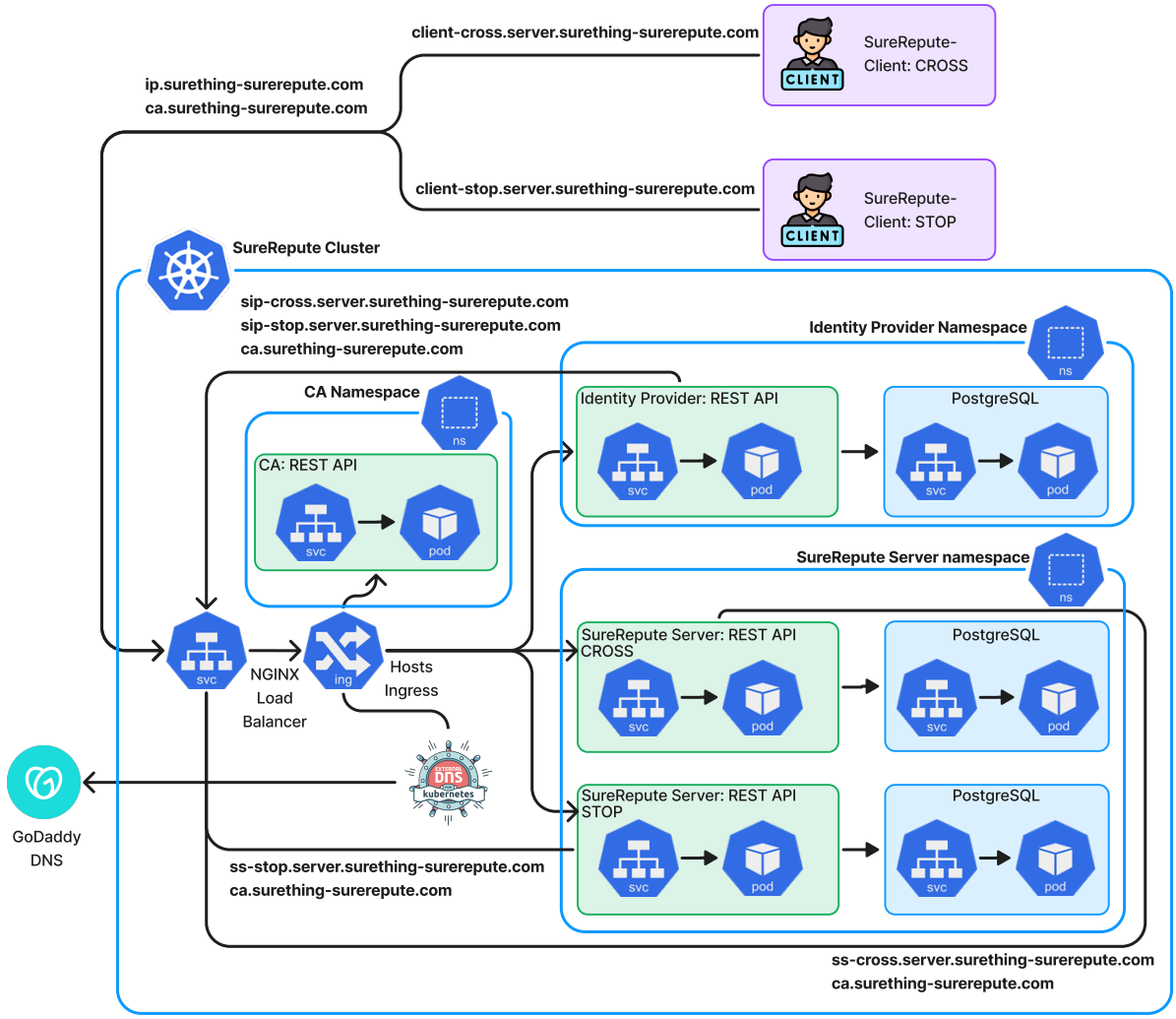

Figure 3.8: Example of deployment of SureRepute in Google Cloud Platform for two domains.

3.3 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[MCP20]. We decided to use CROSS because of the new strategy created by Grade et al. [GPE22] that uses witnesses, and thus, a reputation score of the users could be really helpful, when calculating if a proof should be accepted or not.

3.3.1 CROSS-Server v2.0

Due to maintainability reasons, specifically as a means to address the evolvability and documentation of the code base, the CROSS API Server component has been reimplemented from Golang[16] to Java[17] with Maven[18] as the software project management tool. The interface still follows the RESTful software architectural style for web services. As for the implementation of this server, we used the same technologies as SureRepute, which are GlassFish[19] application server as well as Grizzly[20] for the additional webserver component. For communication we used Jakarta RESTful Web Services 4.0 specification[21].

The redevelopment was used as an opportunity to ensure that the API server is compliant and uniform with other existing SureThing projects with regards to the interface description language and canonical data format. Originally JSON was being used as the data specification, but the SureThing projects use protocol buffers[22], as they are a language-neutral, platform-neutral, extensible mechanism for serializing structured data, it provides binary encoding for typed data across multiple programming languages. We also used OpenAPI[23], which allows to create human-readable documentation about both the data and API specification.

There were some changes made to the actual behavior of CROSS:

- Allow visits to be submitted individually if there is an available Internet connection instead of in the end. This allows users to get more timely feedback about whether the visit was accepted or not;

- Removed the need for completing a trip in a specific order. Now a user can go to a specific

[16]https://go.dev/
[17]https://www.oracle.com/java/
[18]https://maven.apache.org/
[19]https://javaee.github.io/glassfish/
[20]https://javaee.github.io/grizzly/httpserverframework.html
[21]https://jakarta.ee/specifications/restful-ws/4.0/
[22]https://developers.google.com/protocol-buffers
[23]https://swagger.io/specification/%7D

point of interest in the route, as he may be closer to that position. This gives users more flexibility to complete a trip;

- Authentication is now done with JSON Web Token (JWT) sent in the header of every request after login. This is an efficient and widely used authorization method.

A new location proof strategy was added by Grade et al. [GPE22] to further help prove if the traveler is actually attending the location on the itinerary, by using other travellers 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. First, the travel speed is calculated using the Haversine formula[24], using the distance between the coordinates of the current visit and the last validated visit and the time the user had to travel that distance (travelSpeed). Then two configurable constants maxExpectedSpeed and maxAdmissibleSpeed are defined, with the default values off 100 and 200 km/h respectively, but can be adjusted. Based on these values the concrete value assigned to the displacementConfidenceMultiplier is:
  - If travelSpeed $\leq$ maxExpectedSpeed: 100% displacement confidence is assigned
  - If travelSpeed $\geq$ maxAdmissibleSpeed: 0% displacement confidence is assigned
  - Otherwise, the displacement confidence is calculated according to:

$$1 - \left(\frac{travelerSpeed - maxExpectedSpeed}{maxAdmissibleSpeed - maxExpectedSpeed} \right) \tag{3.6}$$

    which essentially means that the closer the travel speed is to maxExpectedSpeed, the higher is the displacement confidence, and the closer to the maxAdmissibleSpeed, the lower is the displacement confidence.

- *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:

$$endorsementWeight(p, w_i) = \frac{Rw_i}{Npw_i + 1} \tag{3.7}$$

[24] https://en.wikipedia.org/wiki/Haversine_formula

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:

$$min\left(\frac{\sum_{i=0}^{n} endorsementWeight(p, w_i)}{endorsementWeightTarget}, 1\right) \tag{3.8}$$

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. [GPE22] work.

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

$$\begin{aligned} min(displacementConfidenceMultiplier \times (wiFiAPsConfidence+ \\ peerEndorsementsConfidence), 1) \end{aligned} \tag{3.9}$$

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.

3.3.2 Location Certificate Transparency v2.0 integration

Location certificate transparency provides accountability to all entities through a tamper-proof ledger based on Merkle trees that provide APIs for storage, retrieval and verification of Location Certificates at a later date. One of the objectives of this work was to complete previous work by Carvalho et al. [CERP21]. LCT was integrated into CROSS, so that the emitted location certificates could be made persistent in a tamper-proof. To complete the work we did:

- A re-factorization of the code and remove some problems found that made LCT not work.
- A refactor on the Data Specification (Contract), due to some inconsistency problems and created an OpenAPI specification in order to help new developers into the project and to have a clear documentation about the LCT architecture described in Section 2.5.

- The integration of LCT into CROSS v2.0.
- A Full specification on how to run each module of the LCT, which is very important to allow for the project to be continued by others.

The integration of the LCT into CROSS is represented in Figure 3.9, where we allowed the possibility of a complete trip to be submitted to the LCT by the CROSS-Server. When a trip is completed, we asynchronously submit a location certificate for each visit inside of that trip to the Log Server (1, 2), which returns back a Signed Location Certificate Timestamp (SLCT) (3, 4), that acts as a promise that this specific certificate will be stored in the log within a fixed interval of time. The CROSS-Server will then wait for each timestamp to pass by (5), and then it will also interact with the Auditor by requesting an audit proof using the SLCT (6). In order for the auditor to verify if the log is behaving correctly and is consistent, it interacts with both the Monitor and Log Server (7), as specified in Carvalho et al. work [CERP21]. The auditor will then return an audit result (8), which is verified by making sure that the certificates for every visit are present.

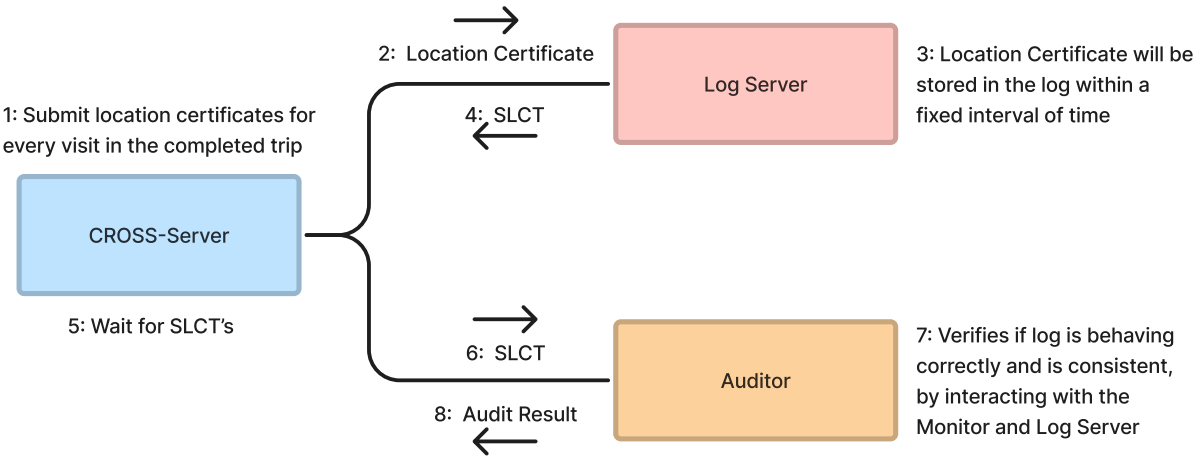

Figure 3.9: CROSS interaction with LCT whenever a complete trip was submitted.

Our initial idea was to also use the Monitor in LCT to submit reports based on suspicious behavior found to SureRepute, and thus provide additional reports for the solution, but as we analysed what the monitor was doing we found that the monitor did not detect problems related with the content of the proof, i.e., related to the endorsements and the location and did not had access to who the user is. Instead currently, the LCT only detects problems related with the consistency of the log. For that reason although we integrated the LCT with CROSS, LCT will not interact with SureRepute for report submission.

3.3.3 SureRepute Integration

To attest how well SureRepute can be integrated with application domains of SureThing, we decided to integrate it with the new version of CROSS. As explained in Section 3.3.1, the goal of CROSS is to visit a set of locations and earn a reward. A visit is considered accepted based on the confidence calculated with Equation 3.9, where the confidence needs to be higher then the confidenceThreshold.

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, as we do not need the result of the report submission in order to return back to the client. Let us now analyse how the scores are used and when reports are submitted.

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 3.7. This endorsement weight is then used to calculate the *peerEndorsementsConfidence*, which is done using the Equation 3.8. 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.

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

$$1 - \frac{1 - confidenceThreshold}{0.5} * travelerReputation \tag{3.10}$$

If the default confidence threshold is 75% then Figure 3.10 shows what is the confidence threshold based on the reputation of the traveller.

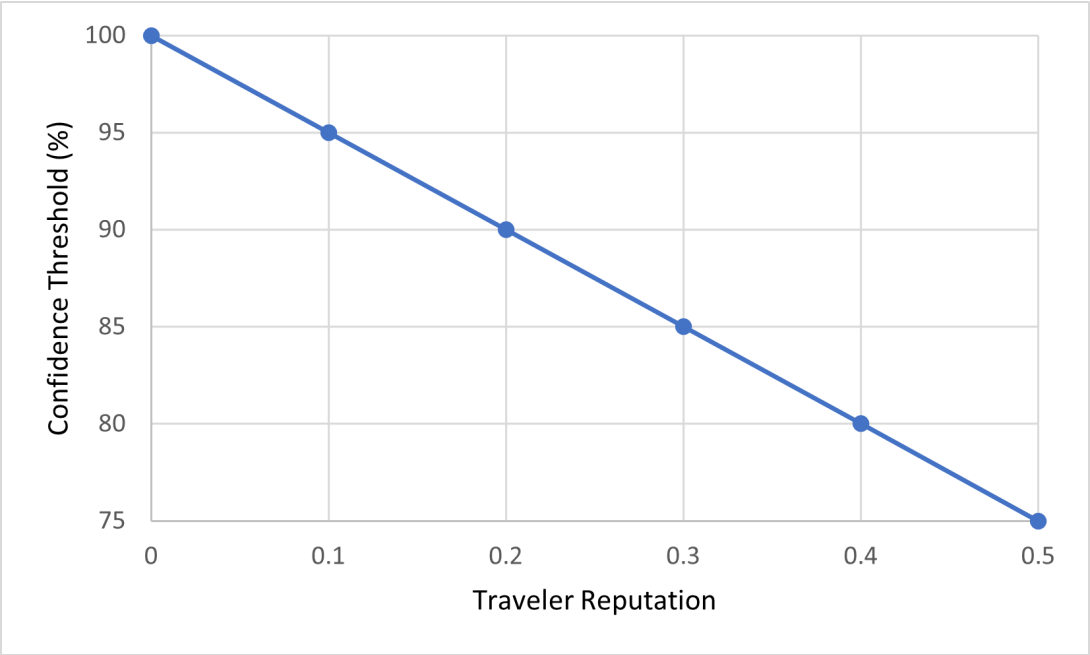

Figure 3.10: Confidence Threshold adjustment according to the traveler reputation.

Report Submission

As explained in Section 3.1.5, there are four types of reports we can submit, and here we adapted the conditions in which each report is submitted to this specific CROSS application:

- Critically Malicious: We submit a report for critically malicious behavior for the traveler when the trip is not coherent, i.e., if the trip does not exist associated with the route given or if a visit that is being submitted does not belong to that trip.
- Intentionally Malicious: We submit a report for intentionally malicious for the traveller whenever we are certain that the traveller acted malicious, i.e, if one of the endorsers is the traveller, if the endorsement is not for that specific visit, if the time of the claim is not correct, or if the claim signature is invalid. As these are always validated in the client side of the application.
- Accidentally Malicious: We submit accidentally malicious behavior for the traveller, when the confidence does not reach the confidence threshold, as we cannot have the complete certainty that the user acted maliciously.
- Well Behaved: We submit reports of good behavior, whenever the confidence threshold is

reached. The reports are sent for the traveller and all for all witnesses that made good endorsements.

3.4 Summary

In this Chapter we presented SureRepute, its design aspects, including the attack model, the requirements and assumptions, its general architecture, a privacy assessment of the whole system and finally the score calculation technique. We then detailed the implementation of the solution, including a detailed explanation of the architecture of all entities as well as its deployment. Finally we also integrated SureRepute in the tourism itinerary use case, which is CROSS, which included a complete re-write of CROSS-Server, the integration with LCT for long-term storage of certificates and finally the integration with SureRepute. Figure 3.11 represents the whole system to support the CROSS application including both LCT and SureRepute.

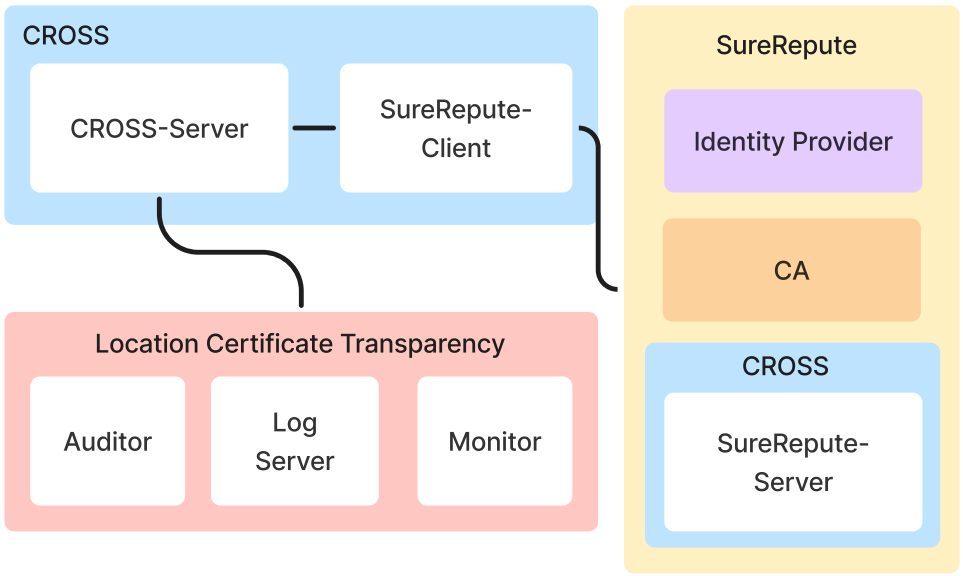

Figure 3.11: CROSS integrated with LCT and SureRepute.

Chapter 4

Evaluation

In this Chapter, we describe how we evaluated our system, and present the obtained results. In Section 4.1, we describe the *qualitative* evaluation, which helped determine whether the requirements have been fulfilled. In Section 4.2 we discuss the *quantitative* evaluation, which will focus primarily on the performance of the system in a specific use case. Finally, in Section 4.3, we present a real scenario where our system integrated with CROSS was used to do a campus tour.

4.1 Qualitative Evaluation

The fulfillment of the functional requirements described in Section 3.1.2 was evaluated in different ways, the evaluation of **R1** "Provides scores that reflect the user's behavior" as well as the traitor attack and Sybil/Whitewashing attack from **R2** "Capable of being resilient to inconsistent behavior and identity management attacks" is presented in Section 4.1.1. **R3** "The same user in different domains maintains a shared reputation that reflects the user general behavior" and the discriminator attack from **R2** are evaluated in Section 4.1.2. The evaluation of **R4** "Use pseudonyms to ensure privacy to the users" as well as the impersonation and man-in-the-middle attacks from **R2** are presented in Section 4.1.3.

To perform the qualitative evaluation we created a demo entity, to perform tests directly with SureRepute. This entity can interact with the entities of SureRepute by using the SureRepute-Client and it allows to perform qualitative tests needed in an easy and re-playable way without using a real domain.

Regarding the non-functional requirements, we ensure the *Adaptability*, and *Extensibility* requirements, by having the SureRepute-Client which works as a library that provides easy interaction with the other SureRepute entities, allowing the application code to abstract away

of specific details about the communication, granting easy integration with new systems while also allowing the system to be flexible for future modifications and extensions. Both *Security* and *Privacy* requirements are ensured when fulfilling the functional requirements R2 and R4.

4.1.1 Score Calculation

In this Section, we want to show that the reputation system is capable of creating scores that reflects the behavior of users 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, namely:

- 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).

The initial score details can be used to choose what score to give to a newcomer when no reports were submitted yet, for example with Details(10, 5) $score = \frac{5+1}{10+5+2} = 0.35$, and it can also give the initial setback before starting to trust the behavior submitted for the user. The forgetting weights serves as a way to make recent behavior matter more than old behavior and can also be used to make malicious behavior to be forgotten much slower than good behavior.

Score Details

To demonstrate how the initial score details serve as the initial setback for the amount of reports that need to be sent before starting to have a reputation that reflects the behavior of the user in the system, we provide some tests where we submitted 5 and 100 reports to the same user, and we also change the number of malicious behavior reports and good behavior reports submitted. For example, when submitting 5 reports we test what the score is when 1, 2, 3, 4 and 5 of the reports submitted are of intentionally malicious behavior. These tests are done using forgetting weights as both 1, i.e., Forgetting(1, 1), to make the behavior of a user to never be forgotten and thus, making order of the report submission to not matter. 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 first four initial score details were chosen to show how these initial score details can represent the initial setback before starting to have a good reputation and the last was chosen to show how the initial score given to a newcomer can be set using these initial values.

The results are presented in Figure 4.1. In the graph there is a black line in 0.5, this is because it is the middle value between scores as their range is between $[0, 1]$, which is the point of when we consider that the users start to have good behavior. Looking carefully at the first graph, i.e., when submitting 5 reports, for Details(1, 0) and Details(5, 0) we can see that if the user submits 5 new 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 even with 5 good behaved reports submitted 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. If we look into the second graph, i.e., where 100 reports were submitted, we can see that with Details(1,0) and Details(10,0) it already provides a score above 0.5, in the case of all reports where of good behavior, because it already surpassed the initial setback for these cases. After surpassing the initial setback, for example for Details(10, 0) with 100 reports submitted, 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 below 0.5.

Figure 4.1: 5 and 100 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 with the default score given to a newcomer, i.e, when no report was submitted yet. For example for Details(10,0) we have $score = \frac{0+1}{10+0+2} = 0.09$ and with Details(10,5) we have $score = 0.35$, this is important when we want to provide an initial setback for the user but we still want them to have a default value further from 0.

This initial setback and the default score given to newcomers is really important in order to defend against Sybil/Whitewashing attacks, because 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.

As mentioned before, the initial score details given to a user depend on the system that is being used. 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 made impossible for the trip 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. The default value should not be 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.

Forgetting Weights

After setting the initial scores details as Details(10,5), we still 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. For example, if we want to submit 100 Reports and have 10 of them be of malicious behavior then if the order is at the start then we submit first 10 reports of malicious behavior and then 90 of good behavior, if it is at the end then we first submit 90 reports of good behavior and then 10 reports of malicious behavior.

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). We tested using all ranges of values for the forgetting weights however adding them all would overload the graph, we choose instead to choose the most meaningful values. 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 the beginning or at the end is the same, the other values always have λ_s bigger than λ_r, because we want to make malicious behavior to be forgotten slower then good behavior, allowing for a slower rebuild of reputation whenever a user acts maliciously, which helps to penalize oscillatory behavior which defends against the traitor attack. The values we tried are all between 0.9 and 1, as lower values excessively prioritize the recent behavior, making old behavior to be forgotten too quickly, which is not our intention, as we want the score to reflect the user's past behavior.

The results are presented in Figure 4.2 and by looking at the graph we can see that whenever λ_s is below 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 below 0.92 is also a bad idea as introduces a drastic difference of when malicious reports are submitted at the end versus at the start. 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 malicious reports submitted at the end, which is not desirable as we want malicious behavior to drastically impact the score. The forgetting weights that do not 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 we would choose for this scenario.

Types of Malicious Behavior

Now that we looked into the most appropriate values for the parameters, Details(10,5) and Forgetting(0.98, 0.92), we can now analyse if the score is able to reflect the behavior submitted.

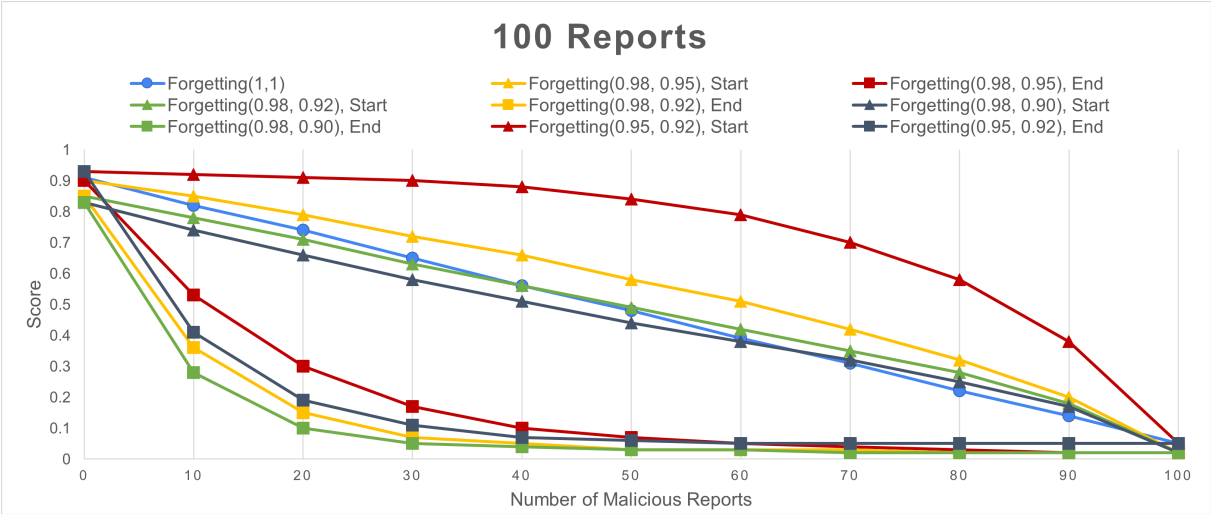

Figure 4.2: 100 Reports submitted with Details(10,5), using different forgetting weights Forgetting(λ_s, λ_r) and changing the number of intentionally malicious reports submitted and its order, which can be at the start or at the end.

For that we did some tests similar as the previous ones, where we submitted 50 and 100 reports for a single user, and once again repeated the tests where we change the number of malicious reports submitted and their order, but now instead of only submitting intentional malicious behavior, we repeat the tests submitting with different types of malicious behavior, i.e., accidentally malicious, intentionally malicious (the previous) and critically malicious. With these tests our intention is to evaluate the difference on the score when submitting the different types of malicious behavior and also analyse the score to see if it reflects the behavior of the user.

The results are shown in Figure 4.3, looking at both graphs we can see that just as in the previous tests there is a clear distinction of when the score is submitted at the end versus at the start, which shows that the score is able to prioritize recent behavior and thus fight against the traitor attack even with accidentally malicious reports and critically malicious reports. However critical malicious reports impact more drastically in the score, for example with 50 reports submitted and 15 (30%) malicious reports submitted at the start, the user already has a score that is below 0.5 when the reports are critical, on the other hand when submitting 50 reports and 5 of them are accidentally malicious and done at the end, the user still maintains a score above 0.5. As more reports are submitted, the more trust we tend to have on the user and so the less malicious behavior will affect the score, but as you can see even with 100 reports submitted, when the user behaves maliciously the score decreases drastically when submitted at the end, making bad behavior be drastically punished, even when it may be accidentally. For example, with 100 reports submitted and 10 (10%) of them malicious at the end, we still have a score below 0.5 even when all malicious reports submitted are accidental. It is also possible to see

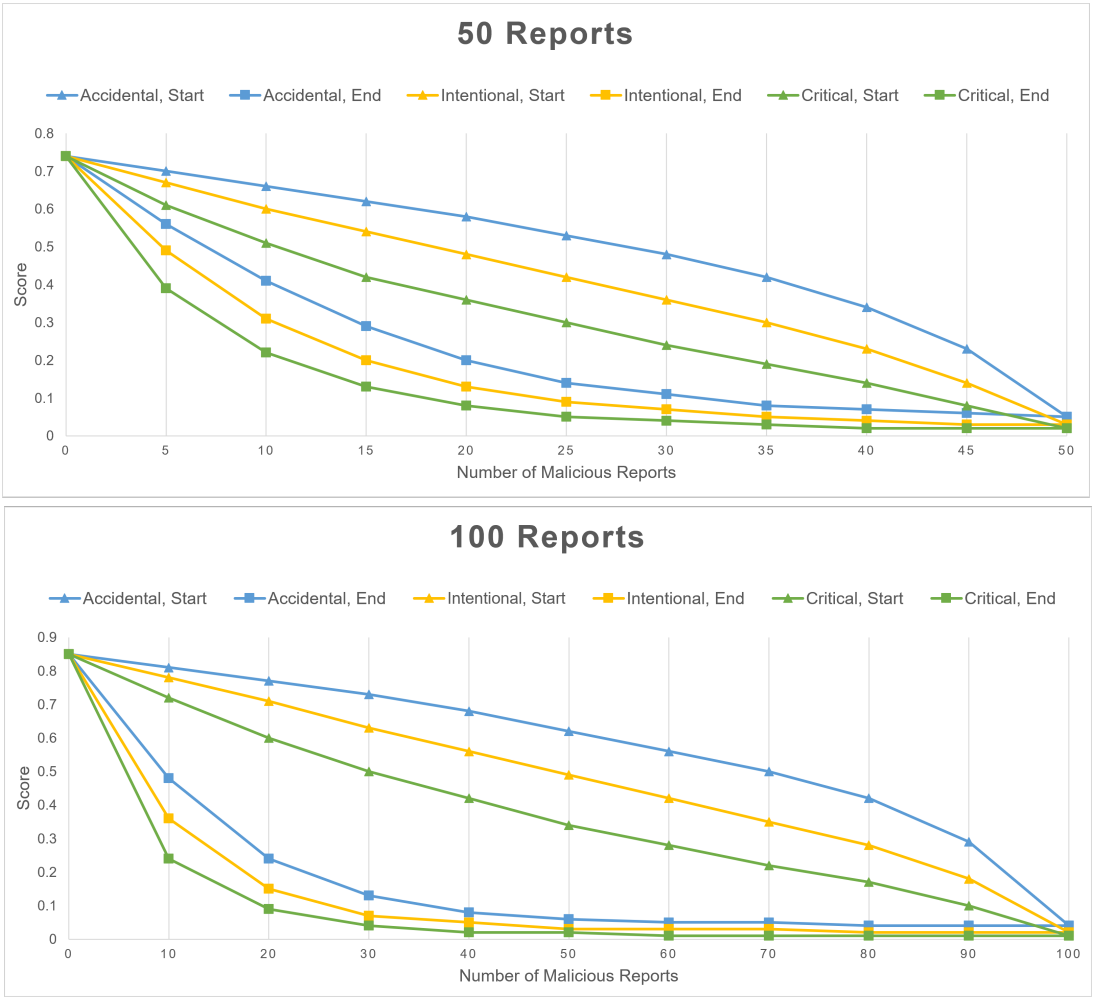

Figure 4.3: 50 and 100 reports submitted using Details(10,5) and Forgetting(0.98,.92), and changing the number of malicious reports submitted, its type (Accidental, Intentional and Critical) and its order, which can be at the start or at the end.

that the score will directly reflect the behavior submitted for the user, i.e, as more reports of malicious behavior are submitted the more the score decreases, and when a malicious action is done the user has a slow rebuild of the reputation.

Summary

We defined a clear scenario to evaluate R1 ("Provides scores that reflect the users behavior") as well as defenses against the traitor attack and the Sybil/Whitewashing attack as the specific parameters for forgetting weights and initial score details depend on what defenses the system needs to achieve. If a system needs to be highly protected against reputation attacks, then the initial value for s must be high in order to provide an initial setback to newcomers before starting to trust their good behavior and the forgetting weights must be: $\lambda_s >> \lambda_r$, to make positive behavior to be forgotten more quickly then negative behavior and making recent behavior affect

more the score. If a system wants to protect against these attacks, but still allow users to commit more mistakes and build a reputation more quickly then the initial value of s can be lower and the difference between λ_s and λ_r can also be lower. In any case it is possible to verify that the score can reflect these preferences and reflect the intended value for the user score based on its behavior.

4.1.2 Shared Reputation Evaluation

To verify that whenever a user is present in multiple application domains and is using the same identification that the system is capable of maintaining a shared reputation, which reflects its general behavior, we setup 4 SureRepute-Servers, and created an automatic testing client that allows to submit multiple reports for the same user to different SureRepute-Servers. Maintaining this general reputation allows the behavior on situations that happened on one domain to be transferred to the others, which defends against the discriminator attack.

Sequential Submission

To accurately test this requirement, 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 four servers. The number of of reports serves as an example of high number of reports to be submitted, the specific value is indifferent. The number of shared servers tested, where:

- One to attest the normal values of the reputation when all is sent to one serve;
- Two to show that when using multiple servers a shared reputation is still maintained;
- Four because even with more than the amount of application domains that currently exist (CROSS, SurePresence and STOP) SureRepute is still able to maintain a shared reputation. This is due to the communication protocol established between servers detailed in Section 3.2.1, which allow all servers to always maintain a shared reputation for the same user.

The results are shown in Table 4.1, and it is possible to see that if all the reports are submitted sequentially all servers contain the same score at the end, which validates the requirement as well as prevents a discriminator attack where a user behaves differently in each application domains, and a shared reputation is created between them.

Table 4.1: Score Results of sending 80 reports sequentially for the same user using 1, 2 and 4 servers, where the first 50% are reports of malicious behavior.

	Server Id	Malicious Behavior	Good Behavior	Score
1 SureRepute-Server	Server 1	40	40	0.46
2 SureRepute-Servers	Server 1	20	20	0.46
	Server 2	20	20	0.46
4 SureRepute-Servers	Server 1	10	10	0.46
	Server 2	10	10	0.46
	Server 3	10	10	0.46
	Server 4	10	10	0.46

Parallel Submission

We also did another scenario were we submit 40 reports using threads of always good behavior. This was also done first with a single server and then we divided the report submission to two servers and finally to four servers. The reason why we are only submitting one type of reports is because by using threads the order of the report submission can change each time we run the tests, which makes the possibility of the score change each time we run the test (as the order matters due to the forgetting weights), but this does not invalidate our requirement because our requirement is that all servers maintain a shared reputation between them.

The results are shown in Table.4.2, where it is possible to see that even when reports are submitted using concurrent threads they still maintain a shared reputation.

Table 4.2: Score Results of sending 40 reports simultaneously, using threads, for the same user using 1, 2 and 4 servers.

	Server Id	Good Behavior	Score
1 SureRepute-Server	Server 1	40	0.71
2 SureRepute-Servers	Server 1	20	0.71
	Server 2	20	0.71
4 SureRepute-Servers	Server 1	10	0.71
	Server 2	10	0.71
	Server 3	10	0.71
	Server 4	10	0.71

Summary

Based on the tests executed we show that a user in different domains is able to maintain a shared reputation that reflects their general behavior, and that there are no inconsistencies between servers, preventing the discriminator attack.

4.1.3 Privacy Evaluation

One of the requirements for our reputation system was to ensure privacy of the score to its users, i.e., the score of a specific user can only be accessed by authorized entities and so no attacker should be able to correlate a score to the real identity of a user. Our solution ensures this with two main features.

The first is that we ensure that all entities communicate with each other using TLS with client authentication enabled, which makes the need for both entities to be trusted. In order for an application domain to talk with the SureRepute-Server or with the identity provider it already has to have a certificate signed by our trusted CA. Assuming that the private keys of the entities are not discovered, this ensures that communication with the reputation system can only be done using trusted entities and the communication always guarantees *Confidentiality*, *Integrity*, *Authenticity* and *Freshness* which prevents man-in-the-middle attack and impersonation attacks.

The second is that we also use pseudonyms to ensure that the SureRepute-Servers have no access to the real identities of the users, this is done by using the identity provider as a highly trusted entity that saves the relation between the real identities and the pseudonyms and only responds to requests made by trusted application domains, where it returns the pseudonym encrypted with the respective SureRepute-Server public key to ensure that the client does not know the relation between the pseudonyms and the real identity of the user.

Let us now evaluate four scenarios where some of the information can be leaked due to a specific attack to verify if the privacy of the users can still being maintained, In the following, we consider that we are only working with one application domain, i.e. one SureRepute-Server.

"*An attacker is not a user of the system but was able to have access to the information inside the database of the SureRepute-Server*". As the information of the database does not have any information about the real identities of the users the attacker cannot correlate the score with the real identities of the users.

"*An attacker is a user of the system and has been tracking its own behavior and was able to have access to the information inside the database of the SureRepute-Server*". The attacker at most is able to figure out its own pseudonym, in the case where it is tracking its own behavior using the same techniques as the reputation system to calculate the score, i.e., using the same forgetting weights and initial score details. This situation is not problematic as it cannot correlate with others.

"*Consider the system with only one application domain, where an attacker was able to get access to the information of the database of the application domain which contains the real identities of the users and also the database of the SureRepute-Server*". The attacker cannot

correlate the real identities of all users to the score associated with the pseudonyms as they do not have anything in common (excluding the trivial case where there is only one user in using the system).

"*Consider the system with only one application domain, where an attacker got access to the information of the database of the SureRepute-Server and the database of the Identity-Provider*". This is the only case where an attacker can have access to the score associated with the real identity of the user, as it can correlate the relation between pseudonym and real identity with the score details associated with a pseudonym. Even in these case, the attacker only has access to the score not the entire behavior that was done by the users as SureRepute-Server does not store the history of all reported behaviors individually.

Summary

All communication is done using TLS with client authentication enabled, which prevents man-in-the-middle attack and impersonation attacks. We were also able to attest that when assuming that it is not possible to access the database of the identity provider, the pseudonyms ensure that only trusted entities can correlate the score of the users with the real identity of the user, and even without this assumption it is not possible to get back the history of all reported behaviors individually.

4.2 Quantitative evaluation

The quantitative evaluation of SureRepute focuses on the performance of the solution with two main questions:

- What is the overhead introduced by the reputation system when integrated with an application domain?
- Does the user score help an application domain to make better decisions?

To answer each of these questions, we used the CROSS application. The summary of how CROSS uses SureRepute when deciding if a visit is accepted is present in Section 4.2.1. In Section 4.2.2 we present the evaluation of question 1. Finally in Section 4.2.3 we present the evaluation of question 2.

4.2.1 CROSS Visit Submission

To answer these two questions, we did several tests using the CROSS smart tourism application domain. As mentioned before, CROSS users, which are called travelers must follow an itinerary

which is made of specific locations that the traveler must go to, which are called visits. In each of these visits the traveler uses an app, which is called CROSS-Client. The app is responsible for capturing the access points available and acquires/issues endorsements from/for other witnesses to support their location. After a visit is done, the location proof is sent back to the server, where it analyses the access points received as well as the witness endorsements in order to build a confidence level based on it (the confidence calculation is detailed in Section 3.3).

Each visit has a default threshold of confidence that needs to be met which is defined when creating a itinerary. The score of the traveler is used to increase the threshold needed when the score of the user is below 0.5, which is done using a linear function detailed in Equation 3.10.

The score of the witnesses are used to calculate the confidence of the witness strategy, which is calculated using Equation 3.8, where the endorsement weight is calculated using Equation 3.7, p is the prover, w_i is the specific witness, Rw_i is the reputation of the witness and the endorsementWeightTarget is the target value that needs to be met by the sum of the endorsementWeights to get 100% confidence.

CROSS submits user reports whenever a user submits a visit based on the decision made. If a visit does not meet the confidence threshold established then it submits an accidentally malicious behavior report for the traveller as we cannot be sure that the user acted maliciously on purpose. If we are certain that user is acting malicious we submit an intentionally malicious behavior report (which happens for example when the traveller submits an endorsement for himself). When the user is able to meet the threshold, CROSS submits a report of good behavior for the traveller as well as fpr witnesses that accurately endorsed the user location.

4.2.2 SureRepute Overhead in CROSS

Let us now evaluate the overhead introduced by SureRepute to CROSS when submitting visits. For these tests we defined two environments:

- *Local environment*, where the CROSS-client, CROSS-Server, SureRepute-Server and Identity-Provider were running in the same computer with an Intel® Core™ i7-8750H CPU 2.20GHz 2.21 GHz, with 16.0 GB;

- *Deployed environment*, where the 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. This cluster is defined to be deployed in the region=europe-west1, with machine-type=e2-standard-4, disk-size=20GB and nodes=2. These were the settings used for testing, but can be adjusted for different configurations.

Normally, CROSS-Client is a mobile application that captures endorsements and/or Wi-Fi access points and submit visits to CROSS-Server during a trip. However, in these case we used a testing client made in Java that makes the requests needed for the tests, in an easy and replayable way. The time we are measuring starts when we submit a request using CROSS-Client, and ends when the response is received. To evaluate the overhead introduced by SureRepute to the visit submission request in CROSS, we first evaluated the overhead introduced by a single visit submission and then the overhead that appears when simultaneous submissions by multiple travellers are being done.

Single Submission

To attest the overhead introduced in a single visit, we first setup a scenario where a traveler submits a visit with N witnesses endorsing the visit, where N is in the range [1,30]. We are assuming that the user at most crosses paths with 1 witness every 30 seconds in a 15-minute visit, which is a number that could be plausibly be achieved in a busy tourist location. We calculated the time taken to answer to the request when using SureRepute versus without using it. In this latter case the score of the users to always be 0.5, as it is the default score. We also tested this scenario in both the local and deployed environment. When we are not using SureRepute, CROSS is still being deployed in the Cloud, and thus will always have a overhead against when using locally, due to the delay of making the requests from a local computer to a remote cloud. The results are present in Figure 4.4, where we can see that when submitting a single visit in any of the environments with a different number of witnesses, SureRepute introduces a low overhead, which is at most 150ms and is barely noticeable to a user, when using the app (CROSS-Client). The reason why the overhead increases linearly with the number of witnesses is because CROSS requests the score of each witness. The Cloud deployment has more overhead overall due to the delays introduced on the communication between the CROSS-Client and the cloud as it is no longer running on the same computer, but it represents a more realistic scenario.

Simultaneous Submissions

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, because it is the average number considered in the previous experiment. Assuming that 10000 users use our app daily and an average visit has a duration of 15 minutes, the likelihood of more than 15 users submitting the visit within the same validation interval is very low: 0.0013%, which is calculated following the binomial dis-

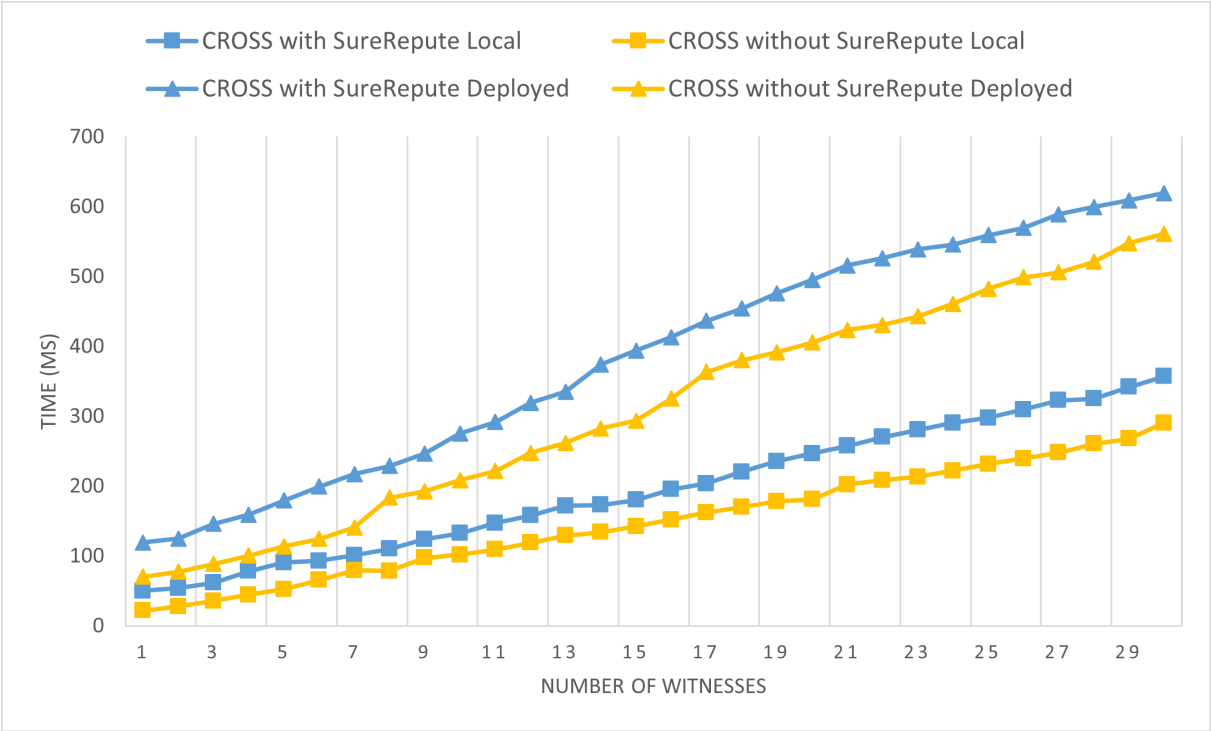

Figure 4.4: CROSS visit validation time with and without SureRepute in a local and deployed environment, varying the number of witnesses

tribution[1], with a success probability of $p = \frac{390ms}{15min} = 0.04\%$, where 390ms comes for the highest time gathered in the previous experiment with 15 witnesses, which was CROSS with SureRepute in the deployed environment. For that reason we decided to do at most 15 submissions at the same time as the probability of it happening is already very low.

The results are presented in Figure 4.5. 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 introduced overhead 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.

Both these tests helped understand that SureRepute introduces a low overhead to CROSS even in the cloud and in a unlikely scenario where a high number of submissions are being made

[1]https://www.cuemath.com/binomial-distribution-formula/

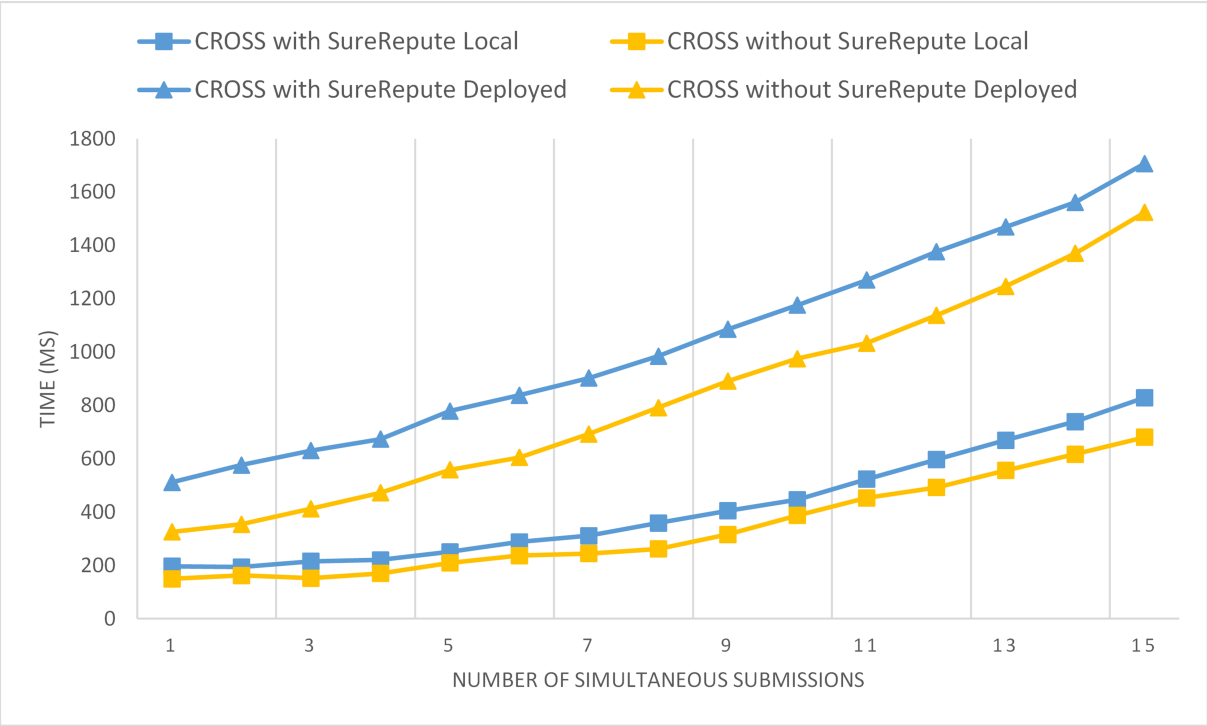

Figure 4.5: CROSS visit validation time with and without SureRepute in a local and deployed environment, varying the number of simultaneous visit submissions

exactly at the same time each with a lot of witnesses, which is what we intended.

4.2.3 SureRepute Benefits to CROSS

To attest the benefits that SureRepute introduces to CROSS, we developed some tests, where we calculated the confidence using only the peer endorsement strategy, i.e. without capturing Wi-Fi access points, as we want to attest the confidence created using only the scores of the users. The traveler score can be used to calculate the confidence threshold that needs to be reached and the score of the witnesses can be used to calculate the weight of the endorsements in the witness strategy.

Let us consider for the tests a scenario where a visit has a default confidence threshold of 75%, and travelers submit visits with at most 15 witnesses (average of previous tests), that never endorsed the traveller before as for the NpW to not change. In these scenario, we created tests where we change the travelers score in order to change the confidence threshold that needs to be reached, if the traveler has a score of 0, then it needs 100% ($1 - \frac{1-0.75}{0.5} * 0 = 1$) confidence, with 0.35 it needs 82.5% confidence ($1 - \frac{1-0.75}{0.5} * 0.35 = 0.825$) and finally with 0.5, the traveller needs 75% confidence. If a traveler has a score higher than 0.5, that does not change the default threshold as it is already considered a positive score and thus, it does not increase the threshold that needs to be achieved.

We also vary the Witness score from 0, 0.25, 0.35, 0.5, 0.75, 1. We increase scores by 0.25, starting with score of 0, the only exception is the score value of 0.35, as this is an additional value we test because it is the default score given to a new user, based on the values we setup for the score details and forgetting weights with the results given by the previous tests.

We consider that using score of 0.5 for both the prover and witness is what simulates a scenario where SureRepute is not used, i.e., where all users always have a middle score of 0.5, because there is no reputation being maintained based on the behavior of the user.

Table 4.3: CROSS acceptance rate of visits based on: prover score, number of witnesses and witness score.

Number of Witnesses		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Prover Score	Witness Score															
0.00	0.00															
	0.25															
	0.35															
	0.50															
	0.75															
	1.00															
0.35	0.00															
	0.25															
	0.35															
	0.50															
	0.75															
	1.00															
0.50	0.00															
	0.25															
	0.35															
	0.50															
	0.75															
	1.00															

Color Caption

Accepted | Rejected

The results are shown in Table 4.3, and as you can see in this table, maintaining the 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 both the witnesses and the traveller, when we have travellers that properly behave, i.e., with good score, communicate with other users that properly communicate, 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 the scores are very low, it is impossible to get a visit accepted with just the witness strategy. Looking carefully into the table, we can see that that the traveller/prover score makes the need of having at least two more witnesses, based on the default threshold given, which is really important is the score of the witnesses, as they are crucial

for the calculation of the confidence. If all witnesses have a score of 0, then not even with 15 witnesses the visit submission is accepted. If all witnesses have a score of 1, then at least 3 witnesses are needed. It is important to note that the score calculation technique makes that as the score increases, the more difficult it becomes to continue to increase, as more reports of good behavior need to be given, and any malicious behavior would significantly decrease the score. Reaching a score of 1 needs thousands of good behavior reports submitted without a single report of malicious behavior.

We can also see that if SureRepute was not being used, then the users would only need at least 4 endorsements, which would benefit malicious users because they could take advantage of the system more easily when comparing with the tracking of users behavior. This is due to the fact that only four endorsements would be needed, whereas with a reputation system it would need more. Good users would also not have any advantage at all when compared with malicious users, which would remove the encouragement of endorsing other users, which is to increase their own score.

4.3 Campus Tour: A real scenario

To complete the evaluation of the system, we attested SureRepute in a real scenario, where we used a real deployment of the smart tourism application CROSS integrated with SureRepute, as explained in Section 3.3.1. CROSS can be used by users during tourism itineraries in order to prove its location in specific points of interest, and get in return some reward for completing the trip. The scenario we created for real users to do was a campus tour in Instituto Superior Técnico on the Alameda campus in Lisbon, Portugal. We created 3 points of interest that the users needed to visit during the trip. To prepare for the tour we captured the Wi-Fi access points that could be seen in each of these points beforehand, which will allow the calculation of the wifiAPsConfidence, detailed in Section 3.3.1. The map of the campus is present in Figure 4.6, the 3 points of interest are represented in the figure as letters and are called: (A) *Pavilhão de Informática III*, (B) *Pavilhão Central* and (C) *Pavilhão de Civil*.

Two experiments were made, where the default confidence threshold for each visit was 75%, and used Equation 3.10 to calculate the confidence threshold. The difference between these experiences was the number of real users that were doing the trip in order to be able to attest the EndorsementsConfidence that uses Equation 3.8 and its value depends on the numbers of witnesses a proof has. The first experience was done with 7 users and the second with 3 users. The users followed the same path in the trip and so they were at the same point of interests at the same time.

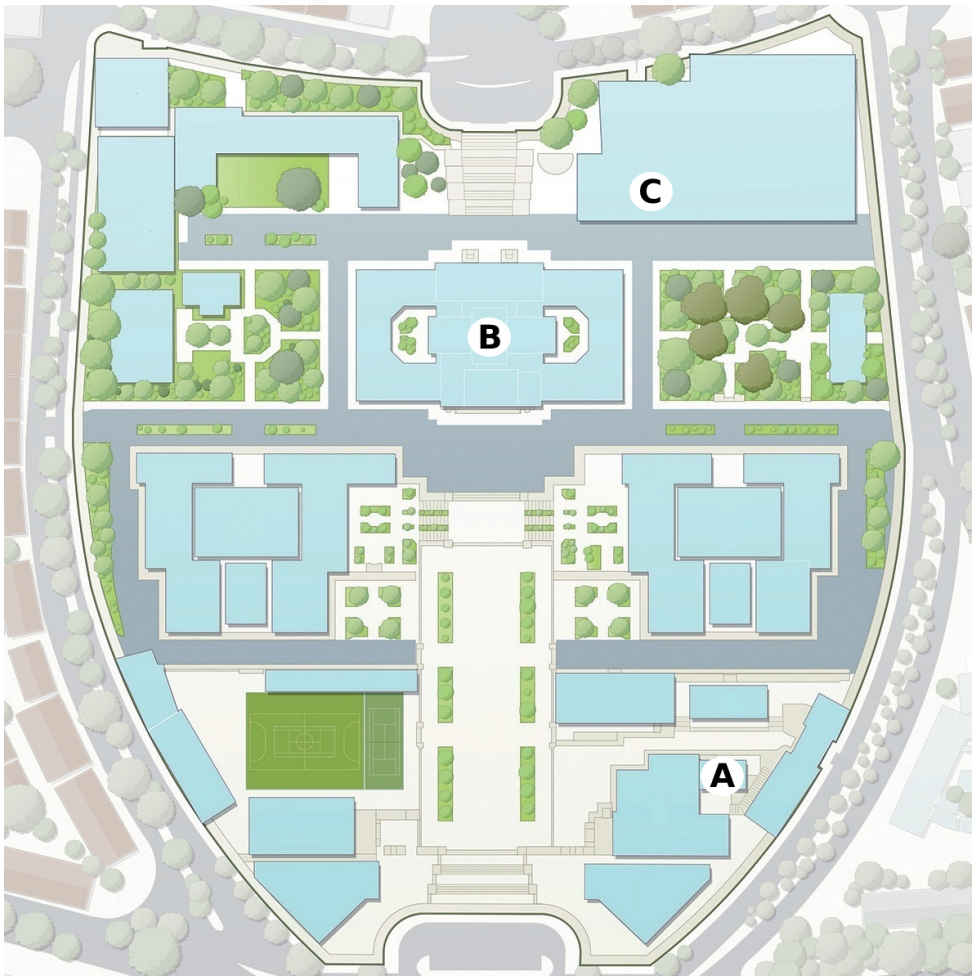

Figure 4.6: Map of the Campus Tour: (A) Pavilhão de Informática III, (B) Pavilhão Central, (C) Pavilhão de Civil

The results of the experience done with 7 users is present in Table 4.4. From this table we are able to take a few conclusions:

- In almost every visit, every user is able to be a witness to one another. When this does not happen, it can be explained by the users not being close enough to exchange bluetooth messages or because the mobile phone of the user is to slow when making the messages exchanges with other travellers which results in its timeout;
- In the first visit, only User1 has the initial score of 0.35, this is because the users do not submit reports exactly at the same time and so the witnesses that endorsed a location that was accepted, already received a report of good behavior. The users are put in the table according to the order of the first visit submission, and thus the scores are increasing, except for User5, as it only endorsed User1 yet;
- In the second and the third visit the endorsements confidence reduces because of the NpW, i.e., the number of times a witness already endorsed a traveller in different trips. NpW

Table 4.4: Campus tour details related with the confidence calculated during the experience with 7 users

POI	Traveler	Traveler Score	Nr. of Endorsements	Witnesses Avg Score	Confidence Threshold	WifiAPs Confidence	Endorsements Confidence	Confidence
Pavilhão de Informática III	User1	0.35	6	0.35	82.5%	95.5%	100.0%	100.0%
	User2	0.38	5	0.38	81.0%	100.0%	90.5%	100.0%
	User3	0.40	5	0.40	80.0%	81.8%	95.2%	100.0%
	User4	0.42	4	0.42	79.0%	86.4%	80.0%	100.0%
	User5	0.38	6	0.44	81.0%	72.7%	100.0%	100.0%
	User6	0.44	6	0.45	78.0%	100.0%	100.0%	100.0%
	User7	0.48	6	0.47	76.0%	90.9%	100.0%	100.0%
Pavilhão de Civil	User1	0.49	5	0.49	75.5%	76.7%	58.1%	100.0%
	User2	0.55	6	0.54	75.0%	60.0%	88.6%	100.0%
	User3	0.56	6	0.55	75.0%	56.7%	90.2%	100.0%
	User4	0.50	5	0.50	75.0%	63.3%	71.0%	100.0%
	User5	0.44	6	0.52	78.0%	50.0%	73.8%	100.0%
	User6	0.53	6	0.53	75.0%	80.0%	75.7%	100.0%
	User7	0.53	6	0.52	75.0%	56.7%	73.8%	100.0%
Pavilhão Central	User1	0.60	6	0.60	75.0%	35.0%	85.0%	100.0%
	User2	0.60	6	0.59	75.0%	82.4%	83.8%	100.0%
	User3	0.59	5	0.59	75.0%	56.9%	70.0%	100.0%
	User4	0.60	2	0.60	75.0%	54.9%	28.6%	83.5%
	User5	0.52	6	0.57	75.0%	52.9%	81.2%	100.0%
	User6	0.58	5	0.58	75.0%	84.3%	69.0%	100.0%
	User7	0.58	5	0.58	75.0%	58.8%	68.8%	100.0%

is apart of the endorsements confidence calculation, serving as a parameter of the witness decay mechanism provided by Grade et. al. [GPE22] work and its purpose is to reduce colluding attacks between users, making new witnesses be more trustworthy than the same ones. This means that a witness that already endorsed a traveller in previous visits will have a NpW of 1. In this scenario, the NpW will be at most 1 as it accounts endorsements in different trips, and only this trip was done by this users;

- All visits are being accepted, due to using both the endorsements confidence and the WifiAPsConfidence;

- Looking at the first visit, we can see that with 5 or more witnesses having the initial score, we can always reach the confidence threshold with only the endorsements confidence. With 4 endorsements we were able to reach the confidence threshold because the users already had increased their scores, if they were to have the initial score, it would not be possible to reach the confidence threshold with only the endorsement confidence;

- Looking at the second visit, we can see that the endorsements confidence reduced significantly due to the NpW. With the scores that they currently have and due to the NpW, only some users are able to reach the confidence Threshold using only the endorsements confidence, they have to have 6 witnesses and the their score much be high;

- Looking at the final visit, we can see the scores of witnesses are increasing, and because of that, all travellers with 6 endorsements can already reach the confidence threshold with only the endorsements confidence. The travelers with 5 endorsements have also increased their confidence.

The results of the experience with 3 users are present in the Table 4.5. From this table we can take that:

- All users are able to endorsed each other;
- The score of the users does not increase as much, as the users only endorse two other users in each visit;
- The endorsements confidence is never enough with only two witnesses to reach the confidence threshold alone, but all users are able to reach the confidence threshold due to the wifiAPs confidence, which is the percentage of the Wi-Fi APs that the users captured that where saved beforehand for that visit in the server;
- The confidence reduces in the second and third visit due to NpW;
- If the users continued to use the system their score would increase, which could make it easier to reach the confidence threshold with only the endorsements confidence.

Table 4.5: Campus tour details related with the confidence calculated during the experience with 3 users

POI	Traveler	Traveler Score	Nr. of Endorsements	Witnesses Avg Score	Confidence Threshold	WifiAPs Confidence	Endorsements Confidence	Confidence
Pavilhão de Informática	User1	0.35	2	0.35	82.5%	77.3%	33.3%	100.0%
	User2	0.38	2	0.38	81.0%	81.8%	36.2%	100.0%
	User3	0.40	2	0.40	80.0%	86.4%	38.1%	100.0%
Pavilhão de Civil	User1	0.46	2	0.46	77.0%	76.5%	21.9%	98.4%
	User2	0.42	2	0.42	79.0%	80.4%	20.0%	100.0%
	User3	0.44	2	0.44	78.0%	64.7%	21.0%	85.7%
Pavilhão Central	User1	0.48	2	0.48	76.0%	73.3%	22.9%	96.2%
	User2	0.49	2	0.49	75.5%	70.0%	23.3%	93.3%
	User3	0.50	2	0.50	75.0%	56.7%	23.8%	80.5%

To analyse how the scores got so high lets look into how many reports were submitted for each visit in both experiences. The results are present in Figure 4.6 and 4.7 for the experience with 7 and 3 users respectively.

In both experiences, the confidence gathered was always able to reach the confidence threshold, and so only good behavior reports where submitted. At the end of each visit a report of good behavior was always sent and every time a user helped another user by endorsing its visits

Table 4.6: Number of behavior reports submitted in the trip with 7 users

Traveler	Nr. of Reports by submitting a visit	Nr. of Reports made as a Witness	Nr. of Reports of Good Behavior	Traveler Score
User1	3	17	20	0.61
User2	3	18	21	0.62
User3	3	17	20	0.61
User4	3	18	21	0.62
User5	3	9	12	0.55
User6	3	16	19	0.60
User7	3	18	21	0.62

and the visit was accepted (which always happened) a good behavior report was also submitted. For the experience with 7 users the number of reports submitted due to being a witness varies a little as there are mobile phones that are not as capable as other to endorse visits, this of course affects their score. User5 submits 12 reports and has a score of 0.55 and User4 submits 21 reports and has a score of 0.62. It is also important to note that as the score increases, it becomes more difficult to continue to increase the score, as we want to prevent reputation attacks. For the experience with 3 users, the users always submitted for each other 2 endorsements and so in total each user did 9 reports, which made each user to end with the same score of 0.52.

Table 4.7: Number of behavior reports submitted in the trip with 3 users

Traveler	Nr. of Reports by submitting a visit	Nr. of Reports made as a Witness	Nr. of Reports of Good Behavior	Traveler Score
User1	3	6	9	0.52
User2	3	6	9	0.52
User3	3	6	9	0.52

In conclusion, with these experiences we were able to see that SureRepute can work in a real scenario as it is able to create a score that can reflect the behavior of its users. The experiences also showed that the system either demands the need for more witnesses or demands the additional use the WifiAPs strategy in order for a visit to be accepted, whenever the score of the witnesses/travellers are low or when most of the witnesses already endorsed the traveller before. The motivation for endorsing other users was also made clear as a user that submits 3 visits and does not make any endorsements would only receive 3 good behavior reports, but if they would endorse 2 other users in each visit, they would be able to have 6 additional reports, which would help the users achieve a higher score which increases their chances of a visit to be accepted and thus makes the completion of trips easier which means more rewards to the users.

Chapter 5

Conclusion

In this document we presented SureRepute, a reputation system that can be integrated into the SureThing framework, and allows easy integration between its different application domains. One of the application domains of the SureThing is CROSS, a smart tourism application that allows users to get rewards for visiting specific locations.

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.

The reputation system allows entities inside each application domain to submit reports of user behavior to build a reputation of the users and also to get the user reputation that can help each domain make better decisions.

Each domain can make requests to SureRepute by using the methods provided by SureRepute-Client. There is one instance of the SureRepute-Server inside each application domain, allowing that each server only handles requests for the users of that application, reducing the number of requests when integrated with multiple application domains. Servers will communicate with each other to maintain a shared reputation of users that are present in different application domains.

We deployed SureRepute using Docker Containers orchestrated by Kubernetes, that allow easy deployment to the cloud.

To evaluate the requirements of our solution as well as the defenses against the reputation attacks, we created a demo entity that uses SureRepute-Client and can make requests to one or more SureRepute-Servers. To evaluate specific details related with the integration with a real application domain, we used the tourism itinerary use case (CROSS).

To make all this work possible, first we had to reimplement the existing code base for CROSS. We did a complete refactorization of the code, changing the language from Go to Java. We also integrated CROSS with the Location Certificate Transparency (LCT) so that the emitted location certificates could be made persistent in a tamper-proof way and verifiable by third parties at a later date, as it was one of the requirements of this work. After this step we integrated SureRepute with CROSS, where the scores were used to provided weights to witness endorsements and also the traveler score was used to increase the level of confidence that needed to be achieved, which helps CROSS make better decisions when deciding if a location claim is valid or not.

We were able to assess, that the system is capable of meeting the defined requirements and also understand that the system is capable of being highly protected against reputation attacks, which reduces usability to the users, but also allows the system to relax these defenses, in favor of usability.

The performance evaluation done using CROSS also helped understand that SureRepute does not produce a significant performance overhead to the requests, and it is also helping making better decisions when deciding if a location claim should be accepted or not.

5.1 Achievements

We developed reputation system capable of creating a score for its users based on the reports received, having in mind the privacy of the users and that is capable of protect against reputation attacks.

We also developed a new score calculation technique for reputation systems that was based on the binomial Bayesian model, but it also introduces new features, by allowing different types of reports to be submitted and using different forgetting weights for good and bad behavior, which helps defend against reputation attacks.

We integrated SureRepute into tourism itinerary use case (CROSS), where a traveller needs to submit proofs that the user is in specific locations during a trip (either by interacting with other users or with deployed infrastructure), in order to earn rewards, and used SureRepute to help making better decisions.

We also contributed to other projects in SureThing by re-writing LCT and CROSS, including its data specification, API specification and documentation.

We validated SureRepute through a qualitative and quantitative evaluation based on the score calculation technique and interactions between the SureRepute entities as well as through concrete tests of SureRepute in the context of a specific domain: CROSS.

Finally, we did a campus tour in order to attest our system in a real scenario.

5.2 Future Work

Some of our assumptions are matters for future work. We assume that accounts with the same email in different application domains are always the same person, but even with different emails it could be the same person, we could have a better way of identifying if the user is the same when using different application domains. We also assume that only trusted entities can interact with the CA, which is better than assuming pre-distributed keys, as even if the certificate becomes invalid it can always request another, but it is not the ideal solution, although in our case only our entities can interact with the CA, a better solution is using some kind of authentication in order to communicate with the CA, allowing the CA to only create certificates for trusted entities (SureRepute-Clients, SureRepute-Servers and Identity Provider). We also assume that all entities of SureRepute never permanently fails, but we could also introduce fault tolerance techniques to our system.

In terms of advancements outside of the assumptions, the LCT, could be further explored to make the monitor also analyse the contents of the proofs and thus also submit reports based on suspicious behavior to SureRepute. Rotating pseudonyms could be incorporated into SureRepute in order to further improve the privacy of the users with a more advanced technique. SureRepute could be integrated with the other application domains that currently exist or will exist of the SureThing project.

Finally, we leave the possibility of integrating SureRepute outside of SureThing framework possibly outside of the specifics of location certifications, as we believe SureRepute could also be valuable for many crowd-based mobile and Internet of Things (IoT) applications.

Bibliography

[AH16] Ahmad J Abdel-Hafez. *Reputation model based on rating data and application in recommender systems*. PhD thesis, Queensland University of Technology, 2016.

[CERP21] Pedro Carvalho, Samih Eisa, Leonardo S Rocha, and Miguel L Pardal. Location certificate transparency for third-party-verifiable location proofs. Master's thesis, Instituto Superior Técnico, Universidade de Lisboa, 2021.

[CP18] Diogo Calado and Miguel L Pardal. Tamper-proof incentive scheme for mobile crowdsensing systems. In *IEEE 17th International Symposium on Network Computing and Applications (NCA)*, pages 1–8. IEEE, 2018.

[CRKH11] Delphine Christin, Andreas Reinhardt, Salil S Kanhere, and Matthias Hollick. A survey on privacy in mobile participatory sensing applications. *Journal of systems and software*, 84(11):1928–1946, 2011.

[dC20] João Paulo Nunes da Costa. A witness protection program for a privacy-preserving location proof system. Master's thesis, Instituto Superior Técnico, Universidade de Lisboa, 2020.

[DDKJ21] Guntur Dharma Putra, Volkan Dedeoglu, Salil S Kanhere, and Raja Jurdak. Blockchain for Trust and Reputation Management in Cyber-physical Systems. *arXiv e-prints*, page arXiv:2109.07721, September 2021.

[FP18] Joao Ferreira and Miguel L Pardal. Witness-based location proofs for mobile devices. In *IEEE 17th International Symposium on Network Computing and Applications (NCA)*, pages 1–4. IEEE, 2018.

[Fra21] Miguel Cordeiro Francisco. Surepresence: Location proofs for wearable and kiosk devices. Master's thesis, Instituto Superior Técnico, Universidade de Lisboa, 2021.

[GFJ09] Florent Garcin, Boi Faltings, and Radu Jurca. Aggregating reputation feedback.

In *Proceedings of the First International Conference on Reputation: Theory and Technology*, volume 1, pages 62–74, 2009.

[GPE22] Ricardo Grade, Miguel Pardal, and Samih 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.

[HBC15] Ferry Hendrikx, Kris Bubendorfer, and Ryan Chard. Reputation systems: A survey and taxonomy. *Journal of Parallel and Distributed Computing*, 75:184–197, 2015.

[HZNR09] Kevin Hoffman, David Zage, and Cristina Nita-Rotaru. A survey of attack and defense techniques for reputation systems. *ACM Computing Surveys (CSUR)*, 42(1):1–31, 2009.

[IFMM10] Antonio Iera, Christian Floerkemeier, Jin Mitsugi, and Giacomo Morabito. The internet of things [guest editorial]. *IEEE Wireless Communications*, 17(6):8–9, 2010.

[JI02] Audun Josang and Roslan Ismail. The beta reputation system. In *Proceedings of the 15th BLED Electronic Commerce Conference*, volume 5, pages 2502–2511, 2002.

[KK11] Amit Kushwaha and Vineet Kushwaha. Location based services using android mobile operating system. *International Journal of Advances in Engineering & Technology*, 1(1):14, 2011.

[KT12] Eleni Koutrouli and Aphrodite Tsalgatidou. Taxonomy of attacks and defense mechanisms in p2p reputation systems—lessons for reputation system designers. *Computer Science Review*, 6(2-3):47–70, 2012.

[MCP20] Gabriel A Maia, Rui L Claro, and Miguel L Pardal. Cross city: Wi-fi location proofs for smart tourism. In *International Conference on Ad-Hoc Networks and Wireless*, pages 241–253. Springer, 2020.

[MGM03] Sergio Marti and Hector Garcia-Molina. Identity crisis: anonymity vs reputation in p2p systems. In *Proceedings Third International Conference on Peer-to-Peer Computing (P2P2003)*, pages 134–141. IEEE, 2003.

[MMH+15] Hayam Mousa, Sonia Ben Mokhtar, Omar Hasan, Osama Younes, Mohiy Hadhoud, and Lionel Brunie. Trust management and reputation systems in mobile participatory sensing applications: A survey. *Computer Networks*, 90:49–73, 2015.

[MPQ+19] Lichuan Ma, Qingqi Pei, Youyang Qu, Kefeng Fan, and Xin Lai. Decentralized privacy-preserving reputation management for mobile crowdsensing. In *International Conference on Security and Privacy in Communication Systems*, pages 532–548. Springer International Publishing, 2019.

[NSY+20] Mohammad Reza Nosouhi, Keshav Sood, Shui Yu, Marthie Grobler, and Jingwen Zhang. Pasport: A secure and private location proof generation and verification framework. *IEEE Transactions on Computational Social Systems*, 7(2):293–307, 2020.

[RKZF00] Paul Resnick, Ko Kuwabara, Richard Zeckhauser, and Eric Friedman. Reputation systems. *Communications of the ACM*, 43(12):45–48, 2000.

[SCRP20] Henrique F Santos, Rui L Claro, Leonardo S Rocha, and Miguel L Pardal. Stop: A location spoofing resistant vehicle inspection system. In *International Conference on Ad-Hoc Networks and Wireless*, pages 100–113. Springer, 2020.

[SJ04] Jean-Marc Seigneur and Christian Damsgaard Jensen. Trading privacy for trust. In *International Conference on Trust Management*, pages 93–107. Springer, 2004.

[TCB10] Manoop Talasila, Reza Curtmola, and Cristian Borcea. Link: Location verification through immediate neighbors knowledge. In *International Conference on Mobile and Ubiquitous Systems: Computing, Networking, and Services*, pages 210–223. Springer, 2010.

[WAB+18] Alexandra Wood, Micah Altman, Aaron Bembenek, Mark Bun, Marco Gaboardi, James Honaker, Kobbi Nissim, David R O'Brien, Thomas Steinke, and Salil Vadhan. Differential privacy: A primer for a non-technical audience. *Vanderbilt Journal of Entertainment and Technology Law*, 21:209, 2018.

[YAA08] Jiang Yang, Lada A Adamic, and Mark S Ackerman. Crowdsourcing and knowledge sharing: strategic user behavior on taskcn. In *Proceedings of the 9th ACM conference on Electronic commerce*, pages 246–255, 2008.