Privacy-preserving Event-sharing Android App Implementation and Testing

Fisher, John
Curs 2013-2014

Director: JOAN MELIÀ SEGUÏ
GRAU EN ENGINYERIA INFORMÀTICA
Treball de Fi de Grau
Privacy-preserving Event-sharing Android App Implementation and Testing

John Fisher

Undergraduate Thesis / 2014

THESIS ADVISOR
Joan Melià Seguí, PhD Department of Information and Communication Technologies
ABSTRACT

Geo-social Networks (GSNs), like Foursquare, have proliferated in recent years, potentially allowing recommender systems to leverage this new rich source of data to improve their results. Current recommender systems don’t take user privacy into account though, which could expose their users to new types of privacy threats. Other literature has defined four main privacy aspects: location-, absence-, co-location-, and identity privacy. Mainly through the use of a Private Set Intersection (PSI) protocol, we tackle the first three in this thesis, in the context of recommending a convenient place and time for \( n \) users to meet up, based on their location histories. For the case of \( n = 2 \), we have developed an Android application with two settings, normal and privacy-preserving. For the case of \( n > 2 \), we have developed a server application to coordinate the efforts of the Android application. This thesis describes the presented problem and documents the process followed to design and implement this software. We analyse the cost of implementing privacy-preservation by comparing the Android application’s battery, time, and network usage between the normal mode and the privacy-preserving mode. The thesis concludes that, while significant, the costs of privacy preservation are assumable. Future work proposed by the thesis includes extending the PSI protocol to work on the server for \( n > 2 \), as currently the server is assumed to be a trusted third party and receives all users’ full location histories.
Les xarxes geosocials (XGS), com Foursquare, han proliferat els últims anys, permetent, potencialment, als sistemes de recomanació aprofitar aquesta nova i rica font de dades per millorar els seus resultats. Els sistemes actuals de recomanació, però, no tenen en compte la privacitat dels seus usuaris, el que podria exposar aquests a nous tipus d’amenaça de privacitat. Altra literatura ha definit quatre aspectes principals de privacitat: d’ubicació, d’absència, de co-ubicació, i d’identitat. Principalment a través de l’ús d’un protocol PSI, aquest treball aborda les tres primeres amenaços, en el context de recomanar un lloc i hora de trobada convenient per a $n$ usuaris, basant-se en els seus històrics d’ubicacions. Pel cas d’$n = 2$, s’ha desenvolupat una aplicació Android amb dos modes: normal i amb preservació de la privacitat. Pel cas d’$n > 2$, s’ha desenvolupat una aplicació de servidor per coordinar els esforços de l’aplicació Android. Aquest treball descriu el problema presentat i documenta el procés seguit per dissenyar i implementar aquest programari. Es dóna un anàlisi del cost d’implementar la preservació de privacitat comparant l’ús de bateria, xarxa i temps de l’aplicació Android entre el mode normal i el que preserva la privacitat. El treball conclou que, mentre és significatiu, el cost de preservar la privacitat és assumible. Treball futur proposat pel treball inclou estendre el protocol d’intersecció privat de conjunts per a que funcioni al servidor per $n > 2$, ja que actualment és suposat que el servidor és un tercer de confiança i rep els històrics d’ubicacions de tots els usuaris.
to Kathleen and Hubert Renusz,
Annie Mary O’Hare and Albert Fisher I,
Lydia and Albert Fisher II
Acknowledgements

I would like to thank my advisor, Joan Melià, for the opportunity to collaborate with him after an innocent visit to his office inquiring about his research. This has been an intellectually challenging and rewarding journey, requiring me to combine the wide range of knowledge learned while at university, from mathematics to algorithms to software engineering. Joan’s guidance has been instrumental in the completion of this thesis.

I would also like to thank Emiliano De Cristofaro, for taking the time to give advice and answer my questions about the private set intersection protocol. This thesis would also not have been possible without him.

Special mention also goes to my good friends Enric León Font, for help with power consumption measurement, and to Pål Røynestad, for help with evaluating the results.

Most importantly, I'd like to thank my family. My parents, Albert and Lydia; my siblings Leona and Albert; and my partner, Eli, for the emotional support, encouragement and for suffering through proof readings.
Contents

ABSTRACT iii
RESUM v
INDEX OF FIGURES xvi
INDEX OF TABLES xvii
LIST OF ACRONYMS xix
1 INTRODUCTION 1
2 STATE OF THE ART 5
  2.1 Related work ................................. 5
  2.2 Baseline ...................................... 6
  2.3 Tools ......................................... 6
    2.3.1 Budget .................................. 7
3 PRIVATE SET INTERSECTION 9
  3.1 System Model ................................. 9
  3.2 Location Prediction .......................... 10
    3.2.1 Intermediate Matrix ......................... 10
    3.2.2 Final Probability Matrix .................... 10
    3.2.3 Candidate Locations and Times ............. 12
  3.3 Privacy Preserving Sharing .................. 12
4 IMPLEMENTATION 15
  4.1 EventProbability Library ..................... 15
    4.1.1 EventProbabilityTest Class ............... 17
    4.1.2 EventProbability Class .................... 17
  4.2 Privacy Preserving Library .................. 18
    4.2.1 EncryptionTest ............................ 19
# List of Figures

4.1 Results of profiling the EventProbability library . . . . . . . . . . 16  
4.2 Results of profiling the EventProbability library . . . . . . . . . . 17  
4.3 EventProbability Unified Modelling Language (UML) class diagram 18  
4.4 PSIClient UML class diagram . . . . . . . . . . . . . . . . . . . 19  
4.5 PSIServer UML class diagram . . . . . . . . . . . . . . . . . . . 20  
4.6 EnergyConsumptionCounter UML class diagram . . . . . . . . . . 21  
4.7 The three screens in PrivaCerver. (a) Settings (b) Locations (c)  
Results . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 22  
4.8 Use case diagram for Peer-to-peer (P2P), non encrypted sharing . 23  
4.9 Sequence diagram for P2P, non encrypted sharing . . . . . . . . . 23  
4.10 Sequence diagram for P2P, encrypted sharing. Note the increased  
complexity compared to Figure 4.9 . . . . . . . . . . . . . . . . . 24  
4.11 Use case diagram for Server trusted third party sharing . . . . . . 26  
4.12 Sequence diagram for Server trusted third party sharing . . . . . . 26  
5.1 Performance degradation comparison . . . . . . . . . . . . . . . . 35  
6.1 Candidate generation comparison between clear text and encrypted 38  
6.2 Candidate sharing comparison between clear text and encrypted . 40  
6.3 Network traffic comparison . . . . . . . . . . . . . . . . . . . . . 42  
6.4 Comparison of energy consumed during candidate generation vs.  
candidate sharing . . . . . . . . . . . . . . . . . . . . . . . . . . 43  
6.5 Comparison of time elapsed during candidate generation vs. can-
didate sharing . . . . . . . . . . . . . . . . . . . . . . . . . . . . 44  
6.6 Full cycle comparison between clear text and encrypted . . . . . . 45  
7.1 Candidate sharing comparison between clear text and encrypted . 50  
7.2 Comparison of energy consumed during candidate generation vs.  
candidate sharing . . . . . . . . . . . . . . . . . . . . . . . . . . 51  
A.1 Project timeline estimation . . . . . . . . . . . . . . . . . . . . . 65  
B.1 Gantt chart work breakdown structure . . . . . . . . . . . . . . . 70
B.2 Gantt chart work breakdown structure diagram . . . . . . . . . . 71

E.1 UK candidate generation comparison . . . . . . . . . . . . . . . . . . . 110
E.2 UK candidate sharing comparison . . . . . . . . . . . . . . . . . . . . . 111
E.3 UK network traffic comparison . . . . . . . . . . . . . . . . . . . . . . . . 112
E.4 UK comparison of energy consumed . . . . . . . . . . . . . . . . . . . . 113
E.5 UK comparison of candidate generation time . . . . . . . . . . . . . . . 114
E.6 UK full cycle comparison . . . . . . . . . . . . . . . . . . . . . . . . . . . 115
List of Tables

2.1 Budget line items .................................................. 7
3.1 Summary of the notation used in this chapter .............. 11
5.1 Reference NY user data ............................................. 34
5.2 UK user data .......................................................... 34
7.1 Privacy preserving penalties as a percentage increase, in isolation and in aggregate .................................. 50
A.1 Milestones and estimated completion timeframe .......... 66
A.2 Milestones and estimated completion timeframe .......... 67
LIST OF ACRONYMS

ADB  Android Debug Bridge .......................................................... 36

AP   Access Point ........................................................................ 25

APM  Application Performance Monitoring ................................. 30

API  Application Programming Interface ................................... 15

CSV  Comma-separated Values ...................................................... 15

DRY  Don’t Repeat Yourself .......................................................... 15

GB   Gigabyte ............................................................................ 6

GHz  Gigahertz ........................................................................... 6

GSN  Geo-social Network ............................................................... iii

GPS  Global Positioning System ..................................................... 5

HTTP Hyper-text Transfer Protocol ....................................... 18

IDE  Integrated Development Environment ............................... 6
I/O Input/Output ................................................................. 18

JSON JavaScript Object Notation ........................................ 18

IS Invention Submission ...................................................... 47

MWE Minimum Working Example ........................................ 32

P2P Peer-to-peer .............................................................. 47

PC Personal Computer ....................................................... 6

POI Point of Interest ........................................................... 5

PSI Private Set Intersection ................................................ 47

RAM Random Access Memory ............................................. 6

SSD Solid-state Drive .......................................................... 6

TCP/IP Transmission Control Protocol/Internet Protocol ............ 25

UI User Interface ............................................................... 21

UID User ID ................................................................. 31

UML Unified Modelling Language ........................................ xv

UX User Experience .......................................................... 15

VCS Version Control System ................................................ 15
Chapter 1

INTRODUCTION

The amount of information available on the Internet nowadays is overwhelming. There has been much research performed in recommender systems as documented by Ricci et al. in [1], with the aim of unearthing relevant content to each user.

Ubiquitous computing has created exciting opportunities for both businesses and customers. For example, the proliferation of GPS enabled smartphones connected to the Internet, coupled with Geo-social networks (GSNs) like Foursquare and other location-tracking services such as Google’s Location History has allowed the creation of rich and detailed location history profiles for these services’ users. Google’s Location History tracks the every movement of those Android users that have accepted the relevant prompt, as Kumparak describes in [2]. This added context can be used to achieve an even higher degree of personalisation and relevance in recommendation systems.

Until recently, users’ privacy has not been a main consideration when designing recommender systems, if at all. In describing early recommender systems, Resnick and Varian in [3] were very forward thinking to mention privacy as a concern. The words “privacy” or “private” don’t appear in Twitter’s paper on their recommendation system [4], nor in the submission that won the “Netflix Prize” [5]. Ricci et al. [1] conclude that:

[T]here is a need to design [recommender systems] that will parsimoniously and sensibly use user data. [...] ensuring that knowledge about the users cannot be freely accessed by malicious users.

An example of privacy erosion in recommender systems is Target in [6]. Target’s [7] recommender system identified a customer as being pregnant and sent her coupons for newborn related products. Unknown to the recommender, this customer was seventeen, still at high school and hadn’t told her father about the
pregnancy. When the coupons arrived, the father saw them and, angry, demanded to see his local Target’s manager, accusing the company of encouraging his high school daughter to get pregnant. During a follow-up call, the father abashedly explained to the manager that his daughter was, in fact, pregnant.

Adding detailed location information —whether automatically collected, or manually via “check ins”— to the mix increases the potential for a user’s privacy to be invaded. Ruiz Vicente et al. in [8] define four privacy notions, of which this thesis focuses on the following three:

- **Location Privacy**: a notion whereby disclosure of a user’s location at a specific time allows inferences to be made regarding “health problems, affiliations, and habits” [8, p. 22], e.g. checking in to an Alcoholics Anonymous venue at a time when a support session is scheduled.

- **Absence Privacy**: where publishing a user’s location (exact or approximate) allows an adversary to determine the user’s absence from any other location, e.g. checking in to a holiday resort in another country implies the user is not at home.

- **Co-location privacy**: which allows an adversary to deduce multiple unwilling users’ presence at a location through observation of third party profiles where the unwilling users are mentioned. E.g.

  Alice and Bob are secretly meeting at a bar when a friend of Alice sees her and mentions so in her check-in at the bar. A friend of Bob does the same with Bob. Thus the adversary with access to the friends’ profiles can deduce that Alice and Bob were both at the bar.

  An example of co-location privacy being pierced is mentioned by Sadilek et al. [9, §7].

  For example, our unsupervised experiments show that location can be inferred even for people who keep their tweets and location private, and thus may believe that they are “untrackable.”

  Another concern regarding privacy is raised by Aïmeur, et al. [10], whereby recommender systems that aren’t privacy-aware leave open the possibility for the owner of the service to abuse the gathered information by selling it or using it to build invasive behavioural models such as the Target example from before.

  The aim of this thesis is twofold. The first aim is to implement a system consisting of an Android application and a server application in such a manner
as to mitigate these threats to users’ privacy when sharing their location history and habits with either recommender systems or with other users. The second aim is to analyse the performance penalty incurred by offering privacy protection by comparing power consumption, network traffic and duration between the normal, privacy preserving, mode and a mode where data is transferred in cleartext.

The system tries to solve the problem of finding a time and place that is convenient for two or more users, based on their historic location data. An example of this problem could be a buyer and seller agreeing on a location and time to finalise the transaction of an item from a classified advertisements website. Craigslist [11] is a well-known American example. As the buyer and seller probably don’t know each other, either one disclosing their place of residence could be a risk. A better option would be to find a time and place other than home that is convenient for both parties. Doing so without disclosing anything other than the most convenient place and time would be the best option as, again, both parties probably don’t know anything about each other.

The system is based on De Cristofaro et al.’s Invention Submission (IS) [12], which proposes a method to preserve a user’s privacy when solving this problem. The method analyses a user’s location history and constructs a set of places and times where that user tends to be: candidate locations and times, or candidates for short. This set of candidates is compared with one or more other user’s sets of candidates to determine which, if any, candidates are in common. This way, the users are presented with a list of times and places they know are convenient for everyone involved. This method ensures that, given another remote user’s set of candidates, the local user will only have access to the remote candidates which are in common with the local ones. That is to say, if Alice’s candidates set $A = \{l_{a1}, l_{a2}, \ldots\}$ is shared with Bob’s candidates set $B = \{l_{b1}, l_{b2}, \ldots\}$ then any candidate $l_{an} \notin B$ will be indecipherable to Bob.
Chapter 2

STATE OF THE ART

In this chapter we give a brief overview of related work, describe the baseline from where our work starts, the tools used in this thesis and their cost.

2.1 Related work

Much research into location recommendation has been performed. User privacy is not a primary concern, if it is one at all.

Zheng et al. [13] focus on Global Positioning System (GPS) tracking, Point of Interest (POI) extraction and activity correlation based on location to recommend places to go and activities to perform once there. Sadilek et al. [9] leverage geo-tagging in public Twitter messages to build a system with natural language processing and machine learning capabilities to predict user locations, even if the users have explicitly kept their data private:

[The system] infers people’s fine-grained location, even when they keep their data private and we can only access the location of their friends.

Zheng et al. describe a system for proximity testing while keeping the initiating user’s actual location private [14]. In summary, this technique computes location tags using Bloom filters, paired with fuzzy extractors to add randomness based on a key of the initiating user’s current location. Nearby users will have a similar key, allowing them to recover the original message using error correction, and indicate to the initiating user that they are close by. Users who are too far away cannot correct all the errors, automatically filtering them out. Our work differs in that the proximity of users is not a factor, and our aim is to recommend a future location and time that is convenient for involved parties.
Magitti is a leisure activity guide [15], analysing a user’s past activity and comparing it to the user’s demographic aggregate habits to predict user activity and recommend venues for those activities. Our work focuses on sharing specific users’ habitual locations with other users, to find convenient common locations and times to meet up.

Prior work that does take privacy into consideration when sharing location data has focused on user-provided data in private scheduling, e.g. calendar entries manually created by the user, as described by Bilogrevic et al. [16]. Our work seems to be the first to integrate future activity recommendation with privacy preserving sharing.

2.2 Baseline

The baseline of our work is an IS submitted by De Cristofaro et al., which is “related to performing computations on user locations while preserving a user’s privacy” [12]. Summarising, it describes a method or apparatus to recommend future locations a user is likely to visit, and when they will do so, based on the analysis of their location history. The location history can be acquired from a variety of sources, among which Foursquare check ins is used as an example. After generating this list of candidate locations and times, the system enables the user to share it with other users in a private manner. Only the locations and times in common to all users involved are disclosed to the users.

In this thesis we implement a prototype of this IS. It consists of an Android client and a Personal Computer (PC) server in Java. We give details of the implementation in Chapter 4.

2.3 Tools

For testing we used a Samsung Galaxy SIII Mini running Android 4.1.2 and a Samsung Galaxy running Android 4.2.2. The Integrated Development Environments (IDEs) used are Android Studio to develop the Android client and IntelliJ IDEA Community Edition to develop the server application. We developed and tested on a laptop with a 2.13 Gigahertz (GHz), dual-core processor with 8 Gigabytes (GBs) of Random Access Memory (RAM) and a 256 GB Solid-state Drive (SSD).
2.3.1 Budget

The budget for this thesis was two-hundred Euro, as we had to upgrade the development machine with more RAM and an SSD due to it being four years old. The rest of the tools were either free or we already had them. For illustrative purposes, though, Table 2.1 contains the budget line items had it been necessary to purchase them all.

Table 2.1: Budget line items

<table>
<thead>
<tr>
<th>Item</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Android devices: 2x Galaxy SIII Mini</td>
<td>€180/ea.</td>
</tr>
<tr>
<td>IntelliJ IDEA</td>
<td>€179</td>
</tr>
<tr>
<td>Development machine</td>
<td>€1000</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>€1539</strong></td>
</tr>
</tbody>
</table>

2.3.1 Budget

The budget for this thesis was two-hundred Euro, as we had to upgrade the development machine with more RAM and an SSD due to it being four years old. The rest of the tools were either free or we already had them. For illustrative purposes, though, Table 2.1 contains the budget line items had it been necessary to purchase them all.
Chapter 3

PRIVATE SET INTERSECTION

We achieve privacy preservation when sharing candidate locations and times through the use of a Private Set Intersection (PSI) protocol, based on the one described by De Cristofaro, et al. in [17]. In this chapter we provide an overview of how it works, with Table 3.1 summarising the notation used.

3.1 System Model

Without loss of generality, we consider two users, Alice and Bob. They want to find out if their typical daily schedules have any overlap, i.e. if they are ever both at the same place at the same time. This would allow them to find a convenient place and time to meet up, with minimum hassle for both. However, they do not want to share all of their typical whereabouts with one another; they just want to find the overlap.

Time Slots In our implementation we have defined the timeslots to be of one hour, though this could become a user-defined setting. In that case, Alice and Bob would have to agree on the same granularity. All one-hour timeslots in the week, or weekday-hour\(^1\), are considered, giving the following set

\[ W = \{ w_h \mid \forall w \in \text{Monday, ..., Sunday}, \forall h \in (00, ..., 23) \} \]

Locations Alice and Bob frequently check in to Foursquare, the past history of which we use to predict their most likely location for each timeslot. In this thesis we call this the set of candidate locations and times, or candidates, for short. We denote with \( L_{A:w_h} \) Alice’s predicted location for a

\(^1\)Strictly speaking, it is day-of-week–hour, from Monday to Sunday and 00h–23h. We thought this was a mouthful, though, and opted for weekday-hour instead.
given weekday-hour \( w_h \). Bob’s predicted location for a given weekday-hour is \( L_{B:w_h} \). Locations are uniquely identified by their GPS coordinates in our implementation.

**Privacy-preserving Meet-up** After sharing candidate sets, Alice and Bob only learn \( \{ w_h \mid L_{A:w_h} = L_{B:w_h} \} \). That is, Alice only learns about the candidates in Bob’s set (and vice versa) if she also has the same candidate in her set.

### 3.2 Location Prediction

Predicting a user’s future location is done in three steps. First we generate an intermediate matrix \( M_{\text{intermediate}} \), which is then used to generate the final probability matrix \( M_{\text{probability}} \). Finally we create a set \( C \) of the candidate locations from the information in the probability matrix.

#### 3.2.1 Intermediate Matrix

The intermediate matrix arranges the information provided by a user’s events in a way that makes it easy for us to calculate the probabilities of them being at a certain place on a certain weekday-hour.

Given the set \( E = \{ e_i \mid e_i = \langle time_i, location_i \rangle \} \) of a user \( U \)’s events, we generate the intermediate matrix by segmenting each event in \( E \) by weekday-hour. We define \( \{ w_h \} \) sets \( S_{w_h} = \{ e \in E \mid \text{timeslot}(e) = w_h \} \), where \( \text{timeslot}(x) \) is a function that, given an event \( x \), returns the weekday-hour \( w_h \) of that event. That is, each event is added to the appropriate set of events according to the weekday-hour it occurred.

\[
M_{\text{intermediate}} = \begin{pmatrix}
S_{Mon00} & S_{Mon01} & \cdots & S_{Mon23} \\
S_{Tue00} & S_{Tue01} & \cdots & S_{Tue23} \\
\vdots & \vdots & \ddots & \vdots \\
S_{Sun00} & S_{Sun01} & \cdots & S_{Sun23}
\end{pmatrix}
\]

#### 3.2.2 Final Probability Matrix

The final probability matrix tells us how likely it is for a user to be at a certain place on a certain weekday-hour.

Using the intermediate matrix we generate the final probability matrix \( M_{\text{probability}} \). For each weekday-hour slot, this matrix stores the probabilities of the user being at each location \( L_n \).

10
<table>
<thead>
<tr>
<th>Notation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$w_h$</td>
<td>Weekday-hour slot corresponding to day of week $w$, from Monday to Sunday, and hour $h$ from 00 to 23.</td>
</tr>
<tr>
<td>$L_{A;w,h}$</td>
<td>The location a user $A$ is expected to be for a given weekday-hour.</td>
</tr>
<tr>
<td>$M_{\text{intermediate}}$</td>
<td>The intermediate matrix containing a user’s past events categorised by the weekday-hour they occur.</td>
</tr>
<tr>
<td>$M_{\text{probability}}$</td>
<td>The final probability matrix containing the probabilities for a user being at each location during each weekday-hour slot of the week.</td>
</tr>
<tr>
<td>$E$</td>
<td>User’s set of past events. Each event has a discrete date and time when it occurred, and the location it occurred at.</td>
</tr>
<tr>
<td>timeslot($e$)</td>
<td>Function returning the weekday-hour slot $w_h$ it occurred at, according to the discrete date and time.</td>
</tr>
<tr>
<td>$S_{w,h}$</td>
<td>Set of user’s events occurring during the given weekday-hour slot, e.g. for $w = \text{Sunday}$, $h = 13$, all user’s events occurring on Sundays at 13h.</td>
</tr>
<tr>
<td>$L_n$</td>
<td>Location where an event occurs. It consists of the GPS coordinates and a friendly name for the event.</td>
</tr>
<tr>
<td>$P_{w,h}(L)$</td>
<td>The probability of a user being at a location $L$ for a given weekday-hour $w_h$.</td>
</tr>
<tr>
<td>$C$</td>
<td>Set of candidate locations and times for a user. Each candidate consists of a weekday-hour slot and a location.</td>
</tr>
<tr>
<td>cand($w_h$, $L$)</td>
<td>A function that creates a candidate for a given weekday-hour slot and a location.</td>
</tr>
<tr>
<td>max($P_{w,h}(L_i)$)</td>
<td>A function that, for a given weekday-hour slot, returns the location that the user has the highest probability of being at.</td>
</tr>
<tr>
<td>$\mathbb{P}$</td>
<td>Set of prime numbers.</td>
</tr>
<tr>
<td>$\mathbb{R}$</td>
<td>Set of real numbers.</td>
</tr>
<tr>
<td>$R_1$</td>
<td>Random value used as a salt for hashing.</td>
</tr>
<tr>
<td>$H(x)$</td>
<td>Function returning the hashed value of $x$.</td>
</tr>
</tbody>
</table>

Table 3.1: Summary of the notation used in this chapter
We calculate each probability $P_{wh}(L_i)$ of a user being at a location $L_i$ on a weekday $w$ at the timeslot $h$ by calculating the mean of the number of events occurring at that location at that time with respect to the total number of events occurring in that slot:

$$P_{wh}(L_i) = \frac{|\{e \in S_{wh} \mid \text{loc}(e) = L_i\}|}{|S_{wh}|}$$

Where $\text{loc}(x)$ is a function that, given an event $x$, returns the location associated to that event.

### 3.2.3 Candidate Locations and Times

The last step is to define a set of candidate locations and times, i.e., those places a user is likely to be, and when they are likely to be there.

We define a candidate $c_{wh} = (w_h, L)$. That is, a candidate consists of a weekday-hour and a location.

Finally, we define the candidates set

$$C = \{\text{cand}(w_h, L_i) \mid L_i \in M_{\text{probability}}, \max(P_{wh}(L_i)) > \text{threshold}\}$$

Where $\text{cand}(x, y)$ is a function that, given a weekday-hour $x$ and a location $y$, creates a candidate. $\text{threshold}$ is a user-defined $\mathbb{R}$ value in the interval $[0, 1[$.

In other words, for each column in $M_{\text{probability}}$, we take the location with the highest probability and, if that probability is greater than the threshold value, we create a candidate location and time and add it to the set of candidate locations and times.

Once we have $C$ we are ready to share it in a privacy preserving manner.

### 3.3 Privacy Preserving Sharing

We achieve privacy-preserving sharing by encrypting the elements of the candidate sets in such a manner as to guarantee that for a given location both users will have the same encrypted result, or identifier. This allows a user to receive a set of
identifiers and compute the intersection with their own set to detect any overlap in candidates. A simple mapping of encrypted identifier to clear text candidate allows the receiver to know which candidates they have in common.

This way, if a user does not have the encrypted identifier for the encrypted identifiers they receive, they have no way of knowing which location or timeslot the candidate refers to.

We will explain the PSI protocol using Alice and Bob as examples, to make it easier on the reader to follow the sequence of events.

Imagine Alice wants to find a convenient place and time to meet with Bob. We will refer to the local device as Alice’s device and to the remote device as Bob’s device.

As we use public-key cryptography, Bob’s device will have a public key and a private one, so it generates values for \((N, e, d) \mid N \in \mathbb{N}\). The tuple \((N, d)\) is Bob’s device’s private key, and \((N, e)\) is the public key. Bob’s device sends the public key to Alice’s device. After receiving Bob’s public key, Alice’s device generates a random value \(R_1 \in Z_N\), for a finite acyclic group \((Z_N = 1, 2, \ldots, N)\).

We define \(A_i\) as a clear text local candidate from Alice’s device, and \(\alpha_i\) as the encrypted identifier of that candidate. Alice’s device encrypts her set of candidates into a set of encrypted identifiers as follows:

\[
\alpha_i = (H(A_i)R_1^e) \mod N \tag{3.1}
\]

\(H(A)\) is a predetermined hash function that is common to both Alice’s and Bob’s devices. This way, although Bob’s device is able to decrypt the identifiers, the result is a set of salted and hashed values whose original values cannot be inferred.

Alice’s device sends this set of encrypted identifiers to Bob’s device, which, in turn, encrypts Alice’s encrypted identifiers as follows:

\[
\gamma_i = \alpha_i^d \mod N \tag{3.2}
\]

Substituting \(\alpha_i^d\) from Equation 3.2 with the expression given in Equation 3.1 gives:

\[
\gamma_i = H(A_i)^dR_1 \mod N \tag{3.3}
\]

Bob’s device sends the set \(local\), resulting from Equation 3.3, back to Alice’s device.

The next step is for Bob’s device to encrypt its own set of candidates \(B_i\) as following:

\[
\beta_i = H(H(B_i))^d \mod N \tag{3.4}
\]
Bob’s device also sends the set *remote*, resulting from Equation 3.4, to Alice’s device.

Alice’s device performs one last computation on the identifiers in *local*. This is so they can be compared to the identifiers in *remote*. Starting from:

$$local' = \left\{ H \left( \frac{\gamma_i}{R_1} \right) \mid \gamma \in local \right\}$$

(3.5)

And substituting $\gamma_i$ from Equation 3.5 with the value from Equation 3.3 gives as a result:

$$local' = \left\{ H(H(A_i)^d \mod N) \mid \gamma \in local \right\}$$

(3.6)

As we can see, elements $\alpha_i' \in local'$ from Equation 3.6 have the same form as $\beta_i$ in Equation 3.4. The consequence is that if Alice has a candidate $A_i$ and Bob has a candidate $B_i$ sharing the same location and weekday-hour then the encrypted identifiers will be identical.

Alice’s device can compute the intersection $overlap = local' \cap remote$, revealing the encrypted identifiers of all candidates she shares with Bob. Alice’s device then looks up the clear text candidates that generated the encrypted identifiers. This reveals only the places and times Alice has in common with Bob, preserving his privacy by not disclosing what other places he frequents.
Chapter 4

IMPLEMENTATION

In this chapter we describe our implementation. Rather than reproducing snippets of source code, we give a higher level description, explaining the difficulties encountered and the solutions chosen. For in-depth detail we refer the reader to the Application Programming Interface (API) reference in Appendix D or to the source code, which is extensively commented.

While implementing the software we have applied widely followed practices from industry. The goals have been to achieve a tested and documented code base, hopefully clear enough for others to leverage in future work. To increase the chance of successful reuse, we have included steps necessary to configure the development environment and to build the software in Appendix C on page 73.

Our development process has been inspired by the Agile Manifesto [18]. We have used a Version Control System (VCS) for all files involved in the thesis, developed features in small iterations, written tests to avoid regressions between iterations, used real users to get feedback on the User Experience (UX), reused existing libraries where possible— to follow Don’t Repeat Yourself (DRY) principles—and strived to write documentation that is sufficient but not excessive.

Each section in this chapter describes a discrete component of our implementation.

4.1 EventProbability Library

The EventProbability library’s responsibility is to predict likely future locations and times for a user based on a body of historic location activity. Our source of historic location activity is a set of Comma-separated Values (CSV) files that contain anonymised Foursquare check in information scraped from public tweets. The fields available in the CSV file are: user ID, date and time of check in, venue ID, venue latitude, venue longitude and venue name.
Given a user ID, the library’s API allows creating a list of candidate locations and times the user is likely to find themselves at, split into one-hour timeslots from Monday at 00:00 to Sunday at 23:00.

While gathering measurements for Chapter 6, it became apparent that candidate generation is the most costly task for Privacerver. To identify the bottleneck we ran a profiler and, as Figure 4.1 reveals, parsing the dates and times of check-ins was taking up 46.1% of the time.

![Figure 4.1: Results of profiling the EventProbability library](image)

This was a known issue [19] in Android with Joda, the library we use for dealing with dates and times. We fixed the bottleneck with a single line of code, as shown in Listing 4.1

```java
System.setProperty(
    "org.joda.time.DateTimeZone.Provider",
    "org.joda.time.tz.UTCProvider"
);
```

Listing 4.1: Fixing the Joda bottleneck

Next we identified that reading the CSV files had become the bottleneck, consuming 41.2% of the time between reading a line (26.3%) and processing it (11.4% + 1.1%) as can be seen in Figure 4.2.

To solve this bottleneck we would have to migrate to a database-driven source for historical information, but we did not have time to do so and therefore leave it as future work.
4.1.1 EventProbabilityTest Class

This class provides an example use of the EventProbability API and, at the same time, checks that the library is working as expected.

This was the first class to be tested, and due to it not having any dependencies on Android, we wanted the tests to be pure JUnit [20] ones. This would allow the tests to be run without instantiating an Android emulator, resulting in fast execution times. Fast execution times enable tests to be run often, with no slowdown to development, catching errors earlier.

As we use Android Studio as an IDE, the default testing tools are aimed at Android-dependent tests, where an emulator is fired up, taking minutes rather than seconds to execute. We invested significant time and effort to configure the Gradle build system and Android Studio to execute and debug these tests as standalone JUnit ones [21, 22, 23, 24].

The effort more than paid off. In one case, the tests detected the introduction of a subtle off-by-one error and caught it immediately. If we had not developed and set up the tests, this error would have caused hard-to-identify bugs farther down the line.

4.1.2 EventProbability Class

This class, available under src\Privacerver\privacerver\src\main\java\org\maian\tfg\privacerver\libs coordinates the generation of candidate locations, and implements the location prediction algorithm described in Section 3.2. The generateIntermediateMatrix method reads a user’s
Figure 4.3: EventProbability UML class diagram

check ins from the relevant CSV file to generate the intermediate matrix. We optimised this method to read the absolute bare minimum amount of lines from the file, as Input/Output (I/O) is slow.

The generateFinalProbabilityMatrix method uses the information contained in the intermediate probability matrix to construct the final probability matrix. We optimised this method to create the list of candidates at the same time, removing the need to iterate over the matrix a second time.

Figure 4.3 shows the UML class diagram.

4.2 Privacy Preserving Library

This library, available under src\Privacerver\privacerver\src\main\java\com.fatherfinder, implements the PSI protocol described in Section 3.3. It was heavily adapted from De Cristofaro et al.’s Genodroid implementation [17].

The original Genodroid implementation used Bluetooth as a communication layer between devices. Privacerver uses Hyper-text Transfer Protocol (HTTP) messages with JavaScript Object Notation (JSON) payloads, so we had to refactor it. We modeled the HTTP messages in a KeyCandidatesMessage object and separated the code implementing the different encryption steps and message passing into two separate classes with a single responsibility: PSIClient.
by the device acting as the initiator and PSIServer, used by the device acting as the participant.

4.2.1 EncryptionTest

Similarly to the EventProbability class, we wrote tests for the privacy preserving library. Debugging encrypted messages is a complicated task. Isolating the environment to a very limited and well known one, with no network or setup steps involved, allowed us to quickly diagnose and iterate on the adaptation of Genodroid’s original code. As we had already set up JUnit testing, the effort to scaffold this test was measured in minutes, and saved hours by locating a bug in our implementation dealing with Java’s object references in foreach loops.

4.2.2 PSIClient

This class provides an API for acting as the local device during the protocol sequence. The main challenge when implementing this class was separating and encapsulating the steps in the protocol’s sequence. Figure 4.4 shows its UML diagram.

4.2.3 PSIServer

The remote device uses this class’ API during the protocol’s sequence, and is analogous to the PSIClient class. Its UML diagram is in Figure 4.5.
4.3 Battery Measurement Library

This library leverages a service provided by another Android application, PowerTutor [25]. Subsection 5.2.2 gives background on the choice of this service.

As explained in that subsection, we had to reverse engineer the UMLoggerService service API from an older version published on GitHub [26]. The executable .apk file had to be modified to allow Privacerver, an external application, to bind to UMLoggerService. We accomplished that by retrieving the installed executable, decompiling its contents with apktool [27], editing the Android-Manifest.xml file to publish the service and allow external access [28], re-compile into an executable, and finally sign the executable so it could be installed on the device again. We wrote a script to automate this process in case it is needed in the future.

4.3.1 EnergyConsumptionCounter Class

Once the service was available to Privacerver, we implemented this class to query it for Privacerver’s energy consumption, based on available code on GitHub.

This class abstracts away the low-level connection to PowerTutor’s service, providing a simple API to startCounting(), stopCounting() and get-EnergyConsumption().

Its UML class diagram is shown in Figure 4.6.
4.4 Android Application

When developing the Android application we have attempted to follow Android’s User Interface (UI) style[29]. We have kept the design simple, avoiding custom, non-standard widgets so that users can more easily use the application as it offers a familiar UX.

The application is divided into three screens:

**Settings** for configuring the sharing session, e.g. user ID, data file, privacy-preserving, Peer-to-peer (P2P)/Server mode, remote IP.

**Locations** for generating the candidates and filtering out (by swiping to the left or right) sensitive candidates.

**Results** to share the candidates with the remote IP and display the resulting overlapping candidates.

Following Android guidelines, the relatively long-running tasks of generating candidates and sharing candidates are run in background threads, so as to keep the UI responsive. A spinning loading symbol was added, which helped users understand they had to wait for the task to finish before proceeding.

Figure 4.7 shows the three screens. Note how, if the remote IP is not set, the sharing button in (c) is disabled. When the remote IP is set, its value is displayed on the button.

4.4.1 Use cases

This section describes each of the use cases identified when using the Android application. They are:
P2P non privacy-preserving between two users who do not activate privacy preserving and who receive the full candidates list from each other.

P2P privacy-preserving between two users who activate privacy preserving, and only learn the overlapping candidates between them.

Server trusted third party sharing between three or more users who only learn the overlapping candidates between them. The server receives the full candidate lists of all users.

P2P non privacy-preserving sharing

This use case is very straightforward, the user generates candidates in the Locations screen and then shares them with the remote user in the results screen.

Figures 4.8 and 4.9 give a graphical idea of the use case and its sequence diagram.

P2P privacy-preserving sharing

The only change affecting the user is that they have to enable privacy preservation in the settings screen, all the rest is transparent and dealt with by the system. The sequence diagram is considerably more complex, as can be seen in Figure 4.10
Figure 4.8: Use case diagram for P2P, non encrypted sharing

Figure 4.9: Sequence diagram for P2P, non encrypted sharing
Figure 4.10: Sequence diagram for P2P, encrypted sharing. Note the increased complexity compared to Figure 4.9
Server trusted third party sharing

This use case involves the initiating user creating a room. To do so, in the settings screen, they set an empty room name and set the password they want for their room. Candidate generation and filtering is the same. When sharing candidates, the room creator taps the "Create room" button and receives a notification of that room’s name. They then share the room name and password with the rest of users, through e.g. instant messaging.

The rest of users then configure the room name and password in their settings screen, generate their candidates and tap the "Share candidates" button.

Once the room creator, through out-of-band communication, receives confirmation everyone has shared their candidates, she taps the "Close room and retrieve results" button. At that moment, the rest of users will also receive the results.

Requiring out-of-band communication between users is less than ideal, and improvements are suggested for future work in Section 7.4.

We achieved the push-like behaviour for room participants through HTTP long polling, where the server holds on to the response until notified by the room creator to compute the overlapping events.

Figures 4.11 and 4.12 show the use case diagram and sequence diagram, respectively, of this use case.

As mentioned, the server is considered a trusted third party in this scenario. Users only learn of the overlapping candidates, while the server receives the full candidate lists of each user. We were not able to extend the PSI protocol in time for this thesis, and leave it as future work.

4.4.2 Communications

After considering different options for inter-device communications, we chose to use Transmission Control Protocol/Internet Protocol (TCP/IP). The other main contender, Bluetooth, had two show-stopping restrictions: the main one was interoperability between P2P and server modes, as the server implementation would require implementing a separate protocol. Another limitation of Bluetooth is that users would have to be physically close, whereas we did not want physical presence to be a requirement.

Once TCP/IP had been chosen, we had the option of using P2P Wifi connections [30] for P2P mode, but this would, again, have required users to be physically close. We opted for traditional Access Point (AP)-backed wireless networks to avoid this.

For the application layer, we decided on HTTP messages with JSON payloads rather than network sockets. HTTP messages are standardised, with efficient libraries existing for most platforms. This saved us the non-trivial task of
Figure 4.11: Use case diagram for Server trusted third party sharing

Figure 4.12: Sequence diagram for Server trusted third party sharing
designing, implementing and debugging a solution ourselves. For the server we used nanohttpd [31], and for the client we used Apache’s HttpComponents HttpClient [32].

JSON was chosen for the payload as it offers trivial serialisation and deserialisation, through existing libraries. We used Google’s gson library [33].

The main advantage of our decisions was that only minimal changes were necessary to add server connection capabilities to the Android application.

4.5 Server application

To develop the server application we decided to use Java, so as to reuse the Event-Probability library. After considering different lightweight web application frameworks, such as Spark [34], we decided to use Play Framework [35], as the documentation was good and it provided a crucial feature needed for room management.

4.5.1 Password-protected rooms

As Privacerver’s goal is privacy preservation, we realised we had to allow users to create password-protected rooms. Otherwise, third parties with nefarious intentions could join in on the sharing sessions and, with a false list of hundreds of candidates, possibly learn about their victims’ habits.

We implemented rooms using Play framework’s built-in session cache functionality [36]. Passwords are clear text, and not hashed, as this implementation is only a proof of concept. In a real world scenario we would hash them with bcrypt [37], a password hashing function resistant to brute-force search attacks.

In the next chapter we explain how, after implementing the software, we measured the effect of enabling privacy preservation.
Chapter 5

MEASUREMENT

In this chapter we describe how we measured the metrics needed to evaluate the effect of adding privacy preservation to Privacerver.

We were only able to measure the P2P mode between devices, as the PSI algorithm doesn’t support more than two users, and we weren’t able to extend it to support $n$ users in the time frame for this thesis.

5.1 Requirements

To analyse the effects of privacy preservation we split application use into three phases:

1. Generating a list of candidate locations and times on the local device.

2. Sharing and computing the overlap of the generated list between the local and remote devices.

3. Full cycle of generating, sharing and computing the overlap of candidate locations (the sum of both previous phases).

For each phase we measured the time elapsed in nanoseconds and the energy expended in millijoules. Specifically for the sharing phase, we also measured network traffic generated, segmented by bytes transmitted and bytes received.

To gather the results we automated the execution of a full cycle —from generating candidates to sharing them and computing the overlap— with Google’s “Espresso” UI test API [38]. The automated test runs twenty iterations of a full cycle, and writes the measurement results to a CSV file on the device.

The following sections describe all these aspects in more detail.
5.2 Metrics

To measure the effects of the PSI algorithm we determined the static metrics and the dynamic metrics involved. The static metrics are those configured in the settings, and will be used in Chapter 6 to segment the data. They are:

**Mode** Either cleartext, if privacy preservation is not enabled, or encrypted if it is not.

**File** Which data file the user being impersonated is from.

**Local user** The ID of the local user being impersonated.

**Remote user** The ID of the remote user.

The dynamic metrics are those whose value change on each execution of Privacerver and we describe them in the following subsections.

5.2.1 Time

To measure the time elapsed during the different steps of the application we have reused the reliable StopWatch class from the Apache Commons library [39]. Privacerver already uses this library, so no additional dependencies are needed.

We count time elapsed in two data points. The first one, nsTotal, is the number of nanoseconds elapsed while generating candidates, sharing them and computing the common ones between local and remote devices. The second one, nsShare, is the number of nanoseconds elapsed sharing candidates and computing the common ones between local and remote devices. With these two data points we can calculate the time spent generating candidates by subtracting nsShare from nsTotal.

5.2.2 Energy consumption

Measurement of energy consumption was a huge challenge for this thesis. Android’s API only exposes the battery charge percentage, through its BatteryManager class [40]. Such a low resolution of data would have impacted the value of any results obtained, so we regarded it as our last option.

The first option we tried was New Relic’s new mobile Application Performance Monitoring (APM) service [41]. After integrating their libraries and some test reporting, we realised it is for coarser, high volume reports, whereas we were in need of very fine grained measurements.
We found a promising profiler called Trepn [42] that offers power consumption reporting. It requires the device to have a Qualcomm chipset but luckily our devices had them. Upon testing the profiler, though, we couldn’t find the option to activate power measurement as described in the user manual [43, p. 22]. Research into the issue revealed a specific device was required to profile power usage [44], at a cost of $995 and, apparently, with a license that would have forbidden any results being reported in this thesis anyway [45].

We finally came across PowerTutor [25], a profiler resulting from research by Gordon et al. [46] that can report energy consumption of applications in millijoules.

To successfully use it we needed to automate its reporting capabilities, so the automated measurement test described in section 5.4 could gather the necessary information.

As published on the Play Market, PowerTutor doesn’t have any API to retrieve reported values, but luckily the authors published the source code from an old version on GitHub [26]. After approximately thirty hours of reverse engineering, we were able to modify PowerTutor to expose its reporting service to other applications.

Once the service is exposed, and after connecting to it, we retrieve PrivacyServer’s running energy consumption with the code from Listing 5.1, on page 32.

The code is from EnergyConsumptionCounter, the utility class we developed for providing a simple energy consumption counter and whose public API was inspired by the Apache Commons StopWatch class. To accelerate implementation, we didn’t implement StopWatch’s “split” functionality, so rather than measuring total energy consumed and energy consumed while sharing, as we do for time elapsed, we segment by energy consumed while generating candidates and energy consumed by sharing candidates. Total energy consumed is calculated by adding these two together.

5.2.3 Network

Network traffic was much easier to measure. Android’s API offers a TrafficStats class exposing this information. The only caveat was to specify PrivacyServer’s User ID (UID), otherwise the traffic generated by all processes on the device is reported, leading to inaccurate results, as reported by this user [47].

We developed NetworkTrafficCounter, a utility class that uses TrafficStats to provide a simple network traffic counter. We used the Apache Commons StopWatch class’ API as inspiration.
int myUid = android.os.Process.myUid();
try {
    byte[] rawUidInfo = mService.getUidInfo(3,
        mNoUidMask | 0);
    if(rawUidInfo != null) {
        UidInfo[] uidInfos = (UidInfo[])new
            ObjectInputStream(
                new ByteArrayInputStream(rawUidInfo)).readObject
            ();
        double totalEnergyConsumption = 0;

        for(UidInfo uidInfo : uidInfos) {
            // Only calculate energy for this process (privacerver).
            if(uidInfo.uid == myUid) {
                // Add to running total.
                totalEnergyConsumption += uidInfo.
                    totalEnergy;
            }
            else {
                continue;
            }
        }
        return totalEnergyConsumption;
    }
}

Listing 5.1: Retrieving energy consumed by Privacerver from PowerTutor

5.3 Writing Results

Just as important as measuring the metrics is recording their values. We implemented ResultsWriter, a simple utility class for creating a timestamped CSV file with an appendResult() method for adding measurement results as rows to said file.

Listing 5.2 shows a Minimum Working Example (MWE) of a results file. The first line corresponds to the CSV headers and the second line to the measured values for a full cycle.
5.4 Automated Test Running

Automating the gathering of measurements was a moderately difficult task. The initial candidate, Robotium [48], was deemed inadequate.

Privacerver uses background tasks to keep the UI responsive when generating candidates and when sharing them. Robotium has no way of reliably detecting when a background task has finished, and users have to rely on looping idly, checking for changes in the UI that signal the task is finished [49, 50]. The resulting tests are slow due to the idle waiting, and brittle, prone to failing if the check fails or the timeout is reached.

We discovered a new library developed by Google called Espresso. Espresso natively synchronises with Android’s AsyncTasks thread pool [51], solving this issue after a simple refactoring of our code to use AsyncTasks instead of Runnables. The API is also clean and relatively simple to use.

We developed an Espresso test in the usageMeasurementsTest class, incorporating the components from section 5.2. The test runs a full cycle twenty times in a row, recording the measurements at the end of each full cycle.

To improve the quality of the measurements, we only measure the actual work Privacerver performs, by pausing the timers when transitioning between screens and pressing buttons.

5.5 Running the Tests

To run the automated tests we created a “Run Configuration” in Android Studio by following the configuration guide [52].

We ran the test twice: one where privacy preservation was disabled, and one where it was enabled. This was done for two pairs of users known to have overlapping events at given thresholds. The reference user pair in this thesis is “u15354898” and “u23196908”, from the “usersNY50.csv” file. Table 5.1 shows the data for these users when the threshold is set to 0.4.
The test was also run for the user pair “u18649322” and “u18975542” from the “usersUK50.csv” file. Their counts are listed in Table 5.2. The results of the analysis described in Chapter 6 for these users is available in Appendix E on Page 109.

The low thresholds necessary to yield overlaps were a disappointment. It is mainly due to our implementation of the Location objects. Android’s Location API [53] would have permitted distance comparison, allowing events occurring within a certain distance of each other to count towards the same location for threshold satisfaction purposes. This would have resulted in a lower number of higher quality candidates.

Due to an early misunderstanding of the documentation, we thought a “LocationService” had to be developed in order to use Android’s Location class. Time was scarce, so we opted to compare Locations by exact GPS coordinates. We discovered too late that it is not necessary to develop a LocationService to use the Location API [54].

As the goal of this thesis was to implement a PSI protocol and analyse its costs, we didn’t consider the below-expected quality of generated candidates to be a show stopper, but consider it for future work. Possible improvements to generated candidate quality could include replacing our naïve algorithm with that presented by Zhang et al. in [55], which yielded an accuracy of 76% for future predictions, or the algorithm proposed by Sadilek et al. in [9], which can combine text, location and friendship graphs from Twitter to predict a user’s location.

For each test run we configured Privacerver on the test device with the appropriate source data file, user to impersonate, threshold and sharing mode, by enabling or disabling privacy preservation. We also primed the remote device,
configuring it with a known user and threshold that yields overlapping candidate locations and setting the same sharing mode as for the test device.

5.5.1 Results discrepancy

After running the tests, we noticed a marked increase in measurements for the encrypted runs. We initially attributed it to the cost of privacy preservation, but on further inspection it was clear that running the Espresso test back-to-back was causing a performance issue on the second run. Specifically, the time and energy to generate candidates was noticeably higher.

We ran the test with privacy preservation enabled both times to check whether it was, in fact, performance degradation or whether it was the penalty for enabling privacy. The results showed a 19% increase in candidate generation energy consumption between the identical test runs. Figure 5.1 illustrates the performance degradation observed.

![Figure 5.1: Illustration of the performance degradation when running the test back-to-back](image)

(a) Measurements from the first run of the test  
(b) Measurements from the second run of the test

To better analyse the measurements, we reran the privacy-enabled test after a clean reboot of the device.
We suspect that performance degradation has to do with Espresso or Android Studio and the Android Debug Bridge (ADB) connection, rather than Privacerver, as full cycle execution measurements within the same test run are all similar to each other.

In Chapter 6 we analyse the results of the measurements.
Chapter 6

EVALUATION

In this chapter we will analyse the results of the measurements performed in Chapter 5 to evaluate the effect adding privacy preservation has on energy consumption, time elapsed and network traffic. In this chapter, non-privacy preservation is referred to as “clear text” mode, and privacy preservation is referred to as “encrypted” mode.

We have split the Privacerver usage life cycle into the different steps, or phases, between when it is launched and when the user can see the results of computing the convenient places and times to meet up with the rest of users. They are as follows:

**Setup** includes choosing which user to impersonate and what settings to use when sharing, e.g. privacy preserving or not. We will not evaluate this step as there was nothing to measure.

**Generation** is the phase where the historical location data of the chosen user is analysed to construct a list of candidate locations and times that are convenient for that user. We measured this phase and refer to it in this chapter as “Candidate Generation”.

**Results** is the phase where the device first shares the local list of candidate locations and times with the remote device, then receives the remote list of candidate locations and finally calculates which candidates are common to both lists. We measured this phase and refer to it in this chapter as “Sharing and Overlap Calculation”.

**Full cycle** this “meta-phase” treats both previous phases as one, so that we can give a context to the effects of adding privacy preservation.

Now we will evaluate the measurements taken for the three measured phases.
6.1 Candidate Generation

The candidate generation phase is not expected to result in much of a penalty for adding encryption. This is because the code involved does not change between clear text and encryption modes.

In this chapter we have plotted the data with R [56], a statistical analysis program. We use ggplot2’s box plots [57] to succinctly express the quartiles of the data we measured. The bottom and top of each box is the first and third quartiles. The line inside each box is the median, or the second quartile. The whiskers extend to the highest and lowest value within $1.5 \times \text{IQR}$ of the hinge, where $\text{IQR}$ is the distance between the first and third quartiles (Inter Quartile Region). Any data outside of these ranges are plotted as dots, and considered outliers.

As seen in Figure 6.1, the collected data seem to uphold this expectation. Regarding energy consumption, the mean for encrypted mode is within less than one percent of the mean for clear text mode. For time elapsed, the encrypted mode’s mean is within one and-a-half percent of the clear text mode’s mean.

![Box plots for energy consumption and time elapsed](Image)

(a) Energy consumption (b) Time elapsed

Figure 6.1: Candidate generation comparison between clear text and encrypted

To gauge how confident we can be this hypothesis is correct, we will run a Welch two-sample $t$ test in R. `nsGenerate` is the column of each data frame
(encrypted and clear text) that contains time elapsed to generate the candidates for each test execution.

```r
> t.test(encrypted$nsGenerate, cleartext$nsGenerate)

Welch Two Sample t-test
data: encrypted$nsGenerate and cleartext$nsGenerate
t = 0.8978, df = 20.715, p-value = 0.3796
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -757166709 1905969542
sample estimates:
mean of x  mean of y
37515709792 36941308376
```

Listing 6.1: Candidate generation phase $t$ test in R

From Listing 6.1, and choosing the common 5\% level of significance, we can see that $p > 0.05$. Therefore the data support the hypothesis that the elapsed time for candidate generation should not differ between clear text mode and encrypted mode.

A subsequent $t$ test of candidate generation energy consumption resulted in $t(24.45) = 0.7999, p = 0.4315$, supporting the hypothesis that candidate generation energy consumption should also not differ between the two modes.

### 6.2 Sharing and Overlap Calculation

We expect this phase to suffer the penalty from encryption, as this is when PrivacyServer behaves differently according to whether encryption is enabled or not.

The plots in 6.2 show that, indeed, enabling encryption adds a significant penalty to both energy consumption and time elapsed when sharing the list of candidates and then calculating which candidates are common to local and remote devices.

Energy consumption shows a 318\% increase of 2436.2 mJ, while time elapsed is 475\% more, or an extra 3976.22 ms.

As in the previous section, we will run a $t$ test to determine how confident we can be that this increase is statistically probable.
Figure 6.2: Candidate sharing comparison between clear text and encrypted
> t.test(encrypted$nsShare, cleartext$nsShare)

Welch Two Sample t-test

data: encrypted$nsShare and cleartext$nsShare
t = 48.2553, df = 35.227, p-value < 2.2e-16
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval: 3808976621 4143459983
sample estimates:
mean of x  mean of y
4812450361 836232059

Listing 6.2: Sharing phase t test in R

As with Listing 6.1, in Listing 6.2 we use the common confidence level of 5%. Our null hypothesis is that the means should not differ. That is: candidate sharing should take the same time in clear text mode as in encrypted mode. In this case, though, the resulting value of $p < 0.01$ has us reject the null hypothesis. This supports our expectation that enabling encryption will result in this phase taking longer to complete.

A subsequent $t$ test of candidate sharing energy consumption data resulted in $t(30.648) = 20.4, p < 0.01$, confirming the same is true for energy consumption: this phase uses more energy when encryption is enabled.

At first view, these results are disheartening and would seem to suggest that encryption is too resource-intensive for mobile environments. Before accepting that conclusion, though, we should put the data into perspective.

Sharing the candidates in clear text took, on average, 836 ms. That is almost instant from a user’s perspective, and within the one second limit defined by Nielsen [58] where a user’s flow of thought is not interrupted. Sharing in encrypted mode took, on average, 4.8 s. While 5.75\(^1\) times slower, it is still within the ten seconds limit for keeping a user’s attention and is acceptable.

Even after adding context to these increases in energy consumption and time, we will see in Section 6.3 that candidate sharing and overlap calculation is still only a part of Privacerver’s workflow.

\(^1\)N.B. A 475% increase means the final value is the original value (100%) plus the increase (475%) with regards to that value, i.e. 575%.
6.2.1 Network Traffic

This is the phase where data are transmitted over the network. We have plotted a graph to illustrate the penalty incurred by network traffic in Figure 6.3.

Adding encryption results in a 113% increase to total network traffic. When in cleartext mode, total network traffic averaged 20 kB, while in encrypted mode it averaged 42.9 kB. This amount of network traffic is relatively small, as other common mobile activities, such as viewing photos, quickly runs into the hundreds, or thousands, of kilobytes.

Figure 6.3: Network traffic comparison
6.3 Full cycle

If we adopt the holistic view that a full interaction with Privacerver will include both the candidate generation phase and the sharing phase, we can better evaluate the encryption penalty as it relates to this full cycle.

![Comparison of energy consumed during candidate generation vs. candidate sharing](image)

Figure 6.4 and Figure 6.5 show that the sharing phase represents a very small percentage of energy and time compared to the candidate generation phase. In clear text mode, sharing the candidates takes 2.5% of the energy and 2.3% of the time it takes to generate them. In encrypted mode, sharing the candidates requires 10.3% of the energy and 12.8% the time it takes to generate them.

Figure 6.6 shows the final penalties incurred by encryption over the full cycle of candidate generation, candidate sharing and overlapping calculation.

Adding encryption has only increased the total energy consumption by 8%, while the total time has increased 12%.

Given this evaluation, we can conclude that the penalties for adding encryption are significant when considering the sharing phase by itself, but represent quite a minor impact when taking the full interaction cycle into account. The bottleneck
Figure 6.5: Comparison of time elapsed during candidate generation vs. candidate sharing

in this implementation is actually candidate generation, as it exceeds Nielsen’s ten second limit for user interfaces.
Figure 6.6: Full cycle comparison between clear text and encrypted

(a) Energy consumption

(b) Time elapsed
Chapter 7

CONCLUSIONS

For this thesis we successfully developed an Android application, PrivateVer, and a server application, PrivacyServer. PrivateVer allows the sharing of events between \( n \) users to find a convenient meeting place and time for all. When \( n = 2 \) the application works in Peer-to-peer (P2P) mode and allows both privacy-unaware sharing and privacy-preserving sharing. When \( n > 2 \) the application can connect to PrivacyServer to coordinate the sharing effort between the \( n \) users. We have observed that privacy preservation measurably affects time, battery, and network traffic, but within acceptable levels.

The objectives for this thesis have been:

**Software development** Develop the necessary software to generate candidate locations and times for a user, given their history of Foursquare checkins, and sharing that list with other user(s) to obtain the overlapping events, indicating prime candidates for convenient meet ups.

**Privacy preservation** Preserve the privacy of the application’s users when sharing their candidate locations lists. The primary technique has been through the implementation of a Private Set Intersection (PSI) protocol, following the method described by De Cristofaro et al. in their Invention Submission (IS) [12].

**Measurement** Measure the cost of preserving a user’s privacy in time, battery consumption and network traffic.

### 7.1 Software development

The software includes an Android application, a server application written in Java, using the “Play” [35] framework, and reusing the EventProbability libraries from the Android application. The server application allows the creation of
password-protected “rooms”, to isolate sharing of events between different groups of users.

We have written unit tests to help avoid regressions when we have needed to iterate on the libraries due to challenges have cropping up. They have been useful “canaries in the coal mine”, immediately catching bugs that would otherwise have initially gone unnoticed.

We have reverse engineered PowerTutor in order to measure battery consumption, which has involved decompiling the published .apk [25] and modifying its AndroidManifest.xml file to publish and give other processes access to the UMLoggerService service. We were able to interact with this service by leveraging the published (although out of date) source code [26], modifying and adapting the service’s .aidl interface to work with Privacerver. Once connection to the service was established, we wrote additional classes to only report the total power consumption of Privacerver, rather than the whole system, as by default. We configured an automated build system for the reverse engineered version of PowerTutor, to allow easy reproduction of our results in the future.

We have written and tooled integration tests to enable semi-automation of the measurement on time, battery and network. These tests automate the steps of generating a list of candidates and sending the list to the remote device. The manual steps are selecting the user pairs with overlapping events to test and launching the integration test.

7.2 Privacy preservation

Of the four privacy threats defined by Ruiz Vicente, et al. [8], three of them are mitigated by Privacerver in P2P mode with privacy preservation enabled, and are discussed below. The fourth threat, identity privacy, is not considered by this thesis. Successful usage of the software would result in people physically meeting, therefore the identity of all parties is necessarily divulged to a certain extent: at a minimum the user’s physical appearance. This suggests that maintaining identity privacy would be a very difficult problem to solve, if it is indeed solvable.

For the following privacy threats to be mitigated, the user has to be aware of them and be selective with the candidate lists they share.

7.2.1 Location privacy

Location privacy concerns the association of a user with a location at a given time [8, p. 21]. The ability to do so can reveal sensitive information such as substance abuse if, for example, a user were to have regular check-ins at a known
Alcoholics Anonymous venue at the times when the support group’s meetings take place.

Privacerver tackles this threat on two levels. First, thanks to the PSI algorithm, locations not in the set of overlapping candidates are undecipherable to the other parties, which ensures a user’s location privacy for these locations. Second, if the user knows or suspects a sensitive location will be in the set of overlapping candidates, they can remove it from the list of candidates. This avoids it being shared and possibly resulting in a recommended place and time to meet up.

This approach solves the shortcomings described by Ruiz Vicente et al. of simply decreasing the accuracy of the location information, as it is removed from the equation.

7.2.2 Absence privacy

Absence privacy requires protection against the inference of a user’s absence from a location at a given time through the confirmation of a user’s presence at a different location at the same time [8, p. 22].

Similar to location privacy preservation, the only events at risk are those in the list of candidates. A user that has made a habit of turning that publicly announced Wednesday 8PM gym workout routine into a burger at McDonald’s only needs to remove that candidate from the list if they want to keep it private.

7.2.3 Co-location privacy

Co-location privacy threats consist of a third party observing multiple users and inferring presence information either directly about the observed users, or users accompanying the observed users [8, p22].

In this case, the attack is rendered moot. The attacker would need to be a member of the group of people trying to determine a convenient place and time to meet up. Furthermore, to conduct an attack similar to that described by Ruiz Vicente et al., the attacker would have to frequent the same place at the same times as their intended victims, increasing the chances of the attacker already having confirmed the co-location through other means, such as visual.

In any case, removing the sensitive candidate from the list would, as with both previous threats, completely shield the user.

7.3 Measurements

The results we obtained from measuring the three metrics suggest that, while preserving a user’s privacy has measurable costs, they are certainly affordable.
Table 7.1: Privacy preserving penalties as a percentage increase, in isolation and in aggregate

As discussed in Chapter 6, and visible in Figure 7.1, the measurements reveal that sharing the list of candidates with privacy preservation enabled is just under five times slower. Energy consumption is increased threefold. Network traffic is doubled. The important detail is that, initially, these values are quite small.

![Graph showing energy consumption and time comparison between clear text and encrypted modes.](image)

(a) Energy consumption  
(b) Time elapsed

Figure 7.1: Candidate sharing comparison between clear text and encrypted

Energy consumed is initially 31.3 J in clear text mode, and increases to 34.1 J in privacy-preserving mode. Time spent sharing goes from 836 ms, quasi-instant, to 4.8 s, which is still acceptable for the end user. Total network traffic doubles from an average of 20 kilobytes to an average of 40 kilobytes.
Figure 7.2: Comparison of energy consumed during candidate generation vs. candidate sharing
Including the time and energy costs of generating candidate locations, the increase caused by privacy preserving is relatively small, as can be seen in Table 7.1 and Figure 7.2, for the case of energy consumption.

In conclusion, this implementation of privacy preservation should be considered viable. The penalty for preserving privacy is significantly absorbed by taking into account the fixed cost of generating candidates: the process of finding convenient places and times for multiple users to meet up will always include this step and represents the bulk of time and battery consumption.

7.4 Future work

In this section we describe possible future work to expand on what has been accomplished in this thesis.

First and foremost would be to optimise candidate generation. It currently is a very long-running task, taking 37.77 seconds to generate 61 candidates with a 0.4 threshold, which would probably frustrate users in a real world scenario. Improvements could include swapping our naïve algorithm for the ones proposed by Zhang et al. [55] or Sadilek et al. [9].

Second in priority would be to integrate real, live historic data into the application. Currently a static set of text files is used as a source. These text files contain the anonymised check in information for users from New York, San Francisco and the United Kingdom. The information was scraped from public Twitter accounts tweeting each check in. For the implementation to be useful, it would have to interface with the user’s own phone as a source of up-to-date historic location data using a collection of sources, for example Google Location [59] and Foursquare [60].

The next improvement would be extending the PSI algorithm, making it possible for more than two users to preserve privacy while sharing locations. Currently the server is assumed to be a trusted third party. While the clients never have access to other users’ location histories, the owner of the server could easily collect information on all users’ full location histories.

Another improvement would involve giving feedback to each user on the status of the other users, as the sequence of events currently has to be coordinated out of band, e.g. through instant messaging. Feedback would include letting the user know the current stage of the other user(s), e.g. “Other user(s) waiting for you to generate candidates”, or “Waiting for other user(s) to generate candidates”.

The recommender used in this thesis is very basic, using arithmetic averages to select candidate locations and times, discriminating by the hour. As the scope of this thesis was to implement a PSI algorithm and measure its cost, quality of recommendations for the candidates has been a secondary concern. A less naïve
recommender could detect more complex and accurate patterns in a user’s habits using, for example, preceding and succeeding events as context.

Improved networking would also dramatically improve the usability. Discovery of other clients being the major networking improvement as currently users have to manually input the IP address of either the remote device (when in P2P mode) or the server (when sharing with more than one other user). Use of HTTP to transmit messages between devices could also be replaced with sockets, to allow pipelining of communication between devices. The effect would probably be most noticeable when privacy preservation is enabled, allowing the remote device to send its locally encrypted local events to the initiating device at the same time as it’s receiving the initiating device’s candidates.
Bibliography


[19] jodatime - Android Java - Joda Date is slow - Stack Overflow; Available from: http://stackoverflow.com/questions/5059663/android-java-joda-date-is-slow [cited 13/06/14].


[31] nanohttpd; Available from: http://nanohttpd.com/ [cited 14/06/14].


[33] Google. google-gson - A Java library to convert JSON to Java objects and vice-versa - Google Project Hosting; Available from: https://code.google.com/p/google-gson/ [cited 14/06/14].

[34] Spark - A small web framework for Java; Available from: http://www.sparkjava.com/ [cited 04/05/14].

[35] Play Framework - Build Modern & Scalable Web Apps with Java and Scala; Available from: http://www.playframework.com/ [cited 13/05/14].

[36] JavaCache; Available from: http://www.playframework.com/documentation/2.0/JavaCache [cited 05/05/14].


[38] android-test-kit - Google’s Testing Tools For Android - Google Project Hosting; Available from: https://code.google.com/p/android-test-kit/ [cited 09/06/14].

[39] StopWatch (Commons Lang 3.1 API); Available from: http://commons.apache.org/proper/commons-lang/javadocs/api-3.1/org/apache/commons/lang3/time/StopWatch.html [cited 08/06/14].

[40] Monitoring the Battery Level and Charging State | Android Developers; Available from: http://developer.android.com/training/
monitoring-device-state/battery-monitoring.html [cited 09/06/14].


robotium-how-do-i-wait-for-loading-to-complete-before-proceedi

cited 09/06/14].

[51] EspressoSamples - android-test-kit - Advanced Espresso Samples -
wiki/EspressoSamples [cited 09/06/14].

com/p/android-test-kit/wiki/Espresso [cited 09/06/14].

android.com/reference/android/location/Location.
html#Location(java.lang.String) [cited 09/06/14].

[54] Making a Location object in Android with latitude and longitude values -
Stack Overflow:. Available from: http://stackoverflow.com/a/
17983974/19927 [cited 09/06/14].

[55] Zhang R, Price B, Chi M, Walendowski A. Location-based pre-
dictions for personalized contextual services using social net-
work data. ...prediction and pro-active .... 2012;Available from:
http://www.ibr.cs.tu-bs.de/dus-beigl/Awarecast/
awarecast2012_submission_4.pdf.

[56] The R Project. The R Project for Statistical Computing;. Available from:
http://www.r-project.org/ [cited 01/06/2014].

[57] ggplot Project. geom_boxplot. ggplot2 0.9.3.1:. Available from:
http://docs.ggplot2.org/0.9.3.1/geom_boxplot.html [cited 13/06/14].

[58] Nielsen J. Response Time Limits: Article by Jakob Nielsen;. Available from:
http://www.nngroup.com/articles/
response-times-3-important-limits/ [cited 08/06/14].

google.com/locationhistory/b/0 [cited 03/02/14].

05/06/14].
Appendices
Appendix A

PROJECT CHARTER

A.1 Introduction

This document describes and documents at a high level the different aspects required for the Privacy-preserving Event-sharing Android Application and Testing project to be successful.

A.2 Project and product overview

The project’s goals are to implement a privacy-preserving event-sharing algorithm [2] in an Android application —coupled with a server application in certain modes— and to test the effects on battery, data, time and other factors important to such resource sensitive environments.

The event sharing is done by finding overlapping locations and times between users, suggesting those as optimal times-and-places to meet. Privacy is preserved by ensuring only the overlapping events are known to all users involved: a user will not be able to infer another user’s habits from the information exchanged.

A.3 Scope

A.3.1 Objectives

The first objective is to achieve an event sharing Android and server applications. This application will be able to function in two modes: the first being direct event sharing between two devices and the second being event sharing through coordination with a server application between more than two devices.
Included in this first objective is implementation of a privacy-preserving layer to the applications via the algorithm mentioned above [2]. This algorithm is of special interest in the first mode of use, where there is reason to distrust the other device and its intentions with respect to private information. In the second mode of use there exists an implicit expectation for the server to not divulge one user’s private information to another user. The second objective is to analyse the effects of running the privacy preserving algorithm on a smartphone. Effects to measure include the amount of power used by the algorithm, how long it takes to find event overlaps and how much network data is generated in the process.

The third objective is to publish a scientific paper with the results of the analysis and a conclusion with respect to their implications, leaving the door open for future research.

A.3.2 High-level requirements

- Overlapping event sharing between two Android devices.
- Overlapping event sharing between more than two Android devices.
- Encryption of the information used to find overlapping events such as to preserve the privacy of all users involved.
- A quantitative analysis of the effects of finding overlapping events via the application in different contexts.

A.3.3 Major deliverables

Event-sharing Android application  An Android application capable of sharing events with another device, and finding any overlapping ones.

Event-sharing coordinating server application A server application capable of coordinating the sharing of events and finding of overlapping ones between more than two Android devices.

Privacy preserving version For each previous deliverable, a version implementing the privacy-preserving algorithm mentioned [2].

Quantitative analysis of application usage An analysis of the effects on various metrics crucial to the viability of a mobile implementation, such as battery drain, time taken to find overlapping events and network data generated during the process.
A.3.4 Boundaries

The project includes the implementation of the Android and server applications as well as the implementation of the privacy preserving algorithm [2], and a final analysis of the implications of its use on a mobile platform with respect to the limited resources available.

The project does not include research and definition of the privacy preserving algorithm, which has already been conducted [2].

A.4 Duration

A.4.1 Timeline

![Project timeline estimation](image)

Figure A.1: Project timeline estimation

A.4.2 Executive Milestones

A.4.3 State of the Art

There is currently no known alternative to the studied algorithm. There have been studies on the implications of publicly sharing private location events via geosocial networks [3] and also on techniques and strategies on preserving the privacy of users while minimising the impact on the usefulness of these services [4].

The studied algorithm would provide protection against the three privacy threats described in [4].

Location privacy As described in their research, location privacy threats arise from disclosing exact locations to others. The privacy preserving algorithm [2] would only reveal overlaps in two or more users’ location and time data, therefore cancelling this threat in most cases.
<table>
<thead>
<tr>
<th>Milestone</th>
<th>Estimated Completion Timeframe</th>
</tr>
</thead>
<tbody>
<tr>
<td>Implementation of the Android application without privacy preservation</td>
<td>Four weeks after specifications have been collected. Two weeks after (1).</td>
</tr>
<tr>
<td>In P2P mode, between two devices</td>
<td>Four weeks after the server mode without privacy preservation has been implemented. Two weeks after (1).</td>
</tr>
<tr>
<td>In server mode, between more than two devices</td>
<td></td>
</tr>
<tr>
<td>Implementation of the privacy preserving algorithm</td>
<td>Two weeks after implementation of the privacy preserving algorithm in server mode</td>
</tr>
<tr>
<td>In P2P mode</td>
<td></td>
</tr>
<tr>
<td>In server mode</td>
<td></td>
</tr>
<tr>
<td>Quantitative analysis results</td>
<td></td>
</tr>
</tbody>
</table>

Table A.1: Milestones and estimated completion timeframe

**Absence privacy** Due, again, to only sharing overlapping events, this threat is minimised by the algorithm. An adversary would have to be in the same location at the same time as a user to infer that user’s absence from another location.

**Co-Location privacy** This threat is minimised by requiring, as in the case of absence privacy exploiting, for the adversary to be in the same location at the same time as their target to gain any information. It wouldn’t be possible, therefore, to infer a user’s location through the location of a third party also.

### A.5 Assumptions, Constraints and Risks

#### A.5.1 Assumptions

The project assumes historical location and time data will be made available to the Android application, e.g. via Google Location History [5] or Google Calendar [6]. Another option is a dataset containing Foursquare user location history scraped via Twitter.
An important assumption is that the privacy algorithm is already implemented and available via one or more Java libraries.

A.5.2 Constraints

The main constraint of the project is that the application will only be implemented on the Android platform, due to time and resource constraints. Due, again, to resource constraints there will only be two devices on which to test: the author’s two Android devices. There is a possibility of receiving a third device on loan from Universitat Pompeu Fabra. For the same reason, the server hardware will be limited to one unit.

Another constraint is that location sharing will be performed from a historical set of events obtained externally; no field work will be performed.

A.5.3 Risks

The risk of historical location availability may delay project deliverables. A possible mitigation strategy would be to develop sets of mock location and time events for testing purposes.

There also exists a risk the algorithm [2] is too intensive for mobile usage: draining the battery too quickly or taking too long to find overlapping events. Network data amount is not expected to be an issue thanks to modern high-speed mobile networks. A mitigation strategy for the first two issues would be to force all event sharing to go through the server application. Server environments aren’t as resource scarce, and scalability can be attained through elastic computing (AWS/Azure/Heroku).

A.5.4 Project Organisation

Roles and responsibilities

Stakeholders (Internal and External)

The main stakeholders of this project are:

<table>
<thead>
<tr>
<th>Project role</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project director</td>
<td>Joan Melià Seguí</td>
</tr>
<tr>
<td>Project implementer</td>
<td>John Fisher</td>
</tr>
</tbody>
</table>

Table A.2: Milestones and estimated completion timeframe
• Joan Melià, as the project’s advisor.

• Emiliano De Cristofaro, as an external expert in the privacy-preserving algorithm.

• John Fisher, as the author.

A.6 References


A.7 Colophon

This project charter has been adapted from the United States Center for Disease Control’s Unified Process Project Charter Template [1]
Appendix B

GANTT CHART
<table>
<thead>
<tr>
<th>Name</th>
<th>Begin date</th>
<th>End date</th>
</tr>
</thead>
<tbody>
<tr>
<td>Android app without privacy</td>
<td>2/14/14</td>
<td>2/28/14</td>
</tr>
<tr>
<td>Requirements analysis</td>
<td>2/14/14</td>
<td>2/28/14</td>
</tr>
<tr>
<td>Probability matrix generation</td>
<td>2/14/14</td>
<td>2/14/14</td>
</tr>
<tr>
<td>Event filtering</td>
<td>2/19/14</td>
<td>2/19/14</td>
</tr>
<tr>
<td>P2P communication</td>
<td>2/21/14</td>
<td>2/21/14</td>
</tr>
<tr>
<td>Event overlap detection</td>
<td>2/28/14</td>
<td>2/28/14</td>
</tr>
<tr>
<td>Software design</td>
<td>2/14/14</td>
<td>2/28/14</td>
</tr>
<tr>
<td>Probability matrix generation</td>
<td>2/14/14</td>
<td>2/14/14</td>
</tr>
<tr>
<td>Event filtering</td>
<td>2/19/14</td>
<td>2/19/14</td>
</tr>
<tr>
<td>P2P communication</td>
<td>2/21/14</td>
<td>2/21/14</td>
</tr>
<tr>
<td>Event overlap detection</td>
<td>2/28/14</td>
<td>2/28/14</td>
</tr>
<tr>
<td>Implementation</td>
<td>2/14/14</td>
<td>2/28/14</td>
</tr>
<tr>
<td>Probability matrix generation</td>
<td>2/14/14</td>
<td>2/14/14</td>
</tr>
<tr>
<td>Event overlap detection</td>
<td>2/19/14</td>
<td>2/19/14</td>
</tr>
<tr>
<td>P2P communication</td>
<td>2/21/14</td>
<td>2/21/14</td>
</tr>
<tr>
<td>Event overlap detection</td>
<td>2/28/14</td>
<td>2/28/14</td>
</tr>
<tr>
<td>Server app without privacy</td>
<td>3/3/14</td>
<td>3/7/14</td>
</tr>
<tr>
<td>Requirements analysis</td>
<td>3/3/14</td>
<td>3/7/14</td>
</tr>
<tr>
<td>Event matrix reception from clients</td>
<td>3/3/14</td>
<td>3/3/14</td>
</tr>
<tr>
<td>Event overlap detection</td>
<td>3/6/14</td>
<td>3/6/14</td>
</tr>
<tr>
<td>Overlapping event sending to clients</td>
<td>3/7/14</td>
<td>3/7/14</td>
</tr>
<tr>
<td>Software design</td>
<td>3/3/14</td>
<td>3/7/14</td>
</tr>
<tr>
<td>Event matrix reception from clients</td>
<td>3/3/14</td>
<td>3/3/14</td>
</tr>
<tr>
<td>Event overlap detection</td>
<td>3/6/14</td>
<td>3/6/14</td>
</tr>
<tr>
<td>Overlapping event sending to clients</td>
<td>3/7/14</td>
<td>3/7/14</td>
</tr>
<tr>
<td>Implementation</td>
<td>3/3/14</td>
<td>3/7/14</td>
</tr>
<tr>
<td>Event matrix reception from clients</td>
<td>3/3/14</td>
<td>3/3/14</td>
</tr>
<tr>
<td>Event overlap detection</td>
<td>3/6/14</td>
<td>3/6/14</td>
</tr>
<tr>
<td>Overlapping event sending to clients</td>
<td>3/7/14</td>
<td>3/7/14</td>
</tr>
<tr>
<td>Android app with privacy</td>
<td>3/10/14</td>
<td>3/3/14</td>
</tr>
<tr>
<td>Requirements analysis</td>
<td>3/10/14</td>
<td>3/3/14</td>
</tr>
<tr>
<td>Local user event encryption w/ local key</td>
<td>3/10/14</td>
<td>3/10/14</td>
</tr>
<tr>
<td>Remote user event encryption w/ remote</td>
<td>3/19/14</td>
<td>3/20/14</td>
</tr>
<tr>
<td>Encrypted P2P communication</td>
<td>3/28/14</td>
<td>3/28/14</td>
</tr>
<tr>
<td>Encrypted event overlap detection</td>
<td>3/3/14</td>
<td>3/3/14</td>
</tr>
<tr>
<td>Software design</td>
<td>3/10/14</td>
<td>3/3/14</td>
</tr>
<tr>
<td>Local user event encryption w/ local key</td>
<td>3/10/14</td>
<td>3/10/14</td>
</tr>
<tr>
<td>Remote user event encryption w/ remote</td>
<td>3/19/14</td>
<td>3/20/14</td>
</tr>
<tr>
<td>Encrypted P2P communication</td>
<td>3/28/14</td>
<td>3/28/14</td>
</tr>
<tr>
<td>Encrypted event overlap detection</td>
<td>3/3/14</td>
<td>3/3/14</td>
</tr>
<tr>
<td>Implementation</td>
<td>3/10/14</td>
<td>3/3/14</td>
</tr>
<tr>
<td>Local user event encryption w/ local key</td>
<td>3/10/14</td>
<td>3/10/14</td>
</tr>
<tr>
<td>Remote user event encryption w/ remote</td>
<td>3/19/14</td>
<td>3/27/14</td>
</tr>
<tr>
<td>Encrypted P2P communication</td>
<td>3/28/14</td>
<td>3/28/14</td>
</tr>
<tr>
<td>Encrypted event overlap detection</td>
<td>3/3/14</td>
<td>3/3/14</td>
</tr>
<tr>
<td>Server app with privacy</td>
<td>4/1/14</td>
<td>4/1/14</td>
</tr>
<tr>
<td>Check functionality is not affected</td>
<td>4/1/14</td>
<td>4/1/14</td>
</tr>
<tr>
<td>Resource usage testing</td>
<td>4/2/14</td>
<td>4/14/14</td>
</tr>
<tr>
<td>Android P2P without privacy</td>
<td>4/2/14</td>
<td>4/3/14</td>
</tr>
<tr>
<td>Android P2P with privacy</td>
<td>4/4/14</td>
<td>4/7/14</td>
</tr>
<tr>
<td>Server P2P without privacy</td>
<td>4/8/14</td>
<td>4/9/14</td>
</tr>
<tr>
<td>Server P2P with privacy</td>
<td>4/10/14</td>
<td>4/14/14</td>
</tr>
<tr>
<td>Testing results analysis</td>
<td>4/15/14</td>
<td>4/15/14</td>
</tr>
<tr>
<td>Android P2P without vs. with privacy</td>
<td>4/14/14</td>
<td>4/9/14</td>
</tr>
<tr>
<td>Android server without privacy vs. with</td>
<td>4/14/14</td>
<td>4/14/14</td>
</tr>
<tr>
<td>Thesis writing</td>
<td>4/17/14</td>
<td>5/2/14</td>
</tr>
<tr>
<td>Draft</td>
<td>4/17/14</td>
<td>5/2/14</td>
</tr>
</tbody>
</table>

Figure B.1: Gantt chart work breakdown structure
Figure B.2: Gantt chart work breakdown structure diagram
Appendix C

README AND BUILD INFORMATION
Privacy-preserving Event-sharing
Android App Implementation and Testing

The project’s goals are to implement a privacy-preserving event-sharing algorithm in an Android application —coupled with a server application in certain modes— and to test the effects on battery, data, time and other factors important to such resource sensitive environments.

Android app

The Android application has three screens: Settings, Locations and Results.

Settings
- Data file: which events data file will be used. E.g. usersuk50.
- User ID: which user to impersonate from the data file. Care has to be taken to choose a user from the correct file, as otherwise no candidates will be generated. E.g. u18649322.
- Share with IP: the remote device's IP. E.g. 192.168.1.133.
- Probability threshold: which is the minimum probability for a location-time pair to be considered a candidate for sharing. E.g. 0.2.

Local locations

Pressing the "Generate candidate locations" button will populate the screen with a list of location and time (day of week and hour) pairs. These location and time pairs are considered to be the usual location of the user for the given time.

Results
If no remote IP is defined in the settings screen the button is deactivated. Once a remote IP is defined, pressing the "Share with IP xxx.xxx.xxx.xxx" button sends the local locations to the remote device. The remote device sends a response back with its own local locations. The two location lists are cross referenced, and the overlapping location and time pairs are listed on the screen.

For testing sharing between devices, users u18649322 and u18975542 from the UK file have overlapping location-time pairs when the threshold is set to 0.2.

**Building**

The Android app source is located under `src\Privacerver\`.

**Development environment**

The build is tested to work under the following development environment. Newer versions may work but are untested.

- Java SDK 8 for Windows x64
- Android Studio 0.4.2

In the Android SDK Manager:
Tools
- Android SDK Tools v22.3
- Android SDK Platform-tools v19.0.1
- Android SDK Build-tools v19.0.1
- Android 4.4.2 (API 19)
  - SDK Platform v2
  - ARM EABI v7a System Image v2
- Extras
  - Android Support Repository v4
  - Android Support Library v19.0.1
  - Google USB Driver v9 (for debugging on stock Android devices, e.g. CyanogenMod)

Android Studio

Import the project into Android Studio:

1. File -> Import project
   i. Navigate to src
   ii. Select the Privacerver directory
   iii. Click OK

Test the project builds correctly by selecting the unitTest build configuration in the toolbar and then run it by clicking on the green arrow next to the dropdown:

If the build and execution were successful the Run window should display a BUILD SUCCESSFUL message.

Project structure

The three screens are implemented as fragments. The fragments are declared as inner classes of the MainActivity class located at src\Privacerver\Privacerver\src\main\java\org.maiain.tf.g.privacerver\MainActivity.java. This
The locations fragment uses the EventProbability class (in the `FourSquareCandidateLocations` method) to generate the list of candidate locations and times for a given user. This class is located at `src\Privacerver\privacerver\src\main\java\org.maiian.tfg.privacerver\libs\`.

The results fragment sends the list of candidate locations, serialised as JSON to the remote device and receives the remote device’s candidate locations. The intersection of the two lists are calculated and the overlapping candidate locations are displayed to the user.

**EventProbability class**

The event probability class generates a list of candidate locations and times in two steps.

1. First the intermediate matrix is generated. The user’s events are sorted by day of week and hour of day. Each day-hour pair contains a list of all events occurring on that day at that hour.
2. Second the final probability matrix is generated, using the intermediate matrix to calculate the probabilities of the user being at each location for every hour of every day of the week. The
probability is calculated as an arithmetic average of the number of times a user was at the given location vs. the total of events for that day and hour. If the average is higher than the threshold then that location is considered a probable location, or candidate, for that day and hour. A list of these probable locations is generated. This list is what is displayed in the "Locations" screen of the Android application and shared with other devices.

Javadocs for the EventProbability class and the classes it uses is available in the /docs/javadoc directory.
Appendix D

PRIVACERVER API REFERENCE
# Table Of Content

<table>
<thead>
<tr>
<th>Class/Method</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>com.fatherfinder</td>
<td>3</td>
</tr>
<tr>
<td>AbstractPSIProtocol</td>
<td>3</td>
</tr>
<tr>
<td>KeyAndCandidatesMessage</td>
<td>5</td>
</tr>
<tr>
<td>PSIClient</td>
<td>6</td>
</tr>
<tr>
<td>PSIServer</td>
<td>8</td>
</tr>
<tr>
<td>PrivateProtocol</td>
<td>9</td>
</tr>
<tr>
<td>org.maian.tfg.privacerver</td>
<td>11</td>
</tr>
<tr>
<td>MainActivity</td>
<td>11</td>
</tr>
<tr>
<td>MainActivity.LocationFilterFragment</td>
<td>14</td>
</tr>
<tr>
<td>MainActivity.PlaceholderFragment</td>
<td>16</td>
</tr>
<tr>
<td>MainActivity.PreferencesFragment</td>
<td>17</td>
</tr>
<tr>
<td>MainActivity.ResultsFragment</td>
<td>18</td>
</tr>
<tr>
<td>MainActivity.SectionsPagerAdapter</td>
<td>19</td>
</tr>
<tr>
<td>PrivacerverPreferences</td>
<td>20</td>
</tr>
<tr>
<td>PrivacerverPreferences.Modes</td>
<td>23</td>
</tr>
<tr>
<td>org.maian.tfg.privacerver.libs</td>
<td>24</td>
</tr>
<tr>
<td>CandidateLocation</td>
<td>24</td>
</tr>
<tr>
<td>CandidateLocation.TimeSlot</td>
<td>26</td>
</tr>
<tr>
<td>CandidateLocationSharingRequest</td>
<td>28</td>
</tr>
<tr>
<td>CandidateLocationSharingResponse</td>
<td>29</td>
</tr>
<tr>
<td>CandidateLocationSharingResponse.Mode</td>
<td>32</td>
</tr>
<tr>
<td>CandidateLocationSharingResponse.Status</td>
<td>33</td>
</tr>
<tr>
<td>CandidateLocationTimeSlotComparator</td>
<td>35</td>
</tr>
<tr>
<td>Event</td>
<td>36</td>
</tr>
<tr>
<td>EventProbability</td>
<td>37</td>
</tr>
<tr>
<td>IntermediateProbabilityMatrix</td>
<td>39</td>
</tr>
<tr>
<td>Location</td>
<td>40</td>
</tr>
<tr>
<td>ResultsWriter</td>
<td>42</td>
</tr>
<tr>
<td>org.maian.tfg.privacerver.libs.PowerTutor</td>
<td>44</td>
</tr>
<tr>
<td>EnergyConsumptionCounter</td>
<td>44</td>
</tr>
<tr>
<td>org.maian.tfg.privacerver.listeners</td>
<td>46</td>
</tr>
<tr>
<td>CandidateLocationsListener</td>
<td>46</td>
</tr>
<tr>
<td>PowerTutorServiceListener</td>
<td>47</td>
</tr>
<tr>
<td>org.maian.tfg.privacerver.test</td>
<td>49</td>
</tr>
<tr>
<td>EncryptionTest</td>
<td>49</td>
</tr>
<tr>
<td>EventProbabilityTest</td>
<td>50</td>
</tr>
<tr>
<td>LocationTest</td>
<td>51</td>
</tr>
<tr>
<td>org.maian.tfg.privacerver.views</td>
<td>53</td>
</tr>
<tr>
<td>NonSwipeableViewPager</td>
<td>53</td>
</tr>
<tr>
<td>PrivateProtocol</td>
<td>9</td>
</tr>
</tbody>
</table>
### Package com.fatherfinder

#### Class Summary

**AbstractPSIProtocol**

KeyAndCandidatesMessage

Models messages passed between server and client, with a field for the corresponding public key, for the client's encrypted CandidateLocations and an optional field for the server's encrypted CandidateLocations.

**PSIClient**

Provides private set intersections of CandidateLocation lists.

**PSIServer**

Provides server side implementation of private set intersections of CandidateLocation lists.

**PrivateProtocol**

This class does all the work for conducting the secure communication with another user.

---

**Class AbstractPSIProtocol**

```java
java.lang.Object
   ^-- com.fatherfinder.PrivateProtocol
      |-- com.fatherfinder.AbstractPSIProtocol
```

**Direct Known Subclasses:**

- com.fatherfinder.PSIClient, com.fatherfinder.PSIServer

**public abstract class AbstractPSIProtocol extends com.fatherfinder.PrivateProtocol**

**Fields**

- `g` protected `java.math.BigInteger g`

- `p` protected `java.math.BigInteger p`

---

**Constructors**

**AbstractPSIProtocol**

```java
public AbstractPSIProtocol()
```

**Methods**

**hash**

```java
protected java.math.BigInteger hash(java.lang.String input)
```

```java
protected java.math.BigInteger hash(java.math.BigInteger input)
```

**loadSharedKeys**

```java
protected void loadSharedKeys()
```

Overrides:

- `loadSharedKeys` in class `com.fatherfinder.PrivateProtocol`

**loadSharedKeys**

```java
protected void loadSharedKeys()
```

```java
protected void loadSharedKeys() overrides loadSharedKeys in class com.fatherfinder.PrivateProtocol
```

---

83
com.fatherfinder

Class KeyAndCandidatesMessage
java.lang.Object
   +--com.fatherfinder.KeyAndCandidatesMessage

public class KeyAndCandidatesMessage extends java.lang.Object
Models messages passed between server and client, with a field for the corresponding public key, for the client's encrypted CandidateLocations and an optional field for the server's encrypted CandidateLocations.

Fields

key
public java.math.BigInteger key

localCandidatesEncryptedLocallyLocalKey
public java.util.List localCandidatesEncryptedLocallyLocalKey

remoteCandidatesEncryptedRemotelyLocalKey
public java.util.List remoteCandidatesEncryptedRemotelyLocalKey

userId
public java.lang.String userId

Constructors

KeyAndCandidatesMessage
public KeyAndCandidatesMessage(java.lang.String userId, java.math.BigInteger key, java.util.List localCandidatesEncryptedLocallyLocalKey, java.util.List remoteCandidatesEncryptedRemotelyLocalKey)

com.fatherfinder

Class PSIClient
java.lang.Object
   +--com.fatherfinder.PrivateProtocol
      +--com.fatherfinder.AbstractPSIProtocol
         +--com.fatherfinder.PSIClient

public class PSIClient extends com.fatherfinder.AbstractPSIProtocol

Constructors

PSIClient
public PSIClient()

Methods

createKACMRequest
public com.fatherfinder.KeyAndCandidatesMessage createKACMRequest(java.util.List localCandidates)
Encrypts the client's CandidateLocations on the client and builds a KeyAndCandidateMessage object populated with the encryption key and encrypted CandidateLocations, ready for sending to the server. The server will in turn encrypt the resulting encrypted CandidateLocations with its own secret. On receiving the KeyAndCandidateMessage response, the client will do another round of encryption on using the server's public key (provided in the response object). At this point the encrypted client's CandidateLocations can be compared to the encrypted server's locations to find overlaps, and a map is used to retrieve the cleartext CandidateLocation.

Parameters:
- localCandidates - LocalCandidates to encrypt and send to the server.

Returns:
KeyAndCandidateMessage, ready to be sent to the server.
encryptClientCandidatesReceivedFromServerAndMapToCandidateLocations

```java
public java.util.Map
    encryptClientCandidatesReceivedFromServerAndMapToCandidateLocations(com.fatherfinder.KeyAndCandidatesMessage kacmResponse,
    java.util.List localCandidates)
```

Given a KACM response where the server has encrypted the client's CandidateLocations with its secret key, do another round of encryption using the server's public key so the client's CandidateLocations' identifiers are equal to the server's CandidateLocations identifiers in case of overlap.

**Parameters:**
- kacmResponse
- localCandidates

**Returns:**
map of encrypted CandidateLocation identifiers to clear text CandidateLocations

encryptClientCandidatesWithClientPrivateKey

```java
public java.util.List
    encryptClientCandidatesWithClientPrivateKey(java.util.List candidateLocations)
```

Encrypts the client's list of CandidateLocations using the client's private key. This list of encrypted CandidateLocations will be sent to the server for further encryption. A string representation of the CandidateLocation's hashCode is used because it is guaranteed to be unique for a given CandidateLocation. Straight CandidateLocation.toString() only contains the CandidateLocation's friendly name and time slot, so there can be collisions with geographically distinct CandidateLocations with the same friendly name.

**Parameters:**
candidateLocations

**Returns:**

generateClientKeyPair

```java
public void generateClientKeyPair()
```

Generates the client's public (x) and private (rc1) keypair.

-- com. fatherfinder.
Class PSIServer
```
java.lang.Object
| +-- com. fatherfinder.PrivateProtocol
|   +-- com. fatherfinder.AbstractPSIProtocol
|     +-- com. fatherfinder.PSIServer
```

**public class PSIServer**
extends com. fatherfinder. AbstractPSIProtocol


**Constructors**

**PSIServer**

**public PSIServer()**

**Methods**

createKACMResponse

```java
public com.fatherfinder.KeyAndCandidatesMessage
    createKACMResponse(java.lang.String userId,
    java.util.List serverCandidates,
    com.fatherfinder.KeyAndCandidatesMessage kacmRequest)
```

Create a KeyAndCandidatesMessage response to send back to the client. The response contains the server's public key, a list of encrypted client CandidateLocations and a list of server CandidateLocations sent back. This will allow both lists of CandidateLocations to be compared for overlapping elements. Given a KeyAndCandidatesMessage request object with the client's encrypted CandidateLocations and public key, encrypt again with the server's private key. The server's CandidateLocations are also encrypted using the client's public key and added to the response.

**Parameters:**
- userId - the user's ID, used in usageMeasurementsTest
- serverCandidates
- kacmRequest

**Returns:**
encryptServerCandidatesWithClientPublicKey

```java
public java.util.List encryptServerCandidatesWithClientPublicKey(java.util.List candidateLocations, java.math.BigInteger clientKey)
```

Encrypts the server's CandidateLocations with the client's public key. The list of encrypted server CandidateLocations will be sent back to the client, ready to be compared against the client's own list of encrypted CandidateLocations after the client finishes processing it.

**Parameters:**
- candidateLocations
- clientKey

**Returns:**

---

generateServerKeyPair

```java
public void generateServerKeyPair()
```

Generates the server's public (y) and private (rs1) keypair.

---

**Class PrivateProtocol**

java.lang.Object

|-- com.fatherfinder.PrivateProtocol

**Direct Known Subclasses:**
- com.fatherfinder.AbstractPSIProtocol

- public abstract class PrivateProtocol
  
  extends java.lang.Object

  This class does all the work for conducting the secure communication with another user. It does not handle any communication directly, but instead relies on an Object that extends BluetoothService

**Author:**

skyf

---

**Constructors**

PrivateProtocol

```java
public PrivateProtocol()
```
# Package org.maian.tfg.privacerver

## Class Summary

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MainActivity</td>
<td></td>
</tr>
<tr>
<td>MainActivity.LocationFilterFragment</td>
<td>Fragment for displaying list of location candidates to be filtered by user.</td>
</tr>
<tr>
<td>MainActivity.PlaceholderFragment</td>
<td>A placeholder fragment containing a simple view.</td>
</tr>
<tr>
<td>MainActivity.PreferencesFragment</td>
<td>Preferences fragment.</td>
</tr>
<tr>
<td>MainActivity.ResultsFragment</td>
<td>Fragment for sending local candidates, receiving remote ones and outputting the intersection.</td>
</tr>
<tr>
<td>MainActivity.SectionsPagerAdapter</td>
<td>A FragmentPagerAdapter that returns a fragment corresponding to one of the sections/tabs/pages.</td>
</tr>
</tbody>
</table>

## PrivacerverPreferences

Provides access to the user's settings.

## PrivacerverPreferences.Modes

## org.maian.tfg.privacerver

### Class MainActivity

```java
java.lang.Object
  |-- Activity
    |-- org.maian.tfg.privacerver.MainActivity
```

All Implemented Interfaces:
- org.maian.tfg.privacerver.listeners.CandidateLocationsListener
- org.maian.tfg.privacerver.listeners.PowerTutorServiceListener

```java
public class MainActivity
  extends Activity
  implements org.maian.tfg.privacerver.listeners.CandidateLocationsListener,
              org.maian.tfg.privacerver.listeners.PowerTutorServiceListener
```

### Constructors

- `public MainActivity ()`

### Methods

- `public org.maian.tfg.privacerver.libs.PowerTutor.EnergyConsumptionCounter
  getEnergyConsumptionCounter ()
  Returns the EnergyConsumptionCounter object, allowing measurement of energy consumed.
  Returns: the EnergyConsumptionCounter`

- `public java.util.Set
  getLocalCandidateLocations ()

- `public java.util.Set
  getOverlappingLocations ()

- `public java.util.Set
  getRemoteCandidateLocations ()

- `public int
  getRemoteCandidateLocationsSize ()

- `public java.lang.String
  getRemoteUserId ()`

- `public java.util.Set
  getRemoteCandidateLocations ()

- `public java.util.Set
  getRemoteCandidateLocations ()

- `public int
  getRemoteCandidateLocationsSize ()

- `public java.lang.String
  getRemoteUserId ()`
getUid
public int getUid()

onCreate
protected void onCreate(Bundle savedInstanceState)

onCreateOptionsMenu
public boolean onCreateOptionsMenu(Menu menu)

onOptionsItemSelected
public boolean onOptionsItemSelected(MenuItem item)

onTabReselected
public void onTabReselected(ActionBar.Tab tab,
FragmentTransaction fragmentTransaction)

onTabSelected
public void onTabSelected(ActionBar.Tab tab,
FragmentTransaction fragmentTransaction)

onTabUnselected
public void onTabUnselected(ActionBar.Tab tab,
FragmentTransaction fragmentTransaction)

setLocalCandidateLocations
public void setLocalCandidateLocations(java.util.Set candidateLocationsSet)

setOverlappingLocations
public void setOverlappingLocations(java.util.Set overlappingLocations)

setRemoteCandidateLocations
public void setRemoteCandidateLocations(java.util.Set candidateLocationsSet)

setRemoteCandidateLocationsSize
public void setRemoteCandidateLocationsSize(int size)

setRemoteUserId
public void setRemoteUserId(java.lang.String remoteUserId)

org.maian.tfg.privacerver
Class MainActivity.LocationFilterFragment
java.lang.Object
|--Fragment
   |--org.maian.tfg.privacerver.MainActivity.LocationFilterFragment
public static class MainActivity.LocationFilterFragment
extends Fragment
Fragment for displaying list of location candidates to be filtered by user.

Constructors

88
LocationFilterFragment
public LocationFilterFragment()

Methods
fourSquareCandidateLocations
public void fourSquareCandidateLocations(java.lang.String userId)

fourSquareCandidateLocations
public void fourSquareCandidateLocations(java.lang.String userId,
java.lang.String dataFile)

newInstance
public static org.maian.tfg.privacerver.MainActivity.LocationFilterFragment
newInstance(int sectionNumber)
Returns a new instance of this fragment for the given section number.

onAttach
public void onAttach(Activity activity)

onClick
public void onClick(View view)

onCreateView
public View onCreateView (LayoutInflater inflater,
ViewGroup container,
Bundle savedInstanceState)

onPause
public void onPause ()

onResume
public void onResume ()

org.maian.tfg.privacerver
Class MainActivity.PlaceholderFragment
java.lang.Object
|--Fragment
| --org.maian.tfg.privacerver.MainActivity.LocationFilterFragment

public static class MainActivity.PlaceholderFragment
extends Fragment
A placeholder fragment containing a simple view.

Constructors

newInstance
public static org.maian.tfg.privacerver.MainActivity.PlaceholderFragment
newInstance(int sectionNumber)
Returns a new instance of this fragment for the given section number.

ViewHolder
public ViewHolder(View itemView)

Methods

Binder
public void binder()

newInstance
public static org.maian.tfg.privacerver.MainActivity.PlaceholderFragment
newInstance(int sectionNumber)
Returns a new instance of this fragment for the given section number.
onCreateView
public View onCreateView(LayoutInflater inflater,
ViewGroup container,
Bundle savedInstanceState)

org.maian.tfg.privacerver
Class MainActivity.PreferencesFragment
java.lang.Object
|-- PreferenceFragment
|   |-- MainActivity.PreferencesFragment

public static class MainActivity.PreferencesFragment
extends PreferenceFragment
Preferences fragment.

Constructors

PreferencesFragment
public PreferencesFragment()

Methods

newInstance
public static org.maian.tfg.privacerver.MainActivity.PreferencesFragment newInstance(int sectionNumber)
Returns a new instance of this fragment for the given section number.

onCreate
public void onCreate(Bundle savedInstanceState)

onPause
public void onPause()

org.maian.tfg.privacerver
Class MainActivity.ResultsFragment
java.lang.Object
|-- Fragment
|   |-- org.maian.tfg.privacerver.MainActivity.ResultsFragment

public static class MainActivity.ResultsFragment
extends Fragment
Fragment for sending local candidates, receiving remote ones and outputting the intersection.

Constructors

ResultsFragment
public ResultsFragment()

Methods

newInstance
public static org.maian.tfg.privacerver.MainActivity.ResultsFragment newInstance(int sectionNumber)
Returns a new instance of this fragment for the given section number.

onAttach
public void onAttach(Activity activity)

onPause
public void onPause()
public void onClick (View view)

public View onCreateView (LayoutInflater inflater,
ViewGroup container,
Bundle savedInstanceState)

public void onPause ()

public void onResume ()

org.maian.tfg.privacerver

Class MainActivity.SectionsPagerAdapter

public class MainActivity.SectionsPagerAdapter
extends FragmentPagerAdapter

A {link FragmentPagerAdapter} that returns a fragment corresponding to one of the
sections/tabs/pages.

Constructors

SectionsPagerAdapter
public SectionsPagerAdapter (FragmentManager fm)

Methods

count
public int getCount ()

getItem
public Fragment getItem (int position)

getPageTitle
public java.lang.CharSequence getPageTitle (int position)

org.maian.tfg.privacerver

Class PrivaServerPreferences

public class PrivaServerPreferences
extends java.lang.Object

Provides access to the user’s settings.

Fields

KEY_PREF_DATA_FILE
public static final java.lang.String KEY_PREF_DATA_FILE
public boolean privacyEnabled

public java.lang.String remotelp

public java.lang.String roomName

public java.lang.String roomPassword

public final SharedPreferences sharedPreferences

public java.lang.String threshold

public java.lang.String userId

public java.lang.String dataFile

public java.lang.String mode

public void refreshPreferences()
Package org.maian.tfg.privacerver.libs

Class CandidateLocation

CandidateLocation.TimeSlot
Models a candidate location sharing request.

CandidateLocationSharingRequest
Models the JSON response from sharing candidate locations.

CandidateLocationSharingResponse
Models the JSON response from sharing candidate locations.

CandidateLocationSharingResponse.Mode
Sharing modes, with two way enum value <-> String, inspired by nanHTTPD source.

CandidateLocationSharingResponse.Status
Response status codes.

CandidateLocationSharingResponse.Status
Models an event, which occurs at a specific location at a specific moment in time.

Event
Models an event, which occurs at a specific location at a specific moment in time.

EventProbability
Class to generate a user's set of candidate locations and times using a historic list of events.

IntermediateProbabilityMatrix
Models the intermediate probability matrix.

Location
Models a location, identified by its latitude and longitude coordinates and optionally having a descriptive name.

ResultsWriter
Creates a CSV file, writes a header to it and exposes a method to add measurement results rows.

Class CandidateLocation

java.lang.Object
---org.maian.tfg.privacerver.libs.CandidateLocation

public class CandidateLocation
extends java.lang.Object

Fields

P2P
public static final java.lang.String P2P

SERVER
public static final java.lang.String SERVER

Constructors

Modes
public Modes()
Constructors

CandidateLocation

```java
public CandidateLocation(org.maian.tfg.privacerver.libs.Location loc,
int dayOfWeek,
int hour)
```

A candidate location consists of a place the user is probably going to be at during a specific day of the week and hour.

Parameters:
- loc: location
- dayOfWeek: day of the week. Value from the DateTimeConstants class, MONDAY-SUNDAY.
- hour: hour of the day.

Methods

equals

```java
public boolean equals(java.lang.Object obj)
```

Overrides:
equals in class java.lang.Object

toString

```java
public java.lang.String toString()
```

Overrides:
toString in class java.lang.Object

```
94
```

Class CandidateLocation.TimeSlot

```java
public class CandidateLocation.TimeSlot extends java.lang.Object
```

implements java.lang.Comparable

Fields

sDayOfWeek

```java
public final int sDayOfWeek
```

sHour

```java
public final int sHour
```

Constructors

TimeSlot

```java
public TimeSlot(int dayOfWeek,
int hour)
```

Creates a one hour timeslot for a given day of the week.

Parameters:
- dayOfWeek: day of the week. Value from the DateTimeConstants class, MONDAY-SUNDAY.
- hour: hour of the day.
### Methods

**compareTo**

```java
public int compareTo (org.maian.tfg.privacerver.libs.CandidateLocation.TimeSlot other)
```

Comparison between two TimeSlots. Compares the number of hours since Monday at 00 of each TimeSlot.

**Parameters:**
- `other` -

**Returns:**

**equals**

```java
public boolean equals (java.lang.Object obj)
```

Overrides:
- equals in class java.lang.Object

**hashCode**

```java
public int hashCode ()
```

Overrides:
- hashCode in class java.lang.Object

**toString**

```java
public java.lang.String toString ()
```

Overrides:
- toString in class java.lang.Object

---

### Class CandidateLocationSharingRequest

```java
org.maian.tfg.privacerver.libs
```

#### Java

```java
public class CandidateLocationSharingRequest
    extends java.lang.Object
```

Models a candidate location sharing request. If the request is destined for a server, `sRoomName` contains the identifier of the "room" the user is going to share with and receive the final list of overlapping locations from.

### Fields

**sCandidateLocations**

```java
public final java.util.Set sCandidateLocations
```

**sPassword**

```java
public final java.lang.String sPassword
```

**sRoomName**

```java
public final java.lang.String sRoomName
```

### Constructors

```java
public CandidateLocationSharingRequest (java.lang.String roomName,
    java.lang.String password,
    java.util.Set candidateLocations)
```

---

95
Class CandidateLocationSharingResponse

Models the JSON response from sharing candidate locations.

Constructors

CandidateLocationSharingResponse

public CandidateLocationSharingResponse(java.lang.String roomName)

CandidateLocationSharingResponse


CandidateLocationSharingResponse


Methods

getCandidateLocations

public java.util.Set getCandidateLocations()}

getErrorMessage

public java.lang.String getErrorMessage()
getMode
public org.maian.tfg.privacerver.libs.CandidateLocationSharingResponse.Mode
getMode ()

getRemoteUserId
public java.lang.String getRemoteUserId ()

getRoomName
public java.lang.String getRoomName ()

getStatus
public org.maian.tfg.privacerver.libs.CandidateLocationSharingResponse.Status
getStatus ()

setCandidateLocations
public void setCandidateLocations (java.util.Set candidateLocations)

setErrorMessage
public void setErrorMessage (java.lang.String errorMessage)

setRoomName
public void setRoomName (java.lang.String roomName)

setMode
public org.maian.tfg.privacerver.libs.CandidateLocationSharingResponse.Mode
setMode (org.maian.tfg.privacerver.libs.CandidateLocationSharingResponse.Mode status)

org.maian.tfg.privacerver.libs

Class
CandidateLocationSharingResponse.Mode
java.lang.Object
+--java.lang.Enum
+--org.maian.tfg.privacerver.libs.CandidateLocationSharingResponse.Mode
All Implemented Interfaces:
java.io.Serializable, java.lang.Comparable

public static final class CandidateLocationSharingResponse.Mode
extends java.lang.Enum

Sharing modes, with two way enum value <-> String, inspired by nanoHTTPD source.

Fields
P2P
public static final org.maian.tfg.privacerver.libs.CandidateLocationSharingResponse.Mode P2P

SERVER
public static final org.maian.tfg.privacerver.libs.CandidateLocationSharingResponse.Mode SERVER

Methods
fromString
public static
org.maian.tfg.privacerver.libs.CandidateLocationSharingResponse.Mode
fromString (java.lang.String method)
valueOf
public static
org.maian.tfg.privacerver.libs.CandidateLocationSharingResponse.Mode
valueOf(java.lang.String name)

values
public static
org.maian.tfg.privacerver.libs.CandidateLocationSharingResponse.Mode[]
values()
values

public static/org.maian.tfg.privacerver.libs.CandidateLocationSharingResponse.Status[]
values();

org.maian.tfg.privacerver.libs

Class

CandidateLocationTimeSlotComparator

java.lang.Object
t---org.maian.tfg.privacerver.libs.CandidateLocationTimeSlotComparator

All Implemented Interfaces:
java.util.Comparator

public class
CandidateLocationTimeSlotComparator
extends java.lang.Object
implements java.util.Comparator

Constructors

CandidateLocationTimeSlotComparator

public
CandidateLocationTimeSlotComparator();

Methods

compare

public int
compare(org.maian.tfg.privacerver.libs.CandidateLocation
candidateLocation,
org.maian.tfg.privacerver.libs.CandidateLocation
candidateLocation2)

Compares CandidateLocations based on the time they occur. Used to sort into a progression of
the week order, e.g. Monday through Sunday.
Parameters:

  candidateLocation -
  candidateLocation2 -

Returns:

org.maian.tfg.privacerver.libs

Class Event

java.lang.Object
t---org.maian.tfg.privacerver.libs.Event

public final class Event
extends java.lang.Object
implements java.util.Comparator

Models an event, which occurs at a specific location at a specific moment in time.

Constructors

Event

public Event
(org.maian.tfg.privacerver.libs.Location loc,
DateTime dt)

Parameters:

  loc -
  dt -

Methods

getDateTime

public DateTime
getDateTime();

getLocation

public org.maian.tfg.privacerver.libs.Location
getLocation();
Class EventProbability

Class to generate a user's set of candidate locations and times using a historic list of events.

Constructors

EventProbability

public EventProbability(java.io.Reader eventsReader,
java.math.BigDecimal probabilityThreshold,
java.util.Map csvHeaders)

Initialises the EventProbability object.

Parameters:
- eventsReader - a Reader to a source of historic events in CSV format
- probabilityThreshold - minimum threshold to consider a location as highly probable for the
  user's whereabouts. From 0.0 to 1.
- csvHeaders - a map of CSV headers to indices.

Methods

generateFinalProbabilityMatrix

public org.maian.tfg.privacerver.libs.EventProbability
generateFinalProbabilityMatrix()

Generate the final probability matrix, where a value of true for a location row's day and hour
column indicates high probability the user will be at the given location on the given day and hour.
Those locations deemed probable are added to a list of candidate locations.

Returns:
- this, to allow a fluent interface

generateIntermediateMatrix

public org.maian.tfg.privacerver.libs.EventProbability
generateIntermediateMatrix(java.lang.String userId)

Generate the intermediate matrix, which consists of a row for each day of the week and a column
for each hour of the day. Events are sorted into the corresponding day of the week and hour they
occur. The intermediate matrix is later used to calculate probabilities for user location based on
day of the week and hour of the day.

Parameters:
- userId - user to generate the intermediate matrix for

Returns:
- this, to allow for a fluent interface

candidateLocations

public java.util.Set getCandidateLocations()

candidateMatrix

public org.maian.tfg.privacerver.libs.IntermediateProbabilityMatrix
candidateMatrix()

Locations

public java.util.Set getLocations()

Probability Matrix

public java.util.Map getProbabilityMatrix()
Class IntermediateProbabilityMatrix

Models the intermediate probability matrix. The matrix consists of each day of the week, Monday through Sunday, with twenty four hours per day. Each hour contains a list of Events occurring during that time.

Constructors

IntermediateProbabilityMatrix()

Methods

addEvent(Event event)

Adds an event into the appropriate day/hour bag.

Parameters:
- event - the event to be added.

Returns:
- itself with the event added.

getEventsByDay(int dayOfWeek)

Gets all events for a given day, keyed by hour of the day.

Parameters:
- dayOfWeek - the day of the week, from DateTimeConstants.

Returns:
- Events for the given day, keyed by hour of the day.

getEventsByDayAndHour(int dayOfWeek, int hourOfDay)

Returns the list of Events for a given day and hour.

Parameters:
- dayOfWeek - the day of the week, from DateTimeConstants.
- hourOfDay - hour of the day in 24h format (no leading zero).

Returns:

getMatrix()

Returns the events in the week, keyed by day.

Returns:
- events in the week, keyed by day.

Class Location

Models a location, identified by its latitude and longitude coordinates and optionally having a descriptive name. The string representation is latitude:longitude and, being coordinates, serves as a unique identifier for a location.

Constructors

Location(double latitude, double longitude, java.lang.String name)

Public class Location extends java.lang.Object
Methods

**distanceTo**

```java
public double distanceTo(org.maian.tfg.privacerver.libs.Location other)
```


**Parameters:**

other -

**Returns:**

**equals**

```java
public boolean equals(java.lang.Object obj)
```

**Overrides:**

equals in class java.lang.Object

**getName**

```
public java.lang.String getName()
```

**hasName**

```java
public boolean hasName()
```

**hashCode**

```java
public int hashCode()
```

**Overrides:**

hashCode in class java.lang.Object

**toString**

```java
public java.lang.String toString()
```

**Overrides:**

toString in class java.lang.Object

Class ResultsWriter

```
org.maian.tfg.privacerver.libs
```

**public class ResultsWriter**

extends java.lang.Object

Creates a CSV file, writes a header to it and exposes a method to add measurement results rows.

**Constructors**

```
public ResultsWriter(java.lang.String baseDir,
boolean privacyEnabled)
```

**Methods**

```
```
AppendResult

```java
public void appendResult(java.lang.String localUserId,
                         java.lang.String remoteUserId,
                         long generationAndSharingElapsedTime,
                         double generationMjConsumed,
                         long bytesTxed,
                         int localCandidates,
                         int overlappingLocations)
```

Add a row to the results CSV with the given measurements

Parameters:
- localUserId - local user id
- remoteUserId - remote user id
- generationAndSharingElapsedTime - time spent generating candidates, in nanoseconds
- generationMjConsumed - how many millijoules were consumed generating candidate locations
- bytesTxed - bytes transmitted while sharing
- bytesRxed - bytes received while sharing
- localCandidates - number of local candidates
- remoteCandidates - number of remote candidates
- overlappingLocations - number of overlapping locations after sharing candidates

Close

```java
public void close()
```

Package

```
org.maian.tfg.privacerver.libs.PowerTutor
```

Class Summary

```
EnergyConsumptionCounter
Gives access to energy consumption information using PowerTutor's UMLoggerService.
```

```java
public class EnergyConsumptionCounter
extends java.lang.Object
```

```
gives access to energy consumption information using PowerTutor's UMLoggerService. PowerTutor has
to be modified to publish an intent filter for its service (otherwise bindService() will silently fail and bound
will always be false). See ~/PowerTutor-modifying directory (open apk/rebuild.bat, modify with correct
path to android SDK and run to get a modified apk). PowerTutor has to be manually opened and its
profiler started for this class to be able to return energy consumption information.
```

Constructors

```
EnergyConsumptionCounter
public EnergyConsumptionCounter(Context context)
```

```
Context needs to be application context, not Fragment/Activity
(http://stackoverflow.com/a/7885863/19927)
```

```
Parameters:
context - context resulting from getApplicationContext()
```

Methods

```
getEnergyConsumption
public double getEnergyConsumption()
```

103
getTotalEnergyConsumption

public double getTotalEnergyConsumption()

Returns the total amount of energy, in millijoules, that privacerver has consumed since PowerTutor's service was started.

Returns:
mJ consumed by privacerver, or Double.MIN_VALUE if the information isn't available.

startCounting

public void startCounting()

Starts counting the energy consumed by the application.

stopCounting

public void stopCounting()

throws java.lang.IllegalStateException

Sets mStartingEnergy to the energy consumed since startCounting was called and resets the count.

Throws:

java.lang.IllegalStateException - if the counter hadn't been previously been started.

Package org.maian.tfg.privacerver.listeners

Interface Summary

CandidateLocationsListener

Interface to allow communication between fragments and the main activity.

PowerTutorServiceListener

Allows fragments to access PowerTutor's energy measuring service.

org.maian.tfg.privacerver.listeners

Interface CandidateLocationsListener

public interface CandidateLocationsListener

Interface to allow communication between fragments and the main activity.

Methods

getLocalCandidateLocations

public java.util.Set getLocalCandidateLocations()

getOverlappingLocations

public java.util.Set getOverlappingLocations()

getRemoteCandidateLocations

public java.util.Set getRemoteCandidateLocations()

getRemoteCandidateLocationsSize

public int getRemoteCandidateLocationsSize()
getRemoteUserId
public java.lang.String getRemoteUserId()

setLocalCandidateLocations
public void setLocalCandidateLocations(java.util.Set candidateLocationsSet)

setOverlappingLocations
public void setOverlappingLocations(java.util.Set overlappingLocations)

setRemoteCandidateLocations
public void setRemoteCandidateLocations(java.util.Set candidateLocationsSet)

setRemoteCandidateLocationsSize
public void setRemoteCandidateLocationsSize(int size)

setRemoteUserId
public void setRemoteUserId(java.lang.String remoteUserId)

org.maian.tfg.privacerver.listeners

Interface PowerTutorServiceListener

public interface PowerTutorServiceListener

Allows fragments to access PowerTutor's energy measuring service.

Methods
**Package org.maian.tfg.privacerver.test**

**Class Summary**
- **EncryptionTest**
  Tests encryption library.
- **EventProbabilityTest**
  Tests candidate generation library.
- **LocationTest**
  Tests hash override for Location objects works.

**org.maian.tfg.privacerver.test**

**Class EncryptionTest**

```java
public class EncryptionTest
extends java.lang.Object
```

Tests encryption library.

**Constructors**
- **EncryptionTest**
  ```java
  public EncryptionTest ()
  ```

**Methods**
- **setUp**
  ```java
  public void setUp ()
  ```

**org.maian.tfg.privacerver.test**

**Class EventProbabilityTest**

```java
public class EventProbabilityTest
extends java.lang.Object
```

Tests candidate generation library.

**Constructors**
- **EventProbabilityTest**
  ```java
  public EventProbabilityTest
  ```

**Methods**
- **setUp**
  ```java
  public void setUp ()
  ```
### testGenerateIntermediateMatrixCreatesLocations
```java
public void testGenerateIntermediateMatrixCreatesLocations()
```

Tests hash override for Location objects works.

### Constructors

**LocationTest**
```java
public LocationTest()
```

### Methods

**testTwoLocationsWithDifferentCoordinatesAreNotEqual**
```java
public void testTwoLocationsWithDifferentCoordinatesAreNotEqual()
```

**testTwoLocationsWithSameCoordinatesAreEqual**
```java
public void testTwoLocationsWithSameCoordinatesAreEqual()
```

---

**org.maian.tfg.privacerver.test**

### Class LocationTest

```java
public class LocationTest
extends java.lang.Object
```

107
Package org.maian.tfg.privacerver.views

Class Summary
NonSwipeableViewPager
A view pager that doesn't allow swiping between fragments.

org.maian.tfg.privacerver.views

Class NonSwipeableViewPager
java.lang.Object
  |--ViewPager
     |--org.maian.tfg.privacerver.views.NonSwipeableViewPager

public class NonSwipeableViewPager
extends ViewPager
A view pager that doesn't allow swiping between fragments. Used because swiping the candidates list dismisses the candidate location and time, to allow filtering of locations shared. Found at http://stackoverflow.com/questions/9650265/how-do-disable-paging-by-swiping-with-finger-in-viewpager-but-still-

Constructors
NonSwipeableViewPager
public NonSwipeableViewPager(Context context)

NonSwipeableViewPager
public NonSwipeableViewPager(Context context, AttributeSet attrs)

Methods
onInterceptTouchEvent
public boolean onInterceptTouchEvent (MotionEvent arg0)

onTouchEvent
public boolean onTouchEvent (MotionEvent event)
Appendix E

UK MEASUREMENTS FIGURES

This appendix contains the figures resulting from the UK data set’s measurements.
Figure E.1: Candidate generation comparison between clear text and encrypted, UK version of Figure 6.1

(a) Energy consumption     (b) Time elapsed
Figure E.2: Candidate sharing comparison between clear text and encrypted, UK version of Figure 6.2
Figure E.3: Network traffic comparison, UK version of Figure 6.3
Figure E.4: Comparison of energy consumed during candidate generation vs. candidate sharing, UK version of Figure 6.4
Figure E.5: Comparison of time elapsed during candidate generation vs. candidate sharing, UK version of 6.5
Figure E.6: Full cycle comparison between clear text and encrypted, UK version of Figure 6.6

(a) Energy consumption
(b) Time elapsed