

11-10-2015


Data Verifications for Online Social Networks

Mahmudur Rahman

Florida International University, mrahm004@fiu.edu

DOI: 10.25148/etd.FIDC000196

Follow this and additional works at: <https://digitalcommons.fiu.edu/etd>

 Part of the [Information Security Commons](#), [OS and Networks Commons](#), and the [Systems Architecture Commons](#)

Recommended Citation

Rahman, Mahmudur, "Data Verifications for Online Social Networks" (2015). *FIU Electronic Theses and Dissertations*. 2299.
<https://digitalcommons.fiu.edu/etd/2299>

This work is brought to you for free and open access by the University Graduate School at FIU Digital Commons. It has been accepted for inclusion in FIU Electronic Theses and Dissertations by an authorized administrator of FIU Digital Commons. For more information, please contact dcc@fiu.edu.

FLORIDA INTERNATIONAL UNIVERSITY

Miami, Florida

DATA VERIFICATIONS FOR ONLINE SOCIAL NETWORKS

A dissertation submitted in partial fulfillment of the

requirements for the degree of

DOCTOR OF PHILOSOPHY

in

COMPUTER SCIENCE

Mahmudur Rahman

2015

To: Interim Dean Ranu Jung
College of Engineering and Computing

This dissertation, written by Mahmudur Rahman, and entitled Data Verifications for Online Social Networks, having been approved in respect to style and intellectual content, is referred to you for judgment.

We have read this dissertation and recommend that it be approved.

Sundaraja Sitharama Iyengar

B. M. Golam Kibria

Niki Pissinou

Naphtali Rishe

Bogdan Carbunar, Major Professor

Date of Defense: November 10, 2015

The dissertation of Mahmudur Rahman is approved.

Interim Dean Ranu Jung
College of Engineering and Computing

Dean Lakshmi N. Reddi
University Graduate School

Florida International University, 2015

© Copyright 2015 by Mahmudur Rahman

All rights reserved.

DEDICATION

I dedicate this thesis to my parents and my beautiful wife Asha. Without their patience, understanding, support, and most of all love, the completion of this work would not have been possible.

ACKNOWLEDGMENTS

First, I would like to thank my parents for allowing me to realize my own potential. All the support they have provided me over the years was the greatest gift anyone has ever given me. Your love and support means more to me than you will ever know. Also, I would like to thank my advisor, Professor Bogdan Carbutar for supporting me during these past four years. He has given me the freedom to pursue various project ideas without objection. I am also very grateful to him for his scientific advice and knowledge, and many insightful discussions and suggestions with endless patience.

I would also thank all my thesis committee members: Professor Sundaraja Sitharama Iyengar, Professor Naphtali Rishe, Professor Niki Pissinou, and Professor B. M. Golam Kibria for their time taken in reading this dissertation and for their support and valuable feedback.

Special thanks to Dr. Umut Topkara, who has been a great collaborator and teacher. I would also like to thank Dr. Jaime Ballesteros. We all created such an exceptional team and we were able to contribute in several aspects of our research. Their valuable insights and feedback were the key to publish our works. I also want to thank all the present members of the Cyber Security and Privacy Research(CaSPR) Lab.

Finally, I would like to dedicate this work to my wife Sabiha S. Asha. Without your unending support, understanding, and love, I may never have gotten to where I am today. Thank you, my love.

The material in this dissertation is based in part upon work supported by the DoD under grant No. W911NF-13-1-0142 NSF and by NSF under grants NSF CNS 1527153, NSF CNS 1526494, and EAGER 1450619.

ABSTRACT OF THE DISSERTATION
DATA VERIFICATIONS FOR ONLINE SOCIAL NETWORKS

by

Mahmudur Rahman

Florida International University, 2015

Miami, Florida

Professor Bogdan Carbutar, Major Professor

Social networks are popular platforms that simplify user interaction and encourage collaboration. They collect large amounts of media from their users, often reported from mobile devices. The value and impact of social media makes it however an attractive attack target. In this thesis, we focus on the following social media vulnerabilities. First, review centered social networks such as Yelp and Google Play have been shown to be the targets of significant search rank and malware proliferation attacks. Detecting fraudulent behaviors is thus paramount to prevent not only public opinion bias, but also to curb the distribution of malware. Second, the increasing use of mobile visual data in news networks, authentication and banking applications, raises questions of its integrity and credibility. Third, through proof-of-concept implementations, we show that data reported from wearable personal trackers is vulnerable to a wide range of security and privacy attacks, while off-the-shelves security solutions do not port gracefully to the constraints introduced by trackers.

In this thesis we propose novel solutions to address these problems. First, we introduce Marco, a system that leverages the wealth of spatial, temporal and network information gleaned from Yelp, to detect venues whose ratings are impacted by fraudulent reviews. Second, we propose FairPlay, a system that correlates review activities, linguistic and behavioral signals gleaned from longitudinal app data, to

identify not only search rank fraud but also malware in Google Play, the most popular Android app market. Third, we describe Movee, a motion sensor based video liveness verification system, that analyzes the consistency between the motion inferred from the simultaneously and independently captured camera and inertial sensor streams. Finally, we devise SensCrypt, an efficient and secure data storage and communication protocol for affordable and lightweight personal trackers. We provide the correctness and efficacy of our solutions through a detailed theoretic and experimental analysis.

TABLE OF CONTENTS

CHAPTER	PAGE
1. Introduction	1
1.1 Background	1
1.2 Contribution	6
1.2.1 A Novel Solution for Fraudulent Behavior Detection	7
1.2.2 A Comprehensive Solution for Video Liveness Verification	9
1.2.3 A Secure Solution for Data Storage and Communication in Health- centric Devices	12
1.2.4 Toward Preserving Privacy and Functionality in Geosocial Networks	13
1.3 Organization of the dissertation	14
2. Related Work	16
2.1 Related Work in Fraudulent Behavior Detection	16
2.1.1 Research in Detecting Fraudulent Reviews.	16
2.1.2 Research in Sybil Detection.	18
2.1.3 Research in Detecting Deceptive and Malware apps	19
2.2 Related Work in Visual Verifications through Liveness Analysis	20
2.2.1 Research in Video & Acceleration	21
2.2.2 Research in Biometric Liveness.	22
2.2.3 Research in Pose Estimation.	23
2.3 Related Work in Secure Management of Health-centric Devices	24
2.4 Related Work in ensuring GSN Privacy and Safe Cities	26
2.4.1 Research in GSN Privacy	26
2.4.2 Research for Smart and Safe Cities.	28
3. Identifying Deceptive Behaviors in Online Social Networks	31
3.1 Motivation and Challenges	31
3.2 System Model	33
3.2.1 Yelp’s Review System.	33
3.2.2 Influential & Elite Yelpers.	34
3.2.3 Fraudulent Reviews & Deceptive Venues.	34
3.2.4 Yelp Events	34
3.2.5 Android App Market of Google Play	35
3.2.6 Adversarial Model for Google Play Market	35
3.3 Collected Dataset.	36
3.3.1 Collected Yelp Data	36
3.3.2 Collected Google Play Data	41
3.4 Proposed Methods	44
3.4.1 Deceptive Venue Detection.	44
3.4.2 App Fraud Detection	55
3.5 Empirical Evaluation	62

3.5.1	Review Classification in Yelp.	62
3.5.2	Venue Classification in Yelp.	64
3.5.3	Marco in the Wild.	67
3.5.4	Detecting Yelp Campaigns	68
3.5.5	Review Classification in Google Play.	72
3.5.6	App Classification in Google Play	73
3.5.7	FairPlay on the Field.	75
3.5.8	Coercive Campaign Apps in Google Play.	76
3.6	Limitations	76
3.7	Summary	77
4.	Visual Verification through Liveness Analysis	79
4.1	Motivation and Challenges	79
4.2	The Model: System and Adversary	82
4.2.1	System Model	82
4.2.2	Adversary Model	83
4.2.3	A Classification of Mobile Videos	86
4.3	Movee: Solution Overview	86
4.3.1	Video Motion Analysis (VMA)	88
4.3.2	Inertial Sensor Motion Analysis (IMA)	92
4.3.3	Similarity Computation (SC)	94
4.3.4	Classification	100
4.3.5	Vamos: Video Accreditation Through Motion Signatures	101
4.4	Movee Implementation	107
4.5	Data Collection	109
4.5.1	Smartphone Data Collection	109
4.5.2	Google Glass Data Collection	111
4.5.3	YouTube Video Collection	112
4.5.4	Attack Datasets	113
4.6	Evaluation	115
4.6.1	Experiment Setup	115
4.6.2	Server Overhead	116
4.6.3	Movee Attack Detection	117
4.6.4	MoveeG Attack Detection	121
4.6.5	MoveeG Battery Impact	124
4.6.6	Video Dataset Classification	125
4.6.7	CL-Vamos on Motion Categories	126
4.6.8	CL-Vamos on Citizen Journalism	130
4.6.9	Vamos Evaluation	130
4.7	Limitations	133
4.8	Summary	135

5. Secure Management of Data Storage and Communication in Social Sensor Networks	137
5.1 Motivation and Challenges	137
5.2 System Model, Attacker and Background	140
5.2.1 System Model	140
5.2.2 Attacker Model	144
5.2.3 Reverse Engineering Fitbit and Garmin	144
5.2.4 Crypto Tools	148
5.3 Security and Privacy Attacks	149
5.3.1 Vulnerabilities	149
5.3.2 The FitBite and GarMax Tools	150
5.3.3 Attacks and Results	150
5.4 A Protocol for Lightweight Security	155
5.4.1 Solution Requirements	155
5.4.2 Public Key Cryptography: A No Go	156
5.4.3 SensCrypt	156
5.4.4 The SensCrypt Protocol	157
5.5 Analysis	160
5.5.1 SensCrypt Advantages	160
5.5.2 Security Discussion	161
5.5.3 Applications	163
5.6 Evaluation	165
5.6.1 Sens.io: The Platform	165
5.6.2 Tracker: RecordData Overhead	167
5.6.3 Webserver: Storage Overhead	168
5.6.4 Upload: End-to-end Overhead	168
5.6.5 Battery Impact	169
5.7 Limitations	171
5.8 Summary	171
6. Toward Preserving Privacy and Functionality in Geosocial Networks	173
6.1 Motivation and Challenges	173
6.2 Model and Background	175
6.2.1 Location Centric Profiles (LCP)	176
6.2.2 Private LCP Requirements	177
6.2.3 Geosocial Network Attacker Model	179
6.2.4 Safe City System Model	179
6.3 Dataset.	180
6.3.1 Geosocial network data.	180
6.3.2 Crime and Census data.	180
6.4 Proposed Methods	181
6.4.1 Profil _R : A framework for constructing location centric profiles	181
6.4.2 Safety for Geosocial Networks	189

6.5	Empirical Evaluation	202
6.5.1	Evaluation of Profil_R	202
6.5.2	Evaluation Results for iSafe	206
6.6	Limitations	213
6.7	Summary	213
7.	Conclusion and Future Work	214
	BIBLIOGRAPHY	219
	VITA	241

LIST OF TABLES

TABLE	PAGE
3.1 Table of Notations	45
3.2 Features used to classify review R written by user U for venue V	52
3.3 Features used to classify a venue V as either deceptive or legitimate.	54
3.4 Review classification: comparison of machine learning algorithms. RF performs best, at 93.83% accuracy.	63
3.5 Significance test: pairwise comparison of machine learning algorithms using McNemar’s test. With the exception of the (Bagging, RF) pair, for all other pairs McNemar’s test produces a χ^2 value with 1 degree of freedom, highly significant with a confidence level of more than 95.0%.	63
3.6 Marco vs. the three deceptive venue detection strategies of Feng et al. [FXGC12]. Marco shows over 23% accuracy improvement over $dist\Phi$	66
3.7 Marco performance on new, unpopular venues: comparison of machine learning algorithms. RF and DT perform the best.	66
3.8 Collected venues organized by city and venue type. Values between parentheses show the number of venues detected by Marco to be deceptive. San Francisco has the highest percentage of deceptive venues.	68
3.9 Review classification results (10-fold cross-validation), of gold standard fraudulent (positive) and genuine (negative) reviews. MLP achieves the lowest false positive rate (FPR) of 1.47%.	73
3.10 FairPlay classification results (10-fold cross validation) of gold standard fraudulent (positive) and benign apps. RF has lowest FPR, thus desirable [CNW ⁺ 11].	73
3.11 FairPlay classification results (10-fold cross validation) of gold standard malware (positive) and benign apps, significantly outperforming Sarma et al. [SLG ⁺ 12]. FairPlay’s RF achieves 96.11% accuracy at 1.51% FPR.	74
4.1 Video motion categories, based on (i) camera distance to the subject, (ii) the user motion and (iii) camera motion.	85
4.2 Camera motion inference based on cumulative shifts along X and Y axes inferred from the video and inertial sensor streams. The device is considered to be in landscape orientation. X++ (and X- -) denote positive (and negative) X axis shifts that dominate shifts along other axes.	90

4.3	Number of chunks of the free-form dataset, per category. Details in Section 4.6.6.	112
4.4	Detailed accuracy results of Movee on the smartphone random and direction sync attacks. For the random attack, Movee with MLP achieves an accuracy of 92%. For the more effective direction sync attack, Movee using Decision Tree (C4.5) achieves an 84% accuracy.	118
4.5	Detailed accuracy results of Movee on the smartphone cluster and replay attack datasets. For both the cluster and replay attacks, Bagging achieves the best accuracy of 73% and 68% respectively.	119
4.6	Detailed accuracy parameters of Movee Glassware (using different classifiers) for all three attack datasets. “Acc” denotes the accuracy of the classifier.	121
4.7	Detailed accuracy results of MoveeG on the Glass cluster and replay attack datasets. Random Forest achieves best accuracy for both cluster and replay attacks. Movee is more accurate on Glass than on smartphone.	123
4.8	CL-Vamos accuracy on cluster attack is as high as 85% (on category 11.)	128
4.9	CL-Vamos accuracy on sandwich attack ranges from 68% to 88%.	129
4.10	Vamos efficacy on cluster based stitch attack. The classifier approach performs best.	132
4.11	Vamos performance on sandwich/stitch attack. The classifier approach performs best.	132
5.1	Types of data harvested by FitBite and GarMax from Fitbit and Garmin. Garmin provides GPS tagged fitness information, which GarMax is able to collect.	150
5.2	Symbol definitions.	157
5.3	Comparison of defenses provided by SensCrypt and FitCrypt against the types of attacks described in Section 5.3.3 when the adversary has a combination of the capabilities described in Section 5.2.2. Each element in the table describes which attacks are thwarted by the corresponding solution.	161
5.4	SensCrypt applicability: fitness trackers, home monitoring solutions. . .	164
5.5	RecordData: computation overhead in ms. FitCrypt-RSA 2048 bit is not viable on Arduino (2.3s). FitCrypt-ECC 224 bit (equivalent of RSA 2048 bit) is even less efficient. SensCrypt is 2-3 orders of magnitude more efficient.	167

5.6	Upload: comparison of tracker, webservice and communication delays (shown in ms) of SensCrypt and FitCrypt. FitCrypt (RSA or ECC) is shown both for the Fitbit (96KB) and Garmin (1MB) memory size. The delay of SensCrypt is independent of <i>mem</i> size, and significantly shorter.	169
6.1	Crime weight assignment using the FCPC.	190
6.2	Error measurement data for ARIMA, LES and ANN. Figures reference to the main document.	208

LIST OF FIGURES

FIGURE	PAGE	
3.1	System overview of Marco. Marco relies on social, temporal and spatial signals gleaned from Yelp, to extract novel features. The features are used by the venue classifier module to label venues (deceptive vs. legitimate) based on the collected data. Section 3.4 describes Marco in detail.	32
3.2	YCrawl system architecture. YCrawl relies on a pool of servers and proxies to issue requests. The scheduler relies on a request queue to ensure there are no loops in the crawling process.	36
3.3	Timelines of positive reviews of 3 deceptive venues (see Section 3.3.1). Each venue has several significant spikes in its number of daily positive reviews.	48
3.4	Evolution in time of the average rating of the venue “Azure Nail & Waxing Studio” of Chicago, IL, compared against the ratings assigned by its reviews. The values in parentheses denote the number of reviews that were assigned a corresponding rating (shown on the y axis) during one day. The lack of consensus between the many low and high rated reviews raises a red flag.	50
3.5	ROC plot of Random Forest (RF), Bagging and C4.5 Decision Tree (DT) for review classification (426 fraudulent, 410 genuine). RF performs best, at 93.83% accuracy.	53
3.6	(a) Distribution of reviewers’ review count: fraudulent vs. genuine review sets. (b) Distribution of reviewers’ expertise levels: fraudulent vs. genuine sets. Note their symmetry: unlike genuine reviewers, fraudulent reviewers tend to have written only few reviews and have low expertise for the venues that they reviewed.	53
3.7	FairPlay system architecture. The CoReG module identifies suspicious, time related co-review behaviors. The RF module uses linguistic tools to detect suspicious behaviors reported by genuine reviews. The IRR module uses behavioral information to detect suspicious apps. The JH module identifies permission ramps to pinpoint possible Jekyll-Hyde app transitions.	57
3.8	Timelines of positive reviews for 3 apps from the fraudulent app dataset.	59
3.9	Mosaic plot of install vs. rating count relations of the 87K apps. Larger rectangles signify that more apps have the corresponding rating and install count range; dotted lines mean no apps in a certain install/rating category. The standardized residuals identify the cells that contribute the most to the χ^2 test. The most significant rating:install ratio is 1:100.	59

3.10	(a) Apks detected as suspicious (y axis) by multiple anti-virus tools (x axis), through VirusTotal [Vir15], from a set of 7,756 downloaded apks. (b) Distribution of the number of “dangerous” permissions requested by malware, fraudulent and benign apps. (c) Dangerous permission ramp during version updates for a sample app “com.battery.plusfree”. Originally the app requested no dangerous permissions.	61
3.11	ROC plot of RF, Bagging and C4.5 DT for the 90 deceptive/100 legitimate venue datasets. RF and DT are tied for best accuracy, of 95.8%.	65
3.12	Distribution of SC(V), for the 90 deceptive and 100 legitimate venues. 60 deceptive venues have at least one review spike. 1 legitimate venue has 1 spike.	65
3.13	(a) Marco’s per-module overhead: FRI is the most expensive, but under 2.3s even for venues with 500 reviews. (b) Zoom-in of FRI module overhead. Computing the $Exp_U(V)$ feature takes the most time. . .	67
3.14	(a) The timeline of “Pink Taco 2” (Los Angeles) and of the Yelp event for this venue. Note the correlation between the two. (b) Yelp events: Positive review spike count as a function of ΔT	69
3.15	(a) Distribution of the short term impact (2 weeks) of Yelp events on venue ratings. (b) Yelp events: Distribution of the improvement due to Elite events.	70
3.16	Mosaic plots: The standardized residuals indicate the importance of the rectangle in the χ^2 test. (a) The dependency between the short term rating change of venues due to events and their number of reviews. (b) The dependency between the long term rating change of venues due to events and their number of reviews.	71
4.1	Snapshot of Citizen Evidence Lab [Cit] training session exercise. It consists of steps to verify the source of the video (i.e., account of uploader, upload time) and its content (e.g., clothes, accents, flags, landmarks).	81
4.2	Movee uses four modules to verify a video stream: the i) Video Motion Analysis (VMA), and the ii) Inertial Sensor Motion Analysis (IMA), produce movement estimations during capture, iii) Similarity Computation extracts features, which iv) Classification uses to make the final decision.	82
4.3	The Video Motion Analysis module processes each consecutive video frame and finds the motion vector by computing the amount of displacement that common image components have shifted between two frames.	87

4.4	(a) Raw Accelerometer Data. (b) Filtered Accelerometer Data. The Y axis is the dominant axis for the direction and orientation of the device.	91
4.5	Illustration of DTW alignment for two time-dependent sequences. The red dots show the optimal warping path. A diagonal (match) move is a match between the two sequences. An expansion duplicates one point of one sequence and a contraction eliminates one of the points.	95
4.6	Example alignment of video and inertial motion streams extracted from the same experiment: (a) when using only DTW. (b) when stretching the shorter vector and applying DTW. (c) after stretching and calibration and applying DTW.	96
4.7	Effect of calibration in the similarity computation. (a) No calibration. (b) Truncated mean calibration. (c) Polynomial curve fitting calibration.	99
4.8	Illustration of the Vamos architecture and operation. Vamos consists of three steps, (i) “chunking”, to divide the (video, acceleration) sample, (ii) chunk level classification, and (iii) sample level classification.	101
4.9	Chunk extraction illustration. For segment based chunking, the first segment produces a single usable chunk. For random chunking, chunk 3 overlaps both chunks 1 and 2.	103
4.10	Movee in action on smartphone: Target icon (bullseye) at the bottom of the screen shows the direction in which the user needs to move the camera.	107
4.11	Movee on Google Glass: Top snapshot shows view with Google Glass. Bottom snapshot shows view from Movee on Glass perspective. Target icon (bullseye) at the right of the screen shows the direction in which the user needs to move the head mounted glass.	108
4.12	Stitch attack example. For a genuine sample of 3 chunks, the attacker produces 3 fake samples, with 1 to 3 fake (red) chunks. The genuine chunks are copied from the genuine sample.	114
4.13	Movee (per-module) server side overhead: video processing is the most expensive. The total cost is however under 1.3s.	117
4.14	Summary of Movee accuracy on smartphone random, direction sync, cluster and replay attack datasets. The accuracy (y axis) labels exceed 100% to fit the legend.	118
4.15	Smartphone data evaluation. (a) ROC curve on random dataset for Movee (using MLP). (b) Impact of SC steps on Movee’s accuracy for the random attack. (c) Impact of SC steps on Movee’s accuracy for the direction sync attack.	120

4.16	(a) ROC curve (using MLP) on random attack dataset for MoveeG. (b) The impact of SC steps on the MoveeG accuracy for the random attack. (c) The impact of SC steps on the MoveeG accuracy for the direction sync attack.	122
4.17	Summary of Movee accuracy on data collected from Google Glass, when different classifiers are used.	124
4.18	Impact of video and acceleration recording on Google Glass battery lifetime. The video recording activity halves the battery lifetime when compared to the acceleration recording activity.	125
4.19	(a) Motion category distribution for YouTube dataset. (b) Distribution for free-form dataset. Table 4.1 defines the 12 categories.	126
4.20	Accuracy of CL-Vamos and Movee. CL-Vamos improves by more than 15% on Movee, for both cluster and sandwich attacks.	127
4.21	Setup of Vamos experiment. Each genuine fold produces a stitch attack fold. In each experiment, 9 genuine folds and the corresponding stitch attack folds are used for training. The rest are used for testing.	131
4.22	Vamos accuracy on stitch attacks. Even for the sandwich stitch attack, the classifier approach (using Bagging) exceeds 93% accuracy. . . .	133
5.1	System components: (a) Fitbit: trackers (one cradled on the base), the base (arrow indicated), and a user laptop. The arrow pointing to the tracker shows the switch button, allowing the user to display various fitness data. (b) Garmin: trackers (the watch), the base (arrow indicated), and a user laptop.	140
5.2	Fitbit Upload protocol. Enables the tracker to upload its collected sensor data to the user’s social networking account on the webserver. SensCrypt’s Upload protocol extends this protocol, see Section 5.4. .	147
5.3	Fitbit service logs: Proof of login credentials sent in cleartext in a HTTP POST request sent from the base to the webserver.	149
5.4	TPDC outcome on Garmin: the attacker retrieves the user’s exercise circuit on a map (shown in red on the right side), based on individual fitness data records (shown on the left in XML format). The data record on the left includes both GPS coordinates, heart rate, speed and cadence.	151
5.5	Outcome of Tracker Injection (TI) attack on Fitbit tracker: The daily step count is unreasonably high (167,116 steps).	152
5.6	Snapshot of Fitbit user account data injection attack. In addition to earning undeserved badges (e.g., the “Top Daily Step”), it enables insiders to accumulate points and receive financial rewards through sites like Earndit [Ear].	153

5.7	Battery drain for three operation modes. The attack mode drains the battery around 21 times faster than the 1 day upload mode and 5.63 times faster than the 15 mins upload mode.	154
5.8	Example SensCrypt tracker memory (mem). Light green denotes “clean”, unwritten areas. Red denotes areas that encode tracker sensor data. (a) After (i-1) records have been written. The ctr is 1. (b) After Upload occurs at the state in (a). The ctr becomes 2, to enable the creation of fresh PRNs, overwritten on the former red area. (c) After n-i+2 more records have been written from state (b), leading to the clean pointer cycling over from the start of the memory. (d) After Upload occurs at the state in (c).	158
5.9	Testbed for SensCrypt. Sens.io is the Arduino Uno device equipped with Bluetooth shield and SD card is the tracker. Nexus 4 is the base. . .	165
5.10	SensCrypt architecture. The tracker relies on locally stored key K_T to authenticate webserver messages and encode sensor data. The webserver manages the Map structure, to authenticate and decrypt tracker reports.	166
5.11	Battery lifetime for 9V cell powered Sens.io device in four scenarios: Baseline, Fitbit, SensCrypt and FitCrypt-RSA-256. The last three scenarios record sensor data every 2s. The Baseline scenario measures the battery lifetime of Arduino device with no functionality. SensCrypt reduces 13% of the battery lifetime over Fitbit’s operation. Even a vulnerable FitCrypt-RSA-256 reduces the battery lifetime to half of SensCrypt.	170
6.1	Solution architecture ($k=2$). The red arrows denote anonymous communication channels, whereas black arrows indicate authenticated (and secure) communication channels.	177
6.2	1 week (July 13-19, 2011) evolution of the number of crimes reported within one Miami-Dade block.	181
6.3	Safety index illustration for the Miami-Dade county: $SI(B, \Delta T)$ values are mapped into color-coded “safety levels”: the higher the level, the safer the block.	192
6.4	Snapshots of iSafe on Android.	201
6.5	iSafe browser plugin overhead: Collecting reviews from venues, as a function of the number of reviews.	202
6.6	<i>Setup</i> dependence on Benaloh modulus size. Note the significant increase to 13.5s for a 2048 bit modulus. This cost is however amortized over multiple check-in executions.	203

6.7	The overhead imposed by ZK-CTR as a function of the Benaloh modulus size. Note the significant overhead increase for a 2048-bit modulus, of approximately 260ms per ZK-CTR round.	203
6.8	The overhead of the ZK-CTR protocol as a function of the number of proof rounds. The linear increase in the number of rounds leads to a 12s overhead for 100 rounds. 100 rounds reduce however the probability of client cheating to an insignificant value, 2^{-100}	204
6.9	Storage and communication overhead (in KB) as a function of b, the number of sub-intervals considered in the statistics computation. Even for b=20, the storage overhead is only 5KB and the communication is 17KB.	206
6.10	Crime Forecasting experiments in Miami-Dade: (a) Prediction of assaults. (b) Prediction of robberies. (c) Prediction of assaults in a given block.	207
6.11	(a) Distribution of block crime index values in the Miami-Dade county. (b) Evolution in time of the SI value of a Miami-Dade block and the average SI values of Yelp users that visited the block.	210
6.12	Distribution of safety index values of Yelp users.	210
6.13	Android iSafe overhead. (a) Secret share generation and secret reconstruction time overhead. (b) iSafe communication overhead. (c) iSafe total communication size.	212

CHAPTER 1

INTRODUCTION

1.1 Background

Social networks are popular infrastructures for communication, interaction, and information sharing on the Internet with friends and the world. Popular social networks such as Facebook, Yelp, Foursquare, Youtube provide communication, storage and social applications for hundreds of millions of users. Users join, establish social links to friends, and leverage their social links to share reviews, content (video, image, etc.), organize events, and search for specific users or shared resources. These social networks provide platforms for organizing events, user to user communication, and are among the Internets most popular destinations.

Online reviews are central to numerous aspects of people’s daily online and physical activities. Which Thai restaurant has good food? Which mover is reliable? Which mechanic is trustworthy? People rely on online reviews to make decisions on purchases, services and opinions, among others. People assume these reviews are written by real patrons of venues and services, who are sharing their honest opinions about what they have experienced. But, is that really the case? Unfortunately, no. Reviews are sometimes fake, written by fraudsters who collude to write glowing reviews for what might otherwise be mediocre services or venues [Seg11, JL08, JLL10, LNJ+10]. In this paper, we at first focus on Yelp [Yela], a popular social networking and location based service that exploits crowdsourcing to collect a wealth of peer reviews concerning venues and services. Crowdsourcing has however exposed Yelp to significant malicious behaviors: Up to 25% of its reviews may be fraudulent [Yelb]. This behavior is not limited to occasional, inexperienced fraudsters, but may be well-organized. Search engine optimization (SEO)

companies tap into review writer markets (see e.g., [Spo13, Pos13, Pay13]) to organize *review campaigns*, “face lift” operations for paying business owners [Att]. A review campaign consists of posting multiple reviews for a target venue, with the goal of altering its (1-5 star) rating. For business owners, profit seems to be the main incentive to drive them to engage in deceptive activities. Studies have shown that in Yelp, an extra half-star rating causes restaurants to sell out 19% more frequently [AM12], and a one-star increase leads to a 5 to 9% increase in revenue [Luc]. In this work, we seek to detect deceptive venues whose ratings are impacted by fraudulent reviews. Furthermore, we study the impact of Yelp “elite” events on the ratings of hosting venues. Elite events are organized by Yelp for the benefit of “Elite”, influential users, who write popular reviews. Yelp attempts to prevent review “unfairness” by encouraging attendees to review the event instead of the venue. However, the ample warning offered to hosts, coupled with the inability of users to accurately follow directions, may be used by adversaries to transform Yelp events into review campaign tools.

Crowdsourcing imposed fraudulent behavior is also common in online social app markets. The commercial success of Android app markets such as Google Play [Goo] has made them a lucrative medium for committing fraud and malice. Some fraudulent developers deceptively boost the search ranks and popularity of their apps (e.g., through fake reviews and bogus installation counts) [Sie14], while malicious developers use app markets as a launch pad for their malware [Min14, Mlo14, Rob15, Gre14]. The motivation for such behaviors is impact, as increased popularity leads to financial benefits and simplifies malware proliferation. Existing mobile malware detection solutions have limitations. For instance, while Google Play uses the Bouncer system [OM12] to remove malware, out of the 7,756 Google Play apps we analyzed using VirusTotal [Vir15], 12% (948) were flagged by at least one

anti-virus tool and 2% (150) were identified as malware by at least 10 tools (see Figure 3.10(a)). Previous work has focused on dynamic analysis of app executables [BZNT11, SKE⁺12, GZZ⁺12] as well as static analysis of code and permissions [SLG⁺12, PGS⁺12, YSM14]. However, recent Android malware analysis revealed that malware evolves quickly to bypass anti-virus tools [ZJ12]. In this dissertation, we seek to identify both malware and search rank fraud targets in Google Play. This combination is not arbitrary: we posit that malicious developers resort to search rank fraud to boost the impact of their malware. Unlike existing solutions, we build this work on our observation that fraudulent and malicious behaviors leave behind telltale signs on app markets. We uncover these nefarious acts by picking out such trails.

While we address the challenges posed by well-organized malicious behavior activities seen in the review centric social networks and app markets, we also research privacy vulnerabilities of users in geosocial networks. Online social networks have become a significant source of personal information. Their users voluntarily reveal a wealth of personal data, including age, gender, contact information, preferences and status updates. A recent addition to this space, geosocial networks (GSNs) such as Yelp [Yela] and Foursquare [fou] further collect fine grained location information, through *check-ins* performed by users at visited venues. Overtly, personal information allows GSN providers to offer a variety of applications, including personalized recommendations and targeted advertising, and venue owners to promote their businesses through spatio-temporal incentives, e.g., rewarding frequent customers through accumulated badges. Providing personal information exposes however users to significant risks, as social networks have been shown to leak [KW10] and even sell [SF] user data to third parties. There exists therefore a conflict. Without privacy people may be reluctant to use geosocial networks; without user informa-

tion the provider and venues cannot support applications and have no incentive to participate. In this dissertation, we take first steps toward addressing this conflict. Our approach is based on the concept of *location centric profiles* (LCPs). LCPs are statistics built from the profiles of (i) users that have visited a certain location or (ii) a set of co-located users.

Providing privacy preserving functionality in geosocial networks enable us to envision a system where users are seamlessly made aware of their safety in a personalized manner, through quotidian experiences such as navigation, mobile authentication, choosing a restaurant or finding a place to live. We propose to achieve this vision by introducing a framework for defining public safety. Intuitively, public safety aims to answer the question “Will location L present any danger for user A when she visits L at a future time T ”? An important challenge to achieving this vision is the need to properly understand and define safety. While safety is naturally location dependent, it is also inherently volatile. It not only exhibits temporal patterns (e.g., function of the season, day of week or time of day) but also depends on the current *context* (e.g., people present, their profile and behavior). Furthermore, as suggested by the above question, public safety has a personal dimension: users of different backgrounds are likely to be impacted differently by the same location/time context. Previous attempts to make people safety-aware include the use of social media to distribute information about unreported crimes [FAdO⁺10], or web based applications for visualizing unsafe areas [Cri, Gua]. The main drawbacks of these solutions stem from the difficulty of modeling safety and of integrating it in quotidian user experiences. Instead, in this dissertation we investigate the combination of space and time indexed crime datasets, with mobile technologies and online social networks to provide personalized and context aware safety recommendations for mobile and social network users. To achieve this, we first define location centric,

static crime and safety metrics, based on recorded crime events. Given observed crime periodicities, we show that time series forecasting tools are able to predict future crime and safety index values of locations, based on past crime events.

Recent times have seen the importance of new kinds of social networks: content based social networks and social sensor networks. The ubiquitous and connected nature of camera-equipped mobile devices has greatly increased the value and importance of visual information they capture and also the personal private and health centric data they store inside the wearable devices. Mobile apps utilize mobile and wearable device cameras for purposes varying from authentication to location verification, tracking, witnessing, and remote assistance. Today, broadcasting videos from camera phones uploaded by unknown users is admissible on news networks, and banking customers expect to be able to deposit checks using mobile devices. We address the fundamental question of whether the visual stream uploaded by a user has been captured live on a mobile device, and has not been tampered with by a malicious user attempting to game the system. We refer to this problem as video “liveness” verification. We exploit the observation that for plagiarized videos, the motion encoded in the video stream is likely inconsistent with the motion from the inertial sensor streams (e.g., accelerometer) of the device. This problem is a cornerstone in a variety of practical applications that use the mobile device camera as a trusted witness. Examples applications include citizen journalism, where people record witnessed events (e.g., public protests, natural or man-made disasters) and share their records with the community at large. Other applications include video based proofs of physical possession of products and prototypes (e.g., for sites like Kickstarter [Kic], Amazon [Ama] and eBay [eBa]), and of deposited checks [BoA14, Fow10].

Recently, popular health centric *social sensor networks* have also emerged. Products like Fitbit [Fit], Garmin Forerunner[For] and Jawbone Up [Jaw] require users to carry wireless trackers that continuously record a wide range of fitness and health parameters (e.g., steps count, heart rate, sleep conditions), tagged with temporal and spatial coordinates. Trackers report recorded data to a providing server, through a specialized wireless base, that connects to the user’s personal computer (see Figures 5.1(a) and 5.1(b)). The services that support these trackers enable users to analyze their fitness trends with maps and charts, and share them with friends in their social networks. All happening too quickly both for vendors and users alike, this data-centric lifestyle, popularly referred to as the Quantified Self or “lifelogging” is now producing massive amounts of intimate personal data. For instance, BodyMedia [Bod] has created one of the world’s largest libraries of raw and real-world human sensor data, with 500 trillion data points [BMD]. This data is becoming the source of privacy and security concerns: information about locations and times of user fitness activities can be used to infer surprising information, including the times when the user is not at home [Ple], and company organizational profiles [TKS13].

1.2 Contribution

In this dissertation, we investigated the security challenges to verify different forms of media data in online social networks. Concretely, we focused on designing and developing novel solutions including (1) detection of venues in Yelp that are targets of deceptive behaviors; (2) identification of both malware and search rank fraud apps in Google Play; (3) a motion sensor based video liveness verification system to authenticate the videos uploaded into the video sharing sites; (4) a lightweight pro-

tol for providing secure data storage and communication in fitness centric social sensor networks; (5) a framework for preserving privacy and functionality in geosocial networks; and (6) a privacy preserving algorithm for computing safety snapshots of co-located mobile devices as well as geosocial network users. In particular, we make the following contributions in this dissertation.

1.2.1 A Novel Solution for Fraudulent Behavior Detection

We propose Marco (MAlicious Review Campaign Observer), a novel system that leverages the wealth of spatial, temporal and social information provided by Yelp, to detect venues that are targets of deceptive behaviors. Marco exploits fundamental fraudster limitations to identify venues with (i) abnormal review spikes, (ii) series of dissenting reviews and (iii) impactful but suspicious reviews. Marco detects both venues that receive large numbers of fraudulent reviews, and venues that have insufficient genuine reviews to neutralize the effects of even small scale campaigns.

We also develop FairPlay, a system that exploits traces left by fraud in e.g., the app review, install and permission change patterns, the relations between reviewers, and the reviewer feedback in order to identify apps involved in search rank fraud attempts, as well as malware.

Our major contributions include:

- We introduce a *lower bound* on the number of reviews required to launch a review campaign that impacts a target venue’s rating, and prove that this bound renders such campaigns detectable. Our theoretical results force fraudsters to compromise between the impact and undetectability of their review campaigns. [Section 3.4]

- We present *Marco*, a system that leverages novel social, spatial and temporal features gleaned from Yelp to flag suspicious reviews and venues. Marco makes it much harder for fraudsters to hide their trails by making the tasks of posting fraudulent reviews much more costly and complex. [Section 3.4]. We demonstrate that Marco is effective and fast; its classification accuracy is up to 94% for reviews, and 95.8% for venues. It flags 242 of the 7,435 venues analyzed as deceptive; manual inspection revealed that they were indeed suspicious. [Section 3.5].
- We present FairPlay, a system that exploits traces left by fraud in e.g., the app review, install and permission change patterns, the relations between reviewers, and the reviewer feedback in order to identify apps involved in search rank fraud attempts, as well as malware. Our results also show that FairPlay not only achieves a 97+% accuracy in classifying fraudulent and legitimate apps, but its accuracy in classifying malware and legitimate apps exceeds 98%.
- We contribute a novel dataset of reviews and venues, which consists of both ground truth (i.e., objectively correct) and gold standard instances (i.e., selected based on best available strategies); and a large collection of 7,435 venues, 270,121 reviews and 195,417 reviewer profiles [Section 3.3]. We also contribute a novel dataset of identified and monitored 87,223 freshly posted apps in Google Play (along with their 2,850,705 reviews, received from 2,380,708 reviewers) between October 2014 and May 2015, and gold standard datasets of fraudulent and genuine reviews, as well as fraudulent, malware and legitimate apps.
- We introduce Profil_R , a framework that allows the construction of LCPs based on the profiles of present users, while ensuring the privacy and correctness of participants. We also propose a completely decentralized Profil_R extension,

built around the notion of *snapshot* LCPs. The distributed Profil_R enables user devices to aggregate the profiles of co-located users, without assistance from a venue device. We also demonstrate that iSafe is efficient: even on a smartphone, the computation and communication overheads are a few hundred milliseconds.

- We introduce iSafe, a distributed algorithm that addresses privacy concerns raised by the use of trajectory traces and associated crime and safety index values. iSafe takes advantage of the wireless capabilities of mobile devices to compute real-time snapshots of the safety profiles of close-by users in a privacy preserving manner. iSafe uses secret splitting and secure multi-party computation tools to aggregate the trajectories of co-located users without learning the private information of participants. We have extensively evaluated Android and browser plugin implementations of iSafe, using crime and census data from the Miami-Dade county (FL) as well as data we have collected from the accounts of users and businesses in Yelp [Yela].

1.2.2 A Comprehensive Solution for Video Liveness Verification

We introduce Movee, a motion sensor based video liveness verification system. Movee leverages the ubiquitous mobile device accelerometers and the intrinsic movements of the user’s hand and body during the shooting of the video. Movee exploits the intuition that video frames and accelerometer data captured simultaneously will bear certain relations. Specifically, the movement of the scene recorded in the video stream should be related to the movement of the device registered by the accelerometer. We conjecture that such relations are difficult to fabricate and em-

ulate. However Movee has important weaknesses: i) it is not user transparent to the extent that it imposes an explicit verification step on users, ii) it severely limits the movements in the verification step to one of four pan movements, and iii) it is vulnerable to “stitch” attacks in which the attacker creates a fraudulent video by first live recording a genuine video and then pointing the camera to a pre-recorded target video.

To address these limitations, we introduce Vamos, a Video Accreditation through Motion Signatures system. Vamos provides liveness verifications for videos of arbitrary length. It is resistant to a wide range of attacks including those by fully automated systems and those employing trained human experts. Vamos is completely transparent to the users; it requires no special user interaction, nor change in user behavior. Instead of enforcing an initial verification step, Vamos uses the entire video and acceleration stream for verification purposes: It divides the video and acceleration data into fixed length chunks. It then classifies each chunk and uses the results, along with a suite of novel features that we introduce, to classify the entire sample. This process enables Vamos to efficiently detect several potent attacks, including stitch attacks. Vamos does not impose a dominant motion direction, thus, does not constrain the user movements. Instead, Vamos verifies the liveness of the video by extracting features from *all* the directions of movement, from both the video and acceleration streams. Vamos improves on the free-form video motion verification accuracy of Movee by more than 15% in the domain of 6 second *Cluster Attack* videos, and by more than 30% in the domain of whole length *Cluster and Stitch Attack* videos (see Section 5.2 for a discussion of the adversary model).

Removed video length and movement constraints provide additional flexibility for attackers to create fraudulent videos. In order to study the security of the new

unconstrained setting, we i) propose a novel, motion based video classification system, ii) introduce several attacks targeted at sensor based video liveness verification, and iii) show experimental evidence on a wide range of data collected through user studies and from public sources.

The contributions of this work are the following.

- Introduce the “liveness” analysis problem to videos captured from mobile devices.
- Develop a video liveness verification solution, Movee to detect fraudulent video and inertial sensor chunks that encode arbitrary motions. Also introduce Vamos, a system that detects fraudulent video and accelerometer streams of arbitrary length, and is resilient to powerful attacks [§ 4.3.5].
- Introduce a novel classification of mobile videos [§ 4.2.3].
- Introduce a sensor based attack model and develop novel attacks targeted against video verification mechanisms.
- Collect datasets of free-form and citizen journalism videos [§ 4.5]. Show that the performance of Vamos is dependent on the video motion classification [§ 4.6.7]. Predict the classification of Vamos on sensor-less citizen journalism videos.
- Provide a full-fledged implementations of Movee and Vamos, each consisting of a mobile client and a server component. Our cross-validation tests show that Movee achieves an accuracy that ranges between 68% and 93% on our attack datasets created on a Samsung Admire smartphone. On a Google Glass device, Movee’s accuracy ranges between 76-91% for the attacks tested.

1.2.3 A Secure Solution for Data Storage and Communication in Health-centric Devices

The third contribution is a protocol, SensCrypt for secure data storage and communication, for use by makers of affordable and lightweight personal trackers. SensCrypt thwarts not only the attacks we introduced, but also defends against powerful JTAG Read attacks. We have built Sens.io, an Arduino Uno based tracker platform, of similar capabilities but at a fraction of the cost of current solutions. On Sens.io, SensCrypt imposes a negligible write overhead and significantly reduces the end-to-end sync overhead of Fitbit and Garmin. Concretely, the contributions can be described as follows:

- Reverse engineer the semantics of the Fitbit Ultra and Garmin Forerunner communication protocol. [Section 5.2.3].
- Build FitBite and GarMax, tools that exploit vulnerabilities in the design of Fitbit and Garmin to implement several attacks in a timely manner [Section 5.3].
- Devise SensCrypt, a secure solution that imposes no storage overhead on trackers and requires only computationally cheap operations. [Section 5.4] Show that SensCrypt protects even against invasive attackers, capable of reading the memory of captured trackers [Section 5.5].
- Implement Sens.io, a tracker platform, of similar capabilities with existing popular solutions but at a fraction of the cost [Section 5.6.1]. Show that SensCrypt running on Sens.io is very efficient [Section 5.6]

While SensCrypt’s defenses may not be immediately adopted by existing products ¹, this paper provides a foundation upon which to create, implement and test new defensive mechanisms for future tracker designs.

1.2.4 Toward Preserving Privacy and Functionality in Geosocial Networks

We propose to take first steps toward addressing the conflict between profit and privacy in geosocial networks. We introduce Profil_R , a framework that allows the construction of LCPs based on the profiles of present users, while ensuring the privacy and correctness of participants. We also investigate the combination of space and time indexed crime datasets, with mobile technologies and online social networks to provide personalized and context aware safety recommendations for mobile and social network users. Concretely, the contributions can be described as follows:

- Introduce the problem of computing location centric profiles (LCPs) while simultaneously ensuring the privacy and correctness of participants.
- Propose Profil_R , a framework for computing LCPs. Devise both a venue centric and a decentralized solution. Prove that Profil_R satisfies the proposed privacy and correctness properties.
- Define location centric, static crime and safety metrics, based on recorded crime events. Show that timeseries forecasting tools are able to predict future crime and safety index values of locations, based on past crime events.

¹We have contacted Fitbit and Garmin with our results. While interested in the security of their users, they have declined collaboration.

- Introduce iSafe, a distributed algorithm that addresses privacy concerns raised by the use of trajectory traces and associated crime and safety index values. iSafe takes advantage of the wireless capabilities of mobile devices to compute real-time snapshots of the safety profiles of close-by users in a privacy preserving manner.
- Evaluate Profil_R through an Android implementation. Show that Profil_R is efficient even when deployed on previous generation smartphones. Extensively evaluate Android and browser plugin implementations of iSafe, using crime and census data from the Miami-Dade county (FL) as well as data collected from the accounts of users and businesses in Yelp [Yela].

1.3 Organization of the dissertation

To facilitate the reading and understanding, we hereby give an outline of the materials presented in this dissertation. In the next chapter, we would firstly state the problems we would address in this work. In Chapter 3, we will study the problem of detecting malicious behaviors performed through review campaigns in review centered social networks and app markets, and also the impact of Yelp elite events on the ratings of hosting venues. Then in Chapter 4, we will address the fundamental question of whether the visual stream uploaded by a user has been captured live on a mobile device, and has not been tampered with by an adversary. We also introduce the concept of video motion categories to annotate the camera and user motion characteristics of arbitrary videos and demonstrate the effectiveness of our solution across different motion categories. Afterwards, in Chapter 5, we will demonstrate vulnerabilities in the storage and transmission of personal fitness data in popular wearable trackers and then devise a secure and efficient solution for storing and

communicating tracker sensor data. After that in Chapter 6, We will at first define the problem of privacy conflict between users and social network providers and then introduce Profil_R along with a distributed, real-time variant of Profil_R and the notion of snapshot LCPs and prove its privacy and correctness. We will then extend Profil_R by introducing the concepts of personalized and context aware safety as well as the iSafe solution after investigating relationships between social networks and crime levels. Finally, we will summarize the contributions of our work and conclude this dissertation in Chapter 7.

CHAPTER 2

RELATED WORK

In this chapter, we would highlight the research efforts that are related to the techniques presented in this dissertation. In particular, Section 2.1 presents the existing works on fraudulent review detection and malware app identification that are used in this problem; Section 2.2 reviews the existing approaches on motion estimation and video authentication; Section 2.3 describes the existing works of exploiting security vulnerabilities and securing solutions in health-centric sensor networks; and Section 2.4 describes the existing approaches for preserving users' privacy and functionality in online social networks.

2.1 Related Work in Fraudulent Behavior Detection

In this section, three related areas that are related to our proposed solution will be discussed: 1) The existing work of detecting fraudulent reviews, which is directly related to the problem we intend to solve; 2) The existing work of sybil detection in social networks, which are very useful approaches in detecting deceptive behaviors; and 3) The existing approaches of malware detection in app markets.

2.1.1 Research in Detecting Fraudulent Reviews.

Jindal and Liu [JL08] introduce the problem of detecting opinion spam for Amazon reviews. They proposed solutions for detecting spam, duplicate or plagiarized reviews and outlier reviews. Jindal et al. [JLL10] identify unusual, suspicious review patterns. In order to detect “review spam”, Lim et al. [LNJ⁺10] propose techniques that determine a user's deviation from the behavior of other users reviewing similar products. Mukherjee et al. [MLG12] focus on fake reviewer groups; simi-

lar organized fraudulent activities were also found on online auction sites, such as eBay [PCWF07]. Mukherjee et al. [MKL⁺13] leverage the different behavioral distributions of review spammers to learn the population distributions of spammer and non-spammer clusters. Li et al. [LHY⁺11] exploit the reviews of reviews concept of Epinions to collect a review spam corpus, then propose a two view, semi-supervised method to classify reviews.

Ott et al. [OCCH11] integrate work from psychology and computational linguistics to develop and compare several text-based techniques for detecting deceptive TripAdvisor reviews. To address the lack of ground truth, they crowdsourced the job of writing fraudulent reviews for existing venues.

Unlike previous research, we focus on the problem of detecting *impactful* review campaigns. Our approach takes advantage of the unique combination of social, spatial and temporal dimensions of Yelp. Furthermore, we do not break Yelp’s terms of service to collect ground truth data. Instead, we take advantage of unique Yelp features (i.e., spelp sites, consumer alerts) to collect a combination of ground truth and gold standard review and venue datasets.

Feng et al [FXGC12] seek to address the lack of ground truth data for detecting deceptive Yelp venues: They introduce three venue features and use them to collect gold standard sets of deceptive and legitimate venues. They show that an SVM classifier is able to classify these venues with an accuracy of up to 75%. In Section 3.5 we confirm their results on our datasets. We show that with an accuracy of 95.8%, Marco significantly outperforms the best strategy of Feng et al [FXGC12].

Li et al. [LHYZ11] and Ntoulas et al. [NNMF06a] rely on the review content to detect review spam. Li et al. [LHYZ11] exploit machine learning methods in their product review mining system. Ntoulas et al. [NNMF06a] propose several heuristic methods for detecting content based spam and combine the most effective

ones to improve results. Our work differs through its emphasis on relationship among reviewers, friends and ratings in the context of Yelp’s spatial and temporal dimensions.

Gao et al. [GHW⁺10] target asynchronous wall messages to detect and characterize spam campaigns. They model each wall post as a pair of text description and URL and apply semantic similarity metrics to identify large subgraphs representing potential social spam campaigns and later incorporate threshold based techniques for spam detection. Instead, we focus on temporal and geosocial review context, the where reviewer activity and behavioral pattern are of significant importance.

Wang et al. [WXLY11] introduce the concept of heterogeneous review graphs and iterative methods exploring relationship among reviewers, reviews and stores to detect spammers. While we also consider social relations among reviewers we differ on our focus on temporal and spatial dimensions.

2.1.2 Research in Sybil Detection.

Sybil accounts can be used to launch review campaigns, by enabling a single adversary to write multiple reviews for the same venue, each from a different account. Yelp identifies venues that receive multiple reviews from the same IP address (but different user accounts). Tools such as proxies [Hid] and anonymizers (e.g., Tor [DMS04]) can however be used to avoid detection.

SybilInfer [DM09], detects Sybil nodes in social networks by using Bayesian inference and knowledge of the social network graph. Sybil tolerant solutions like DSybil exploit the heavy-tail distribution of the typical behavior of honest users and rely on user weights to identify whether the system needs more opinions or not. Similarly, SumUp [TML09] uses “adaptive vote flow aggregation” to limit the

number of fake feedback provided by an adversary to the number of attack edges in the trust network - that is, the number of bi-directional trust edges the attacker is able to establish to other users. Molavi et al. [KKSM13] propose to associate weights with ratings and introduce the concept of “relative ratings” to defend against bought ratings and ratings from Sybil accounts. When given access to the perspective of the social network provider, Wang et al. [WKW⁺13] proposed an approach that detects Sybil accounts based on their click stream behaviors (traces of click-through events in a browsing session).

Our work aims to complement Sybil detection techniques. Reviews written from accounts detected to be Sybils may be classified as fraudulent. The number (or percentage) of reviews of a venue written from Sybil accounts can be used as a feature to detect “deceptive” venues. Conversely, user accounts with high numbers of posted fraudulent reviews may be used as candidates for further Sybil detection screenings.

2.1.3 Research in Detecting Deceptive and Malware apps

Zhou and Jiang [ZJ12] collect 1,200 Android malware samples and characterize their installation method, activation mechanism and the nature of their malicious payload. They reveal a fast evolution of malware to bypass the detection mechanisms of anti-virus tools. Burguera et al. [BZNT11] use crowdsourcing to collect system call traces from real users then use a partitioned clustering algorithm to cluster the collected data and differentiate between benign and malicious apps. Shabtai et al. [SKE⁺12] extract features from monitored apps (e.g., CPU consumption, packets sent, running processes, keyboard/touch-screen presses) and use machine learning to identify malicious apps. Grace et al. [GZZ⁺12] use static analysis to efficiently

identify high and medium risk apps, that does not rely on malware samples and signatures.

Previous work has also studied the ability of app permissions to pinpoint malware [SLG⁺12, SK12, SSL⁺13]. Sarma et al. [SLG⁺12] use the permissions requested by an app and by apps in the same category to inform users of the risks vs. benefits tradeoffs of the app. Peng et al. [PGS⁺12] propose a score to measure the risk of apps, based on probabilistic generative models such as Naive Bayes. Sahs and Khan [SK12] used features extracted from app permissions and control flow graphs to train an SVM classifier on 2000 benign and less than 100 malicious apps. Yerima et al. [YSM14] also use features extracted from app permissions, API calls and commands extracted from the app executables. Sanz et al. [SSL⁺13] rely strictly on permissions as sources of features for several machine learning tools. They use a dataset of around 300 legitimate and 300 malware pps.

Google has deployed Bouncer, a framework that monitors published apps to detect and remove malware. Oberheide and Miller [OM12] have analyzed and revealed details of Bouncer (e.g., based in QEMU, using both static and dynamic analysis). Bouncer is not sufficient - our results show that 948 apps out of 7,756 apps that we downloaded from Google Play are detected as suspicious by at least 1 anti-virus tool.

2.2 Related Work in Visual Verifications through Liveness Analysis

There are mainly three areas related to this research topic: 1) Video authentication based on video sequence and sensors present in the mobile device; 2) Biometric

liveness verification approaches; 3) Motion and pose estimation. In the following, we will discuss these related works in detail.

2.2.1 Research in Video & Acceleration

The combination of video and accelerometer data has been studied by Hong et al. [HRWZ08] in order to improve the compute-intense motion estimation in video encoding. They have shown that the use of accelerometer data improves the speed of the encoding process by a factor of 2-3. Moiz et al. [MLSL] introduced and developed a wearable, multi-modality, motion capture platform, and used its inertial and ultrasonic sensors to estimate position. The focus of our work is different, on verifying liveness of a video through the consistency of its video and accelerometer data. Indyk et al. [IIS99] studied the problem of finding pirated video on the Internet. They propose to extract a small number of pertinent features (temporal fingerprints) based on the shot boundaries of a video sequence, and match them against a database of videos. We note our work is on an orthogonal problem, of verifying the liveness of a video claimed to have been taken by a mobile device user. As such, these two problems can complement each other. Liu et al. [LLLS14] proposed a solution for summarizing (i.e., extracting important frames from) mobile videos captured simultaneously with acceleration and orientation streams. The acceleration values are used to exclude outliers. Abdollahian et al. [ATPD10] define a “camera view” concept, and use camera motion parameters to temporally segment, summarize and annotate user generated videos. It will be interesting to evaluate a more efficient, video summary based Vamos: detect fraud by identifying discrepancies between video and acceleration summaries.

Several video watermarking algorithms has been proposed for video content authentication [ZZL⁺12, CJHP10]. The goal of Vamos is however not to authenticate the recorded video, but to verify the video liveness claim. We note that watermarking only works if all the videos in the world employ it. Furthermore, the defenses provided by invisible watermarks are defeated by projection attacks.

2.2.2 Research in Biometric Liveness.

Kollreider et al. [KFB09] study the problem of verifying the actual presence of a live face in contrast to a photograph (playback attack) for face recognition based biometrics. They introduce a lightweight optical flow approach that estimates face motion estimation on the structure tensor and a few input frames. Park et. al. [PWM07] introduce a liveness detection method for distinguishing a two-dimensional object from a three-dimensional object. The approach proposed uses video sequence images and does not require additional hardware or user interaction. Their work has direct application to face recognition biometrics: it can identify the use of a flat picture. Further work is needed to understand the vulnerability of this approach to photo movement and photo bending/3D printing attacks.

Multi-modal approaches relying on different sensor sets [FD00, CW06, Chely] have been proposed, to exploit the static and dynamic relationship between voice and face information from speaking faces for biometric authentication. Chetty [Chely] proposed liveness checking techniques for multimodal biometric authentication systems. Their techniques fuse acoustic and visual speech features and measure the degree of synchronization between the lips and the voice extracted from speaking face video sequences.

Accelerometers have been used to provide biometric information, in the form of gait or gesture recognition. Mantyjarvi et al. [MLV⁺ch] proposed solutions that achieve low EER (equal error rates) for identifying users of mobile devices from gait signal acquired with three-dimensional accelerometers, when the device was worn on the belt, at the back. Pylvänäinen [Pyl05] used 3D accelerometers and hidden Markov models to identify gestures performed using a mobile device.

2.2.3 Research in Pose Estimation.

Full-body human pose recognition from images is a fundamental problem in computer vision, that has been extensively studied, see e.g., [MCT09, ROUMdR08, BYS07, AT06, MM06, SVD03]. Rogez et al. [RRR⁺08] propose an efficient method to jointly localize humans and recognize their pose in images, using an exemplar based approach and fast search techniques. Murphy-Chutorian and Trivedi [MCT09] survey work done in estimating head pose from images. Rodgers et al. [RAPK06] propose a probabilistic framework to detect articulated objects and their pose in 3D range scan data, without knowledge of the object orientation, in the presence of occlusion and clutter. Huang and Trivedi [HT04] introduce a framework to detect and track pose estimation of faces in video streams. Wang et al. [WLTX06] propose real-time multi-view face detection and pose estimation in video streams. Rehlinger and Ghosh [RG03] perform rigid body pose estimation using inertial sensors and a monocular camera.

The detection and tracking of human and object pose in the captured videos can benefit Movee as long as we can identify consistencies between the changes in pose and the simultaneously captured acceleration information. One difficulty may arise from the presence of multiple humans and objects in captured videos.

Movee achieves its goal through a simpler yet effective approach instead of trying to accurately perform pose estimation: it extracts and verifies the consistency of motion features from both the video frames and the acceleration stream.

2.3 Related Work in Secure Management of Health-centric Devices

In the context of implantable medical devices (IMDs) Halperin et al. [HHBR⁺08] introduce novel software radio attacks and propose zero power notification, authentication and key exchange solutions. Rasmussen et al. [RCHBC09] propose proximity based access control solutions for IMDs. The different mission of fitness trackers creates different design constraints. First, unlike IMD security, where the focus is on authentication and key exchange, SensCrypt’s focus is on the secure storage and communication of tracker data. This is further emphasized by our need to also consider attackers that can perform Capture and JTAG-R attacks, for both trackers and bases (readers in the IMD context). While such attacks may not be possible for IMDs, and IMD readers may be expensive enough to afford tamper proof memory, these assumptions do not hold for most existing fitness centric social sensor network solutions. Furthermore, while additional user interaction may be naturally accepted for IMDs, fitness security solutions should minimize or even eliminate user involvement.

Tsubouchi et al. [TKS13] have shown that Fitbit data can be used to infer surprising information, in the form of working relations between tracker carrying co-workers. This information could be used to surreptitiously learn the organizational profile of a company. This work assumes access to the fitness data of other users, a task that part of our paper undertakes.

Naveed et al. [NZD⁺14] introduced an “external device mis-bonding attack” for Bluetooth enabled Android health/medical devices, then collected sensitive user data from and fed arbitrary information into the user’s account. They developed Dabinder, an OS level defense that generates and enforces secure bonding policies between a device and its official app. Our work differs in the types and implementation of attacks, and in the solution placement: SensCrypt is implemented at the tracker and webserver, whereas Dabinder is focused on the base.

Lim et al. [LOCL10] analyzed the security of a remote cardiac monitoring system where the data originating from the sensors is sent through a Body Area Network (BAN) gateway and a wireless router to a final monitoring server. Muraleedharan et al. [MO08] proposed DoS attacks including Sybil [NSSA04] and wormhole [KD03] attacks, for a health monitoring system using wireless sensor networks. They introduced an energy-efficient cognitive routing algorithm to address such attacks. Our work differs through its system architecture, communication model and tracker capabilities.

Barnickel et al. [BKM10] targeted security and privacy issues for HealthNet, a health monitoring and recording system. They proposed a security and privacy aware architecture, relying on data avoidance, data minimization, decentralized storage, and the use of cryptography. Marti et al. [MD07] described the requirements and implementation of the security mechanisms for MobiHealth, a wireless mobile health care system. MobiHealth relies on Bluetooth and ZigBee link layer security for communication to the sensors and uses HTTPS mutual authentication and encryption for connections to the backend.

2.4 Related Work in ensuring GSN Privacy and Safe Cities

2.4.1 Research in GSN Privacy

Location cloaking. Location and temporal cloaking techniques, or introducing errors in reported locations in order to provide 1-out-of- k anonymity have been initially proposed in [GG03], followed by a significant body of work [HGH⁺08, OTGH10, PMX09]. We note that Profil_R provides an orthogonal notion of k -anonymity: instead of reporting intervals containing k other users, we allow the construction of location centric profiles only when k users have reported their location. Computed LCPs hide the profiles of participating users: user profiles are anonymous, only aggregates are available for inspection, and interactions with venues and the provider are indistinguishable.

l-diversity. Machanavajjhala et al. [MGKV06] have shown that k -anonymity for published user data, where each record is indistinguishable from at least $k - 1$ other records (for sensitive attributes), is not sufficient to provide anonymity. To address this, they defined an l -diverse data block of tuples from various users, as one that contains at least l “well-represented” values for any sensitive attribute. We note that we do not collect individual (anonymized) user data. Instead, we build statistics over user data, that can be published only if k users contribute.

GSN privacy. Puttaswamy and Zhao [PZ10] require users to store their information encrypted on the GSN provider. This includes “friendship” and “transaction” proofs, cryptographically encrypted tokens encoding friend relations and messages. The proofs can only be decrypted by those who know the decryption keys. Transaction proofs are stored in “buckets” associated with approximate locations (e.g., blocks), enabling users to retrieve information pertinent to their current location.

Profil_R takes the next step, by enabling the aggregation of user data in a privacy preserving manner.

Mascetti et al. [MFB⁺11] propose solutions that hide user location information from the provider and enable users to control the information leaked to participating friends (e.g., co-location events), with a view to improve service precision, computation and communication costs. Freni et al. [FRVM⁺10] argue that the inherent nature of geosocial networks makes it hard for users to gauge their privacy leaks. The proposed solution relies on a trusted third party to process posted locations according to user preferences, before publishing them on the GSN provider. Wernke et al. [WDR12] use secret sharing and multiple, non-colluding service providers to devise secure solutions for the management of private user locations when none of the providers can be fully trusted. The position of a user is split into shares and each server stores one. A compromised server can only reveal erroneous user positions.

In contrast, Profil_R provides the novel functionality of allowing the provider, venues and even users to privately compute LCPs over visitors or co-located users. Profil_R does not require multiple, mutually untrusted servers, or trusted third parties.

Thompson et. al. [THH⁺09] proposed a solution in which database storage providers compute aggregate queries without gaining knowledge of intermediate results; users can verify the results of their queries, relying only on their trust of the data owner. In addition to assuming a different environment, Profil_R does not assume venue owners to be trustworthy. Toubiana et. al [TNB⁺10] proposed Adnostic, a privacy preserving ad targeting architecture. Users have a profile that allows the private matching of relevant ads. While Profil_R can be used to privately provide location centric targeted ads, its main goal is different - to compute location (venue) centric profiles that preserve the privacy of contributing users.

Online social network privacy. Recent work on preserving the privacy of users from the online social network provider includes Cutillo et al. [CMS09], who proposed Safebook, a distributed online social networks where insiders are protected from external observers through the inherent flow of information in the system. Tootoonchian et al. [TSGW09] proposed Lockr, a system for improving the privacy of social networks by using the concept of a social attestation, which is a credential proving a social relationship. Baden et al. [RB09] introduced Persona, a distributed social network with distributed account data storage. While Profil_R builds on this work by requiring users to store their GSN information, its focus rests on protecting the privacy of users while *simultaneously* allowing venues to collect valuable statistics over visitors. This dual goal of Profil_R differentiates this paper from previous work.

2.4.2 Research for Smart and Safe Cities.

Smart cities have been the focus of recent efforts at IBM [IBM] and several academic research groups at MIT [Lab] and UCLA [UCL]. Caragliu et. al. [CDBN09] present a study on the factors that determine the performance of a “smart city”. They focus specifically on European cities by analyzing urban environments, levels of education and different accessibility modalities that are positively correlated with urban wealth. Since one important aspect of smart cities is safety, Patton [Pat10] propose the use of audio sensors and cameras that allow authorities to quickly respond in an emergency event without receiving a 911 call. We note that we consider a preventive angle, of making users aware of their surroundings.

Furtado et. al. [FAdO⁺10] propose the use of social media in a collaborative effort to inform people about crime events that are not reported to police. Their

wiki website spots areas on the map where participant users have reported crime events. Police departments also release tools to make citizens aware of their safety, e.g., the Miami-Dade police department, deployed an web application [Dep] that identifies crime areas based on current crime reports. Instead, iSafe seamlessly integrates context and time sensitive safety metrics into the everyday user experience. Dynamic safety practices leveraging social networks and GPS mobile phones have been introduced in [YBL⁺08] to create a system for personalized safety awareness. The definition of safety indexes that leverage crime, social and mobile activities, as well as the use of safety predictions, differentiate iSafe.

Participatory sensing is receiving increasing attention. Estrin [Est10] discuss advantages of participatory sensing in health and transportation and provide insights on the architecture of participatory sensing applications. Thiagarajan et. al. [TBGE10] propose cooperative transit tracking using mobile phones. Privacy becomes a serious concern when the user personal information may be compromised. Christin et. al. [CRKH11] present a survey on the efforts made to preserve privacy in participatory sensing systems. In contrast, iSafe does not collect user information, but instead allows devices to aggregate information collected from co-located users without learning personal information.

The problem of crime prediction has been explored in several contexts. Hotspot mapping [CTU08] is a popular analytical technique used by law enforcement agencies to identify future patterns in concentrated crime areas. Different methods and techniques have been analyzed to review the utility of hotspot mapping in [ECC⁺05], [CR05], [Jef99], [CRS02]. Hot spot analysis however, often lacks a systematic approach, as it depends on human intuition and visual inspection.

A variety of univariate and multivariate methods have been used to predict crime. Univariate methods range from simple random walk [BSV98] to more sophisticated

models like exponential smoothing. While exponential smoothing offers greater accuracy to forecast "small to medium-level" changes in crime [GO01], we have shown that ARIMA and ANN models outperformed it on our data. We also note that the end goal of our work is not intrinsically crime forecasting. Instead, we incorporate crime forecasting techniques into our safety metrics, in an attempt to provide to participating users a dynamic framework for safety awareness.

CHAPTER 3

IDENTIFYING DECEPTIVE BEHAVIORS IN ONLINE SOCIAL NETWORKS

In this chapter, we would mainly focus on the problem of detecting deceptive and malicious behaviors seen in review-centric social networks and also in social app markets. Part of the content in this section has been published during my Ph.D study, including the problem formulation and the proposed solutions and its evaluation results.

The outline of this chapter is as follows: The motivation and challenges of this topic will be presented in Section 3.1. The system model will be introduced then in Section 3.2. In Section 3.3, the collected dataset will be described. Then in Section 3.4, the deceptive behavior detection mechanisms will be introduced and evaluated. In Section 3.5, experimental results will be presented to evaluate the performance of our proposed methods. Some limitations of our work will be discussed in Section 3.6. Finally, a short chapter summary about this problem and our proposed solutions will be provided in Section 3.7.

3.1 Motivation and Challenges

Review based geosocial networks are online social networks centered on the location of venues and users as well as on reviews left by users for visited venues. Similarly, in app markets like Google Play, reviews play an influential role to motivate install count and generate revenues for developers. The popularity and impact of reviews makes them an ideal tool for influencing public opinion. The incentive is profit: Anderson and Magruber [AM12] show that in Yelp, an extra half-star rating causes restaurants to sell out 19 percentage points (from 30% to 49%) more frequently. The boundless demand for positive reviews has made the review system an arms

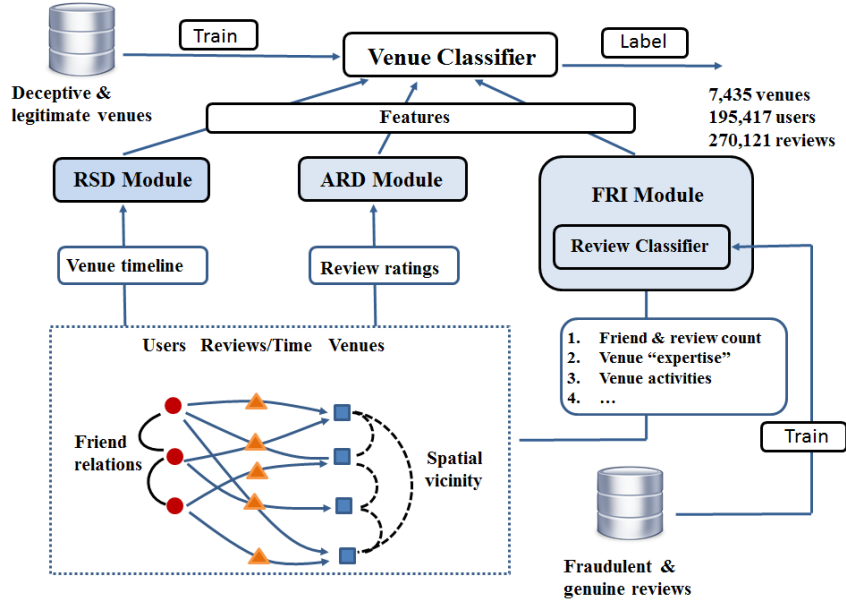


Figure 3.1: System overview of Marco. Marco relies on social, temporal and spatial signals gleaned from Yelp, to extract novel features. The features are used by the venue classifier module to label venues (deceptive vs. legitimate) based on the collected data. Section 3.4 describes Marco in detail.

race of sorts. As more five-star reviews are handed out, even more five-star reviews are needed. Few want to risk being left behind. Determining the number of fake reviews on the social networks is extremely difficult. In the past few years, there was a growing interest in mining reviews and reviewer’s sentiment from both academia and industry. However, the existing work has been mainly focused on extracting and summarizing reviews using natural language processing and data mining techniques [DLP03, HL04, NNMF06b]. Little is known about the characteristics of reviews and behaviors of reviewers.

We present Marco, a novel system that exploits the unique combination of social, spatial and temporal signals gleaned from Yelp, to detect venues whose ratings are impacted by fraudulent reviews. Marco increases the cost and complexity of attacks, by imposing a trade-off on fraudsters, between their ability to impact venue ratings and their ability to remain undetected.

Marco (see Figure 3.1) exploits fundamental fraudster limitations to identify venues with (i) abnormal review spikes, (ii) series of dissenting reviews and (iii) impactful but suspicious reviews. Marco detects both venues that receive large numbers of fraudulent reviews, and venues that have insufficient genuine reviews to neutralize the effects of even small scale campaigns.

We also propose FairPlay, a system that leverages the above observations to efficiently detect Google Play fraud and malware.

3.2 System Model

3.2.1 Yelp’s Review System.

For this work, we focus on Yelp [Yela], a review centric geosocial network that hosts information concerning users and venues. Subscribed users (“yelpers”) have accounts and can write reviews, befriend other subscribers, report locations and search for venues of interest. We use the term “venue” to represent a business or event with an associated location (e.g., restaurants, shops, offices, concerts).

Reviews have a star *rating*, an integer ranging from 1 to 5, with 5 being the highest mark. In Yelp, an *average rating* value is computed for each venue (rounded to the nearest half star), over the ratings of all the posted reviews. For a review R , let $R.\rho$ denote its rating and $R.\tau$ to denote the time when the review was posted. We say a review is “positive” if its rating is at least 4 stars and “negative” if its rating is 2 stars or fewer. In our analysis we do not consider 3 star reviews. Their impact on the rating of the venue is likely to be small: Yelp denotes a 3 star rating as “A-OK”.

3.2.2 Influential & Elite Yelpers.

Users can rate the reviews of others, by clicking on associated buttons (e.g., “useful”, “funny” or “cool” buttons). They can upload photos taken at venues reviewed and perform “check-ins”, to formally record their real-time presence at the venue. Yelp rewards “influential” reviewers (often peer-recommended) with a special, yearly “Elite” badge.

3.2.3 Fraudulent Reviews & Deceptive Venues.

A review is *fraudulent* if it describes a fictitious experience. Otherwise, the review is *genuine*. We say a venue is *deceptive* if it has received a sufficient number of fraudulent reviews to impact its average rating by at least half a star. Otherwise, the venue is *legitimate*.

Yelp relies on proprietary algorithms to filter reviews it considers fraudulent. See [MVL⁺13] for an attempt to reverse engineer Yelp’s filter. Furthermore, Yelp has launched a “Consumer Alert” process, posting “alert badges” on the pages of venues for which (i) people were caught red-handed buying fraudulent reviews, offering rewards or discounts for reviews or (ii) that have a large number of reviews submitted from the same IP address. The consumer alert badge is displayed for 90 days.

3.2.4 Yelp Events

Yelp organizes special *Elite events*, at select venues, where only Elite badge holders are invited. For each event, Yelp creates a separate Yelp page, containing the name of the event and the name, address and information for the hosting venue. Attendees

are encouraged to review the event account, which then lists the reviews, just like a regular venue.

3.2.5 Android App Market of Google Play

We focus on the Android app market ecosystem of Google Play. The participants, consisting of users and developers, have Google accounts. Developers create and upload apps, that consist of executables (i.e., “apks”), a set of required permissions, and a description. The app market publishes this information, along with the app’s received reviews, aggregate rating, install count, size, version number, time of last update, and a list of “similar” apps.

Only after a user installs an app on a registered mobile device, is the user allowed to either rate or review the app. A review consists of a star rating (1-5 stars) and a text component. The text is optional and consists of a title and a description. Google Play limits the number of reviews displayed for an app to 4000, but publishes the total number of reviews received and their aggregate rating. Google also publishes the app’s install range, consisting of predefined buckets (e.g., 50-100, 100-500). Reviewers may have Google+ accounts, in which case they have followers.

3.2.6 Adversarial Model for Google Play Market

We consider both rational fraudulent and malware developers. To goal of fraudulent developers is to maximize their revenue for minimal investment. To achieve this goal, they often rely on crowdsourcing sites (e.g., Freelancer [Fre], Fiverr [Fiv], BestAppPromotion [Bes15]), to hire teams of willing and often experienced workers to commit fraud collectively. On the other hand, we also consider malicious devel-

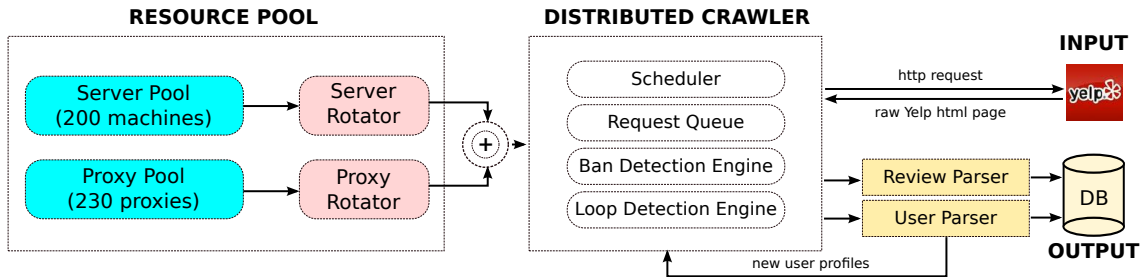


Figure 3.2: YCrawl system architecture. YCrawl relies on a pool of servers and proxies to issue requests. The scheduler relies on a request queue to ensure there are no loops in the crawling process.

opers, who upload malware and even attempt to fraudulently promote it in order to maximize its impact.

3.3 Collected Dataset.

3.3.1 Collected Yelp Data

In this section we describe the Yelp datasets we collected using the *YCrawl* crawler that we developed. Our data consists of: (i) 90 deceptive and 100 legitimate venues; (ii) 426 fraudulent and 410 genuine reviews; and (iii) a large collection of 7,435 venues and their 270,121 reviews from 195,417 reviewers, from San Francisco, New York City and Miami.

YCrawl.

We have developed YCrawl, a crawling engine for automatically collecting data from Yelp user and venue pages. YCrawl consists of 1820 lines of Python code. It fetches the raw HTML pages of target Yelp user and venue accounts. Figure 3.2 illustrates the system design of YCrawl.

Yelp keeps track of requests made from a single IP and suppresses any IP making an exorbitant number of requests within a short time window ¹. To overcome this limitation, YCrawl uses a pool of servers and IP proxies: For every request, YCrawl randomly picks a server and proxy. If the request is not successful, a new request is made using a different proxy. A centralized scheduler maintains a request queue to ensure there are no loops in the crawling process.

At the time when we performed this data collection, Yelp’s filtered reviews could only be accessed by solving a CAPTCHA. In order to collect filtered reviews we used DeathByCaptcha [DBC] to programmatically collect CAPTCHA protected reviews filtered by Yelp.

We used YCrawl to collect a seed dataset of random venue and user accounts, using a breadth-first crawling strategy and stratified sampling [TD00]. First, we selected a list of 10 major cities (e.g., NY, San Francisco, LA, Chicago, Seattle, Miami) in the U.S. and we collected an initial random list of 100 venues from each of these cities as a seed dataset. We note that the strata venues are mutually exclusive, i.e. venues do not belong to two or more different cities. We then randomly selected 10,031 Yelp users who reviewed these venues, and collected their entire Yelp data (the html pages), including all their reviews, for a total of 646,017 reviews. This process enabled us to avoid bias toward high degree nodes (users with many friends, venues with many reviews), which is a common problem when crawling social networks [GKBM10]. We have then randomly selected a list of 16,199 venues, reviewed by the previously collected 10,031 Yelp users. We have collected the html pages of the selected the venues, including all their reviews.

¹Such IP addresses are suppressed from Yelp’s servers and this remains in place for a few weeks (or sometimes forever).

The Data.

We use the term “ground truth” set to denote data objectively known to be correct. We use the term “gold standard” to denote data selected according to the best available strategies. We collect such data following several stringent requirements, often validated by multiple third-parties.

Ground truth deceptive venues. We relied on Yelp’s “Consumer Alert” feature to identify deceptive venues. We have used Yelp and Google to identify a snapshot of all the 90 venues that received consumer alerts during July and August, 2013.

Gold standard legitimate venues. We have used the collected list of 16,199 venues previously described to first selected a preliminary list of venues with well known consistent quality, e.g., the “Ritz-Carlton” hotel. We have then manually verified each review of each venue, including their filtered reviews. We have selected only venues with at most one tenth of their reviews filtered by Yelp and whose filtered reviews include a balanced amount of positive and negative ratings. While Yelp tends to filter reviews received from users with few friends and reviews, Feng et al. [FXGC12] showed that this strategy is not accurate. In total, we selected 100 legitimate venues.

In addition to collecting the html pages of all the reviews of the selected deceptive and legitimate venues, we have also collected the html pages of all the users who wrote reviews for them, and the html pages of all the reviews written by these reviewers. This data enables us to extract the features that we introduce in the following sections.

For the 90 deceptive venues we have collected their 10,063 reviews written by 7,258 reviewers. We have collected all the reviews (311,994 in total) written by the 7,258 reviewers of the 90 deceptive venues. In addition, we have collected the 9,765 reviews, written by 7,161 reviewers, of the 100 legitimate venues. We have

then collected all the reviews written by these 7,161 reviewers, for a total of 530,408 reviews. Thus, for these 190 venues, we have collected more than 840,000 reviews. Note how the 90 deceptive venues have received more reviews than the 100 legitimate venues. However, the total number of reviews written by reviewers of legitimate venues significantly exceeds those written by the reviewers of deceptive venues.

Gold standard fraudulent reviews. We have used spelp (Spam + Yelp) sites (e.g., [SPE, FLE]), forums where members, often “Elite” yelpers with ground truth knowledge, reveal and initiate the discussion on fraudulent Yelp reviews. While in theory such sites are ideal targets for fraudulent behavior, the high investment imposed on fraudsters, coupled with the low visibility of such sites, make them unappealing options. Nevertheless, we have identified spelp reviews that (i) were discussed by and agreed upon by *multiple* other Yelp users, (ii) were written from accounts with no user photo or with web plagiarized photos (identified through Google’s image search), and that (iii) were short (less than 50 words). From this preliminary set, we have *manually* selected 410 generic reviews, that provide no venue specific information [Revb].

Specifically, each “spelp” review we collected was posted by a Yelp users, and discussed and agreed upon by *multiple* other Yelp users.

Gold standard genuine reviews. Given the seed user and venue datasets previously described, we have extracted a list of 410 genuine reviews satisfying a stringent test that consists of multiple checkpoints. In a first check we used Google (text and image search) to eliminate reviews with plagiarized text and reviewer account photos. In a second check we discarded short (less than 50 words), generic reviews, lacking references to the venue. Third, we gave preference to reviews written by users who

- Reached the “Elite” member status at least once.

- Participated in forums e.g. Yelp Talk.
- Garnered positive feedback on their reviews.
- Provided well thought out personal information on their profile.

We have collected the 54,213 reviews written by the writers of the 410 genuine reviews. We have also collected the 1,998 reviews written by the writers of the 426 fraudulent reviews.

Large Yelp Data Set. We have used YCrawl to collect the data of 7,435 car repair shops, beauty & spa centers and moving companies from San Francisco, New York City and Miami. The collection process took 3 weeks. Of the 7,345 venues, 1928 had no reviews posted. We have collected all their 270,121 reviews and the data of their 195,417 reviewers (one user can review more than 1 of these venues). Table 3.8 shows the number of venues collected for each venue type and city. Yelp limits the results for a search to the first 1000 matching venues. Entries with values less than 1000 correspond to cities with fewer than 1000 venues of the corresponding type.

Yelp Event Collection. We have collected Yelp events from 60 major cities covering 44 states of USA. The remaining states had no significant Yelp events or activities (WY, VT, SD, NE, WV, ND). After identifying an Elite event, we identified the hosting venue through either its name or address. We used YCrawl to collect a majority of the available Yelp events and hosting venues, for a total of 149 pairs.

For each Yelp event and corresponding venue, we have collected their name, number of reviews, star rating and all their reviews. For each review, we have collected the date when it was written, the rating given and the available information about the reviewer, including the Elite status, number of friends and number of reviews written. In total, we have collected 24,054 event/hosting venue reviews.

While we are unable to make public these datasets, due to possible legal action from Yelp, we recommend researchers to contact us with questions concerning this data.

3.3.2 Collected Google Play Data

GPCrawler: Google Play Data Collector

We have developed GPCrawler, a tool to automatically collect data published by Google Play for apps, users and reviews. At the time of writing this paper, Google prevents scripts from scrolling down a user page, thus, to collect the ids of more than 20 apps reviewed by a user. To bypass this restriction, we developed a Python script and a Firefox add-on. Given a user id, the script opens the user page in Firefox. When the script loads the page, the add-on becomes active. The add-on interacts with Google Play pages using content scripts (Browser specific components that let us access the browsers native API) and port objects for message communication. The add-on displays a “scroll down” button that enables the script to scroll down to the bottom of the page. The script then uses a DOMParser to extract the content displayed in various formats by Google Play. It then sends this content over IPC to the add-on. The add-on stores it, using Mozilla XPCOM components, in a sandboxed environment of local storage in a temporary file. The script then extracts the list of apps rated or reviewed by the user.

We have run these tools on 4 servers (PowerEdge R620, Intel Xeon E-26XX v2 Processors, 64GB RAM, 2TB HDD) to collect the following data.

Monitored Fresh Apps

We used Google Play’s “New Releases” link to identify newly released apps, with a short history on Google Play. Google does not publish the first upload date of an app; we approximate it based on the time of the app’s first review. We have started with a seed set of 25K+ new releases (July 2014) and by October 2014 we had a set of 87,223 new releases, whose first upload time was under 40 days prior to our first collection time. Most of the apps had under 100 reviews.

We have monitored these 87,223 apps between October 24, 2014 and May 5, 2015: we took a “snapshot” of each app, twice a week. An app snapshot consists of app metadata, and includes the install bucket, the permissions requested, the reviews, developer data and similar apps. For each of the 2,850,705 reviews we have collected from these apps, we recorded the reviewer’s name and id (2,380,708 unique ids), date of review, review title, text, and rating.

Gold Standard Data

We now describe the process we employed to collect gold standard datasets of fraudulent and genuine reviews, as well as fraudulent and legitimate apps.

Fraudulent reviews. We have used contacts established among Freelancer [Fre]’s search rank fraud community, to obtain the identities of 15 Google Play accounts that were used to write fraudulent reviews. We call these “seed fraud accounts”. We have retrieved all the apps reviewed from the seed fraud accounts, for a total of 201 unique apps. We call these the “seed fraud apps”.

We have then collected all the 53,625 reviews received by the 201 seed fraud apps². We have used these reviews to identify 188 accounts, such that each account was used to review at least 10 of the 206 seed fraud apps (for a total of 6,488 reviews). We call these, *guilt by association* (GbA) accounts.

To reduce feature duplication, we have used the 1,969 fraudulent reviews written by the 15 seed accounts and the 6,488 fraudulent reviews written by the 188 GbA accounts for the 201 seed fraud apps, to extract a *balanced* set of fraudulent reviews. Specifically, from this total set of 8,457 (=1,969+6,488) reviews, we have collected 2 reviews from each of the 203 (=188+15) suspicious user accounts. Thus, the gold standard dataset of fraudulent reviews contains 406 reviews.

Genuine reviews. We have manually collected a gold standard set of 315 genuine reviews, as follows. First, we have selected the reviews written for apps installed on the Android smartphones of the authors. We then used Google’s text and reverse image search tools to identify and filter those that plagiarized other reviews or were written from accounts with generic photos. We have then manually selected reviews that mirror the authors’ experience, have at least 150 characters, and are informative (e.g., provide information about bugs, crash scenario, version update impact, recent changes, game characters and level difficulty).

Malware apps. We have used GPad (see Section 3.3.2) to collect the apks of a randomly selected subset of 8,220 apps from the 87K “fresh” apps. Figure 3.10(a) shows the distribution of flags raised by VirusTotal, for the 8,220 apks. We note that these apps have not been filtered by Google’s Bouncer [OM12]. From the 523 apps that were flagged by at least 3 tools, we selected those that had at least 10 reviews to form our “malware app” dataset, for a total of x apps.

²The 15 seed fraud accounts were responsible for 1,969 of these 53,625 reviews.

Fraudulent apps. We use the 201 “seed fraud apps”, see above, as the gold standard fraudulent app dataset.

Legitimate apps. We have selected a subset of 925 candidate apps, that have been developed by Google designated “top developers”. We have then used GPAD to download their apks and filter out those detected to be suspicious even by one anti-virus tools. We have manually investigated 601 of the remaining apps, and selected a set of 200 apps that have more than 10 reviews and where developed by reputable media outlets (e.g., NBC, PBS) or have an associated business model (e.g., fitness trackers), or whose reviews report experiences similar to the ones experienced by the paper authors.

3.4 Proposed Methods

We present Marco, a system for automatic detection of fraudulent reviews, deceptive venues and impactful review campaigns. We begin with a description of the adversary and his capabilities.

3.4.1 Deceptive Venue Detection.

Overview of Marco.

Marco, whose functionality is illustrated in Figure 3.1, consists of 3 primary modules. The Review Spike Detection (RSD) module relies on temporal, inter-review relations to identify venues receiving suspiciously high numbers of positive (or negative) reviews. The Aggregate Review Disparity (ARD) module uses relations between review ratings and the aggregate rating of their venue, at the time of their posting, to identify venues that exhibit a “bipolar” review behavior. The Fraudu-

Notation	Definition
\mathcal{A}	Adversary
V	Target venue
$H_V, \Delta T$	V 's timeline and active interval
$\rho_V(T)$	Rating of V at time T
δr	Desired rating increase by \mathcal{A}
δt	Review campaign duration
q	Number of fraudulent reviews by \mathcal{A}
$R, R.\rho, R.\tau$	Review, its rating and its posting time
n	Number of genuine reviews of V
σ	Sum of ratings of all genuine reviews
p	Number of genuine positive reviews

Table 3.1: Table of Notations

lent Review Impact (FRI) module first classifies reviews as fraudulent or genuine based on their social, spatial and temporal features. It then identifies venues whose aggregate rating is significantly impacted by reviews classified as fraudulent. Each module produces several features that feed into a venue classifier, trained on the datasets of Section 3.3.1. Table 3.1 shows the notations used by Marco.

The approach used in Marco leverages manually labeled data, including fraudulent and genuine reviews, as well as deceptive and legitimate venues, to classify reviews and venues. Marco does not require knowledge of all the data and can classify new data in an online manner. A drawback of this approach stems from the difficulty of acquiring ground truth and gold standard data. While it is also difficult to identify relevant features that are hard to bypass by adversaries, we note that Marco introduces a trade-off for attackers, between impact and detectability.

An alternative approach is to use unsupervised outlier detection solutions [HA04, ZSK12, YTW00, ZSGL07, FM06]. While such solutions do not require labeled data, they require knowledge of the entire dataset. This approach is thus suitable for the providers (i.e., Yelp). We note however that an adversary with sufficient

knowledge of the data can attempt to bypass this approach, by determining and introducing fraudulent data that would not be classified as outlier.

Review Spike Detection (RSD) Module.

A review campaign needs to adjust (e.g., increase) the rating of its target venue, by posting (fraudulent) reviews that compensate the negative ratings of other reviews. The RSD module detects this behavior by identifying venues that receive higher numbers of positive (or negative) reviews than normal.

In the following, our first goal is to prove that review campaigns that impact the ratings of their target venues are detectable. For this, let q denote the total number of fraudulent reviews that \mathcal{A} posts for the target venue V . We focus on the typical scenario where an attacker attempts to increase the rating of V (ballot stuffing). Attempts to reduce the rating of V (bad mouthing) are similar and omitted here for brevity.

\mathcal{A} can follow any strategy, including (i) *greedy*, by posting all q reviews in a short time interval and (ii) *uniform*, by spreading reviews over a longer time interval. While a greedy strategy is likely to quickly impact the venue, a uniform strategy seems more likely to pass unnoticed. However, we show in the following that, if the review campaign is successful, it becomes detectable.

Let T_s and T_e denote the start and end times of the campaign, the times when the first and last fraudulent reviews initiated by \mathcal{A} are posted. $\delta t = T_e - T_s$ is the campaign duration interval. Let n denote the number of genuine reviews V has at the completion of the campaign (time T_e). We assume V receives fraudulent reviews only from \mathcal{A} . We prove the following lower bound on the number of reviews that \mathcal{A} needs to write in order to impact the rating of V .

Claim 1 *The minimum number of reviews \mathcal{A} needs to post in order to (fraudulently) increase the rating of V by half a star is $q = n/7$.*

Proof. Let R_1, R_2, \dots, R_n denote the n genuine reviews of V . Let $\sigma = \sum_{i=1}^n R_i \cdot \rho$. According to Yelp semantics, $R_i \cdot \rho \in [1, 5]$, thus $\sigma \in [n, 5n]$. The ‘‘genuine’’ rating of V is $\rho_V^g = \frac{\sigma}{n}$. In order to minimize q , \mathcal{A} has to write only 5 star reviews. Let δr be the increase in the rating of V generated by \mathcal{A} ’s review campaign. Note that $\delta r \in [0.5, 4)$. Furthermore, $\frac{\sigma}{n} + \delta r \leq 5$, as the final rating of V cannot exceed 5. Hence,

$$\frac{\sigma + 5q}{n + q} = \frac{\sigma}{n} + \delta r,$$

Thus, $q = \frac{n^2 \delta r}{5n - \sigma - n \delta r}$. Given that $\sigma \geq n$, we have $q \geq \frac{n \delta r}{4 - \delta r}$. When $\delta r = 1/2$, this results in $q \geq n/7$. For $\delta r = 1$, $q \geq n/3$, when $\delta r = 2$, $q \geq n$, etc. \square

We say a review campaign is *successful* if it increases the rating of the target venue by at least half a star ($\delta r \geq 1/2$). We introduce now the notion of venue timeline:

Definition 3.4.1 *The timeline of a venue V is the set of tuples $H_V = \{(U_i, R_i) | i = 1..n\}$, the list of reviews R_i received by V from users U_i , chronologically sorted by the review post time, $R_i \cdot \tau$. Let $\Delta T = T_c - T_1$ denote the **active interval** of the venue, where T_c denotes the current time and $T_1 = R_1 \cdot \tau$.*

Figure 3.3 illustrates this concept, by showing the evolution of the positive review (4 and 5 star) timelines of 3 venues selected from the ground truth deceptive venue dataset (see Section 3.3.1). Let p denote the number of positive reviews received by V during its active interval, ΔT . We now show that:

Claim 2 *Assuming a uniform arrival process for genuine positive reviews, the maximum number of genuine positive reviews in a δt interval is approximately $\frac{p}{\Delta T} (1 + \frac{1}{\sqrt{c}})$, where $c = \frac{p}{\Delta T} \frac{\delta t}{\log \frac{\Delta T}{\delta t}}$.*

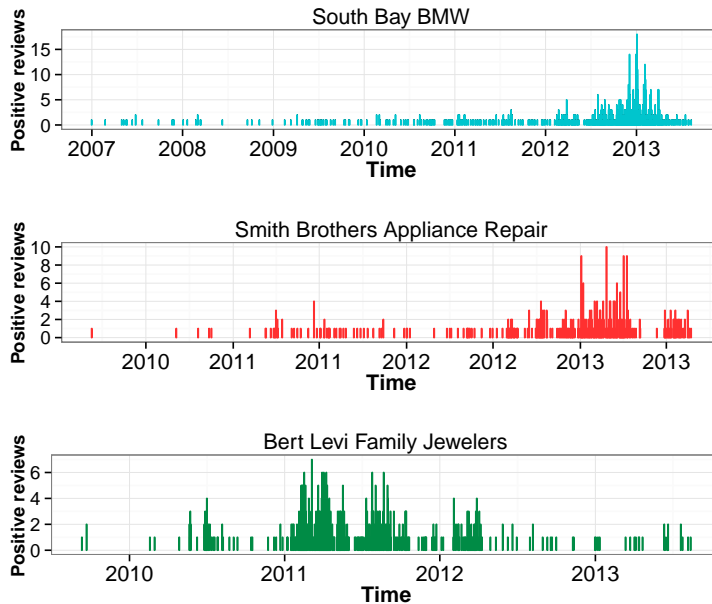


Figure 3.3: Timelines of positive reviews of 3 deceptive venues (see Section 3.3.1). Each venue has several significant spikes in its number of daily positive reviews.

Proof. The distribution of reviews into δt intervals follows a balls and bins process, where p is the number of balls and $\Delta T/\delta t$ is the number of bins. It is known (e.g., [BBFN10, RS98]) that given b balls and B bins, the maximum number of balls in any bin is approximately $\frac{b}{B}(1 + \frac{1}{\sqrt{c}})$, where $c = \frac{b}{B \log B}$. Thus, the result follows.

□

We introduce now the following result.

Theorem 1 *If $n > 49$, a successful review campaign will exceed, during the attack interval, the maximum number of reviews of a uniform review distribution.*

Proof. Let p denote the number of positive, genuine reviews received by the target venue at the end of the review campaign. $p < n$, where n is the total number of genuine reviews at the end of the campaign. According to Claim 1, a successful review campaign needs to contain at least $n/7$ positive (5 star) reviews. Then,

since the expected number of positive genuine reviews to be received in a δt interval will be $\frac{p\delta t}{\Delta T}$, following the review campaign, the expected number of (genuine plus fraudulent) positive reviews in the attack interval will be $\frac{n}{7} + \frac{p\delta t}{\Delta T}$.

The maximum number of positive genuine reviews posted during an interval δt , assuming a uniform distribution, is, according to Claim 2, approximately $\frac{p}{\Delta T} \delta t + \sqrt{\frac{p\delta t}{\Delta T} \log \frac{\Delta T}{\delta t}}$. Thus, the number of positive reviews generated by a review campaign exceeds the maximum positive reviews of a uniform distribution if

$$\frac{n}{7} + \frac{p\delta t}{\Delta T} > \frac{p\delta t}{\Delta T} + \sqrt{\frac{p\delta t}{\Delta T} \log \frac{\Delta T}{\delta t}}.$$

Since $n > p$, this converts to $\frac{n}{49} > \frac{\log \frac{\Delta T}{\delta t}}{\frac{\Delta T}{\delta t}}$. Since $\Delta T > \delta t$, we have that $\frac{\log \frac{\Delta T}{\delta t}}{\frac{\Delta T}{\delta t}} < 1$. Thus, the above inequality trivially holds for $n > 49$. \square

Theorem 1 introduces a tradeoff for attackers. Specifically, an attacker can choose to either (i) post enough reviews to impact the rating of a venue (launch a successful campaign) but then become detectable (exceed the maximum number of reviews of a uniform distribution), or (ii) remain undetected, but then do not impact the rating of the venue.

Detect abnormal review activity. We exploit the above results and use statistical tools to retrieve ranges of abnormal review activities. In particular, our goal is to identify spikes, or outliers in a venue’s timeline. For instance, each venue in Figure 3.3 has several significant review spikes. The RSD module of Marco uses the measures of dispersion of Box-and-Whisker plots [TD00] to detect outliers. Specifically, given a venue V , it first computes the quartiles and the inter-quartile range IQR of the positive reviews from V ’s timeline H_V . It then computes the upper outer fence (UOF) value using the Box-and-Whiskers plot [TD00]. For each sub-interval d of set length (in our experiments $|d| = 1$ day) in V ’s active period, let P_d denote the set of positive reviews from H_V posted during d . If $|P_d| > UOF$, the RSD

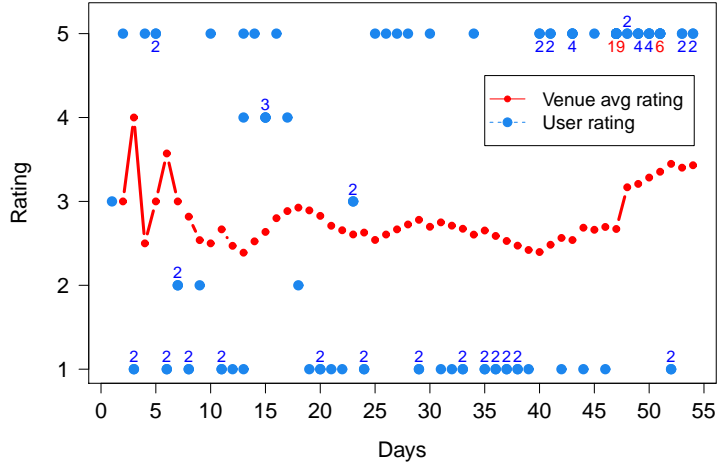


Figure 3.4: Evolution in time of the average rating of the venue “Azure Nail & Waxing Studio” of Chicago, IL, compared against the ratings assigned by its reviews. The values in parentheses denote the number of reviews that were assigned a corresponding rating (shown on the y axis) during one day. The lack of consensus between the many low and high rated reviews raises a red flag.

module marks P_d , i.e., a spike has been detected. For instance, the “South Bay BMW” venue (see Figure 3.3) has a UOF of 9 for positive reviews: any day with more than 9 positive reviews is considered to be a spike.

We note that a different empirical approach, proposed by Fei et al. [FML⁺13] is to use Kernel Density Estimation (KDE) to estimate the probability distribution function of the reviews of a venue.

The RSD module outputs two features (see Table 3.3): $SC(V)$, the number of spikes detected for a venue V , and $SAmp(V)$, the amplitude of the highest spike of V , normalized to the average number of reviews posted for V during an interval d .

Aggregate Rating Disparity (ARD).

A venue that is the target of a review campaign is likely to receive reviews that do not agree with its genuine reviews. Furthermore, following a successful review

campaign, the venue is likely to receive reviews from genuine users that do not agree with the venue’s newly engineered rating.

Let $\rho_V(T)$ denote the average rating of a venue V at time $T \in [T_1, T_c]$. We define the rating disparity of a review R written at time $R.\tau$ for V to be the divergence of R ’s rating from the average rating of V at the time of its posting, $|R.\rho - \rho_V(R.\tau)|$. Let R_1, \dots, R_N , $N = n + q$, be all the reviews received by V (both genuine and fraudulent) during its active interval ΔT . We define the aggregate rating disparity score of V to be the average rating disparity of all the reviews of V :

$$ARD(V) = \frac{\sum_{i=1}^N |R_i.\rho - \rho_V(R_i.\tau)|}{N}$$

By influencing the average rating of a venue, a review campaign will increase the rating disparity of both fraudulent and of genuine reviews. This is illustrated in Figure 3.4, that plots the evolution in time of the average rating against the ratings of individual reviews received by the “Azure Nail & Waxing Studio” (Chicago, IL). The positive reviews (1 day has a spike of 19, 5-star reviews, shown in red in the upper right corner) disagree with the low rated reviews, generating a high ARD value. The ARD module contributes one feature, the *ARD* score, see Table 3.3.

We note that Jindal and Liu [JL08], Lim et al. [LNJ⁺10], Mukherjee et al. [MLG12] and Mukherjee et al [MKL⁺13] proposed a feature similar to ARD. However, the ARD feature we introduce differs, in that the disparity is between the rating of a review and the rating of the venue at the time when the review was written. Previous work considers a formula where the disparity is computed at the present time.

Fraudulent Review Impact (FRI) Module.

Venues that receive few genuine reviews are particularly vulnerable to review campaigns (see also Theorem 1). Furthermore, long term review campaigns that post

Notation	Definition
$f(U)$	The number of friends of U
$r(U)$	The number of reviews written by U
$Exp_U(V)$	The expertise of U around V
$c_U(V)$	The number of check-ins of U at V
$p_U(V)$	The number of photos of U at V
$feedback(R)$	The feedback count of R
$Age_U(R)$	Age of U 's account when R was posted

Table 3.2: Features used to classify review R written by user U for venue V .

high numbers of fraudulent reviews can re-define the “normal” review posting behavior, flatten spikes and escape detection by the RSD module. They are also likely to drown the impact of genuine reviews on the aggregate rating of the venue. Thus, the ARD of the campaign’s target venue will be small, controlled by the fraudulent reviews.

We propose to detect such behaviors through fraudulent reviews that significantly impact the aggregate rating of venues. For this, in a first step, the FRI module uses machine learning tools to classify the reviews posted for V as either fraudulent or genuine. It uses features extracted from each review, its writer and the relation between the review writer and the target venue (see Table 3.2). Specifically, let R denote a review posted for a venue V , and let U denote the user who wrote it. In addition to the friend and review count of U , we introduce the concept of *expertise* of U around V . $Exp_U(V)$ is the number of reviews U wrote for venues in the vicinity (50 mile radius) of V . Furthermore, FRI uses the number of activities of U recorded at V , the feedback of R , counting the users who reacted positively to the review, and the age of U 's account when R was posted, $Age_U(R)$. Section 3.5.1 shows that the Random Forest tool achieves 94% accuracy when classifying fraudulent and genuine reviews.

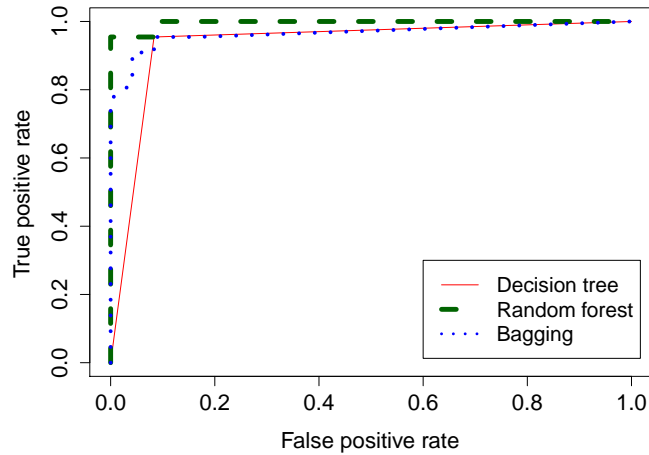


Figure 3.5: ROC plot of Random Forest (RF), Bagging and C4.5 Decision Tree (DT) for review classification (426 fraudulent, 410 genuine). RF performs best, at 93.83% accuracy.

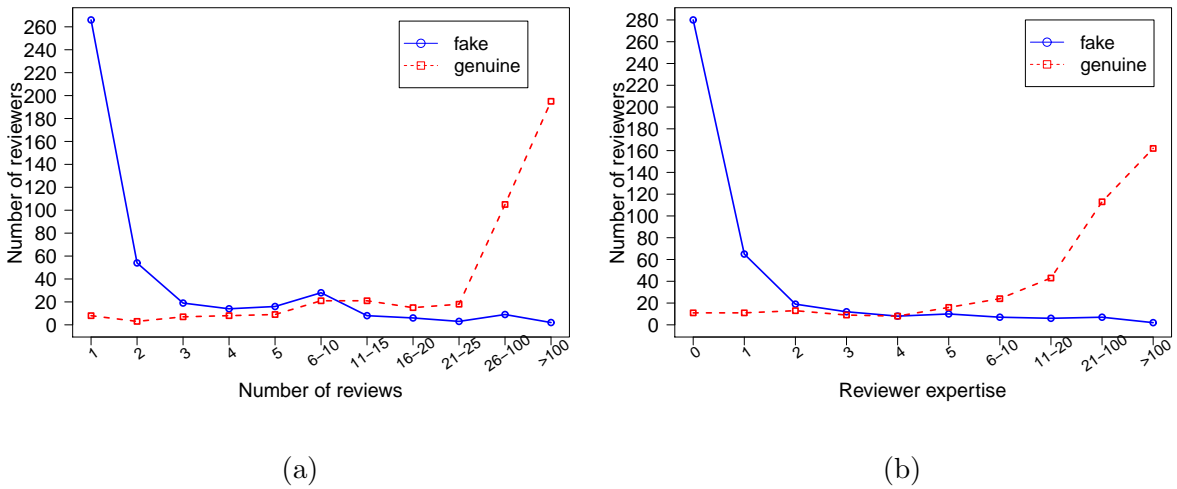


Figure 3.6: (a) Distribution of reviewers' review count: fraudulent vs. genuine review sets. (b) Distribution of reviewers' expertise levels: fraudulent vs. genuine review sets. Note their symmetry: unlike genuine reviewers, fraudulent reviewers tend to have written only few reviews and have low expertise for the venues that they reviewed.

Notation	Definition
$SC(V)$	The number of review spikes for V
$SAmp(V)$	The amplitude of the highest spike
$ARD(V)$	Aggregate rating disparity
$FRI(V)$	The fraudulent review impact of V
$CF(V)$	Count of reviews classified fraudulent
ρ_V	The rating of V
N	The number of reviews of V
$cir(V)$	The number of reviews with check-ins
$pr(V)$	The number of reviews with photos
$Age(V)$	The age of V

Table 3.3: Features used to classify a venue V as either deceptive or legitimate.

In a second step, the FRI module introduces the notion of *fraudulent review impact*, to model the impact of fraudulent reviews on the final rating of the venue. Let $\rho_V^g = \frac{\sigma}{n}$ denote the genuine rating of V , computed as an average over its n genuine reviews. Then, $FRI(V) = \rho_V(T_c) - \rho_V^g$, where $\rho_V(T_c)$ is the average rating of V at current time T_c . Note that $FRI(V)$ can be negative, for a bad-mouthing campaign. The FRI module contributes two features, $FRI(V)$, and the percentage of reviews classified as fraudulent for V , $CF(V)$ (see Table 3.3).

Venue Classification.

In addition to the features provided by the RSD, ARD and FRI modules, we also use the rating of V , ρ_V , its number of reviews N , its number of reviews with associated user check-ins, $cir(V)$, and with uploaded photos, $pr(V)$, and the current age of V , $Age(V)$, measured in months since V 's first review. Table 3.3 lists all the features we selected. Section 3.5.2 shows that the features enable the Random Forest classifier to achieves 95.8% accuracy when classifying the venue sets of Section 3.3.1.

3.4.2 App Fraud Detection

FairPlay Overview.

FairPlay organizes the analysis of longitudinal app data into the following 4 modules, illustrated in Figure 3.7. The Review Feedback (RF) module exploits feedback left by genuine reviewers, while the Inter Review Relation (IRR) module leverages relations between reviews, ratings and install counts. The Jekyll-Hyde (JH) module monitors app permissions, with a focus on dangerous ones, to identify apps that convert from benign to malware. Each module produces several features that are used to train an app classifier. FairPlay further uses general features such as the app’s average rating, total number of reviews, ratings and installs, for a total of 28 features. In the following, we detail each module and the features it extracts.

The Co-Review Graph (CoReG) Module.

Let the *co-review graph* of an app be a graph where nodes correspond to users who reviewed the app, and undirected edges have a weight that indicates the number of apps reviewed in common by the edge’s endpoint users. We seek to identify cliques in the co-review graph.

To address the problem’s NP-hardness, we exploit two observations. First, fraudsters hired to review an app are likely to post those reviews within relatively short time intervals (e.g., days). Second, perfect cliques are not necessary. Instead, we relax this requirement to identify “pseudo cliques”, or groups of highly but not necessarily completely connected nodes. Specifically, we use the weighted density definition of Uno [Uno07]: given a co-review graph, its weighted density $\rho = \frac{\sum_{e \in E} w(e)}{\binom{n}{2}}$, where E denotes the graph’s edges and n its number of nodes (reviews). We are

interested then in subgraphs of the co-review graph whose weighted density exceeds a threshold value θ .

CoReG features. CoReG extracts the following features (i) the number of cliques whose density equals or exceeds θ , (ii) the maximum, median and standard deviation of the densities of identified pseudo cliques, (iii) the maximum, median and standard deviation of the node count of identified pseudo cliques, normalized by n (the app’s review count), and (iv) the total number of nodes of the co-review graph that belong to at least one pseudo clique, normalized by n .

Reviewer Feedback (RF) Module.

Reviews written by genuine users of malware and fraudulent apps may describe negative experiences. The RF module exploits this observation through a two step approach: (i) detect and filter out fraudulent reviews, then (ii) identify malware and fraud indicative feedback from the remaining reviews.

Step RF.1: Fraudulent review filter. We posit that users that have higher expertise on apps they review, have written fewer reviews for apps developed by the same developer, have reviewed more paid apps, are more likely to be genuine. We exploit this conjecture to use supervised learning algorithms trained on the following features, defined for a review R written by user U for an app A :

- *Reviewer based features.* The *expertise* of U for app A , defined as the number of reviews U wrote for apps that are “similar” to A , as listed by Google Play (see § 3.2). The *bias* of U towards A : the number of reviews written by U for other apps developed by A ’s developer. In addition, we extract the total money paid by U on apps it has reviewed, the number of apps that U has liked, and the number of Google+ followers of U .

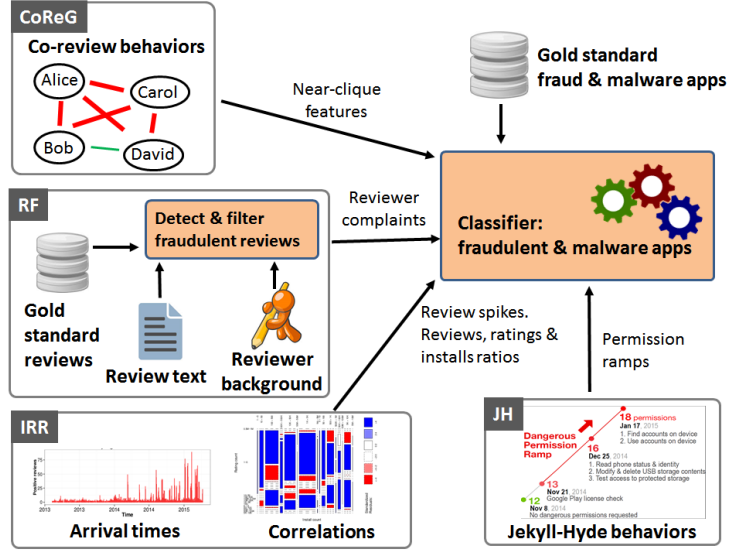


Figure 3.7: FairPlay system architecture. The CoReG module identifies suspicious, time related co-review behaviors. The RF module uses linguistic tools to detect suspicious behaviors reported by genuine reviews. The IRR module uses behavioral information to detect suspicious apps. The JH module identifies permission ramps to pinpoint possible Jekyll-Hyde app transitions.

- *Text based features.* We used the NLTK library [BKL09] and the Naive Bayes classifier, trained on two datasets: (i) 1,041 sentences extracted from randomly selected 350 positive and 410 negative Google Play reviews, and (ii) 10,663 sentences extracted from 700 positive and 700 negative IMDB movie reviews [PLV02]. 10-fold cross validation of the Naive Bayes classifier over these datasets reveals a FNR of 16.1% and FPR of 19.65%. We used the trained Naive Bayes classifier to determine the statements of R that encode positive and negative sentiments. We then extracted the following features: (i) the percentage of statements in R that encode positive and negative sentiments respectively, and (ii) the rating of R and its percentile among the reviews written by U .

Step RF.2: Reviewer feedback extraction. We conjecture that (i) since no app is perfect, a “balanced” review that contains both app positive and negative sentiments is more likely to be genuine, and (ii) there should exist a relation between

the review’s dominating sentiment and its rating. Thus, after filtering out fraudulent reviews, we extract feedback from the remaining reviews. For this, we have used NLTK to extract 5,106 verbs, 7,260 nouns and 13,128 adjectives from the 97,071 reviews we collected from the 613 gold standard apps (see § 3.3.2). We used these words to manually identify lists of words indicative of malware, fraudulent and benign behaviors. Our malware indicator word list contains 31 words (e.g., risk, hack, corrupt, spam, malware, fake, fraud, blacklist, ads). The fraud indicator word list contains 112 words (e.g., cheat, hideous, complain, wasted, crash) and the benign indicator word list contains 105 words.

RF features. We extract 3 features, denoting the percentage of genuine reviews that contain malware, fraud, and benign indicator words respectively. We also extract the *impact* of detected fraudulent reviews on the overall rating of the app: the absolute difference between the app’s average rating and its average rating when ignoring all the fraudulent reviews.

Inter-Review Relation (IRR) Module.

This module leverages temporal relations between reviews, as well as relations between the review, rating and install counts of apps, to identify suspicious behaviors.

Temporal relations. We detect outliers in the number of daily reviews received by an app. We identify days with spikes of positive reviews as those whose number of positive reviews exceeds the upper outer fence of the box-and-whisker plot built over the app’s numbers of daily positive reviews.

Reviews, ratings and install counts. We used the Pearson’s χ^2 test to investigate relationships between the install and rating counts of the 87K new apps, at the end of the collection interval. We grouped the rating count in buckets of the same size as Google Play’s install count buckets. Figure 3.9 shows the mosaic plot

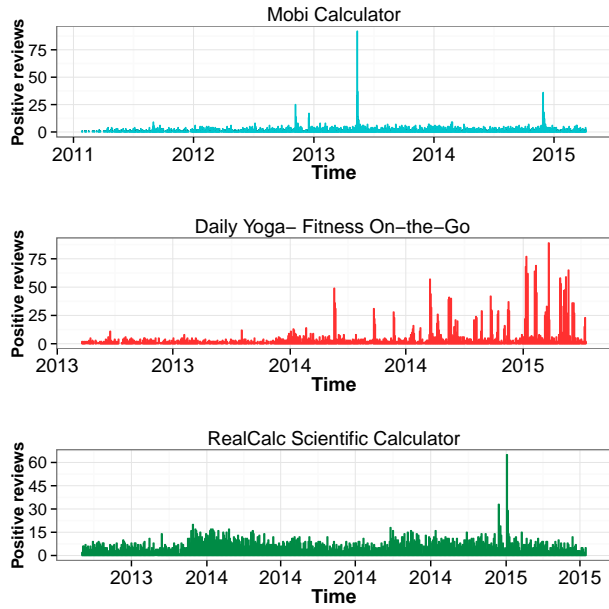


Figure 3.8: Timelines of positive reviews for 3 apps from the fraudulent app dataset.

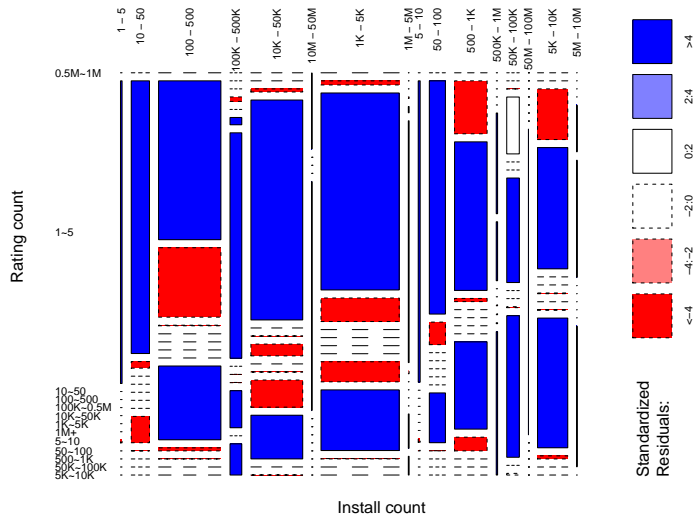


Figure 3.9: Mosaic plot of install vs. rating count relations of the 87K apps. Larger rectangles signify that more apps have the corresponding rating and install count range; dotted lines mean no apps in a certain install/rating category. The standardized residuals identify the cells that contribute the most to the χ^2 test. The most significant rating:install ratio is 1:100.

of the relationships between rating and install counts. $p=0.0008924$, thus we conclude dependence between the rating and install counts. We leverage this result to conjecture that adversaries that post fraudulent ratings and reviews, or create fake app install events, may break a natural balance between their counts.

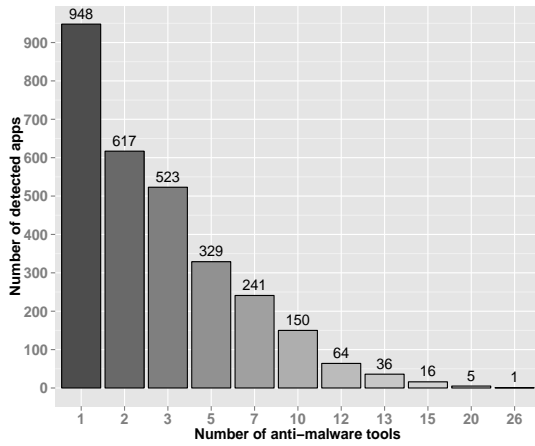
IRR features. We extract temporal features: the number of days with detected spikes and the maximum amplitude of a spike. We also extract (i) the ratio of installs to ratings as two features, I_1/Rt_1 and I_2/Rt_2 and (ii) the ratio of installs to reviews, as I_1/Rv_1 and I_2/Rv_2 . $(I_1, I_2]$ denotes the install count interval of an app, $(Rt_1, Rt_2]$ its rating interval and $(Rv_1, Rv_2]$ its (genuine) review interval.

Jekyll-Hyde App Detection (JH) Module.

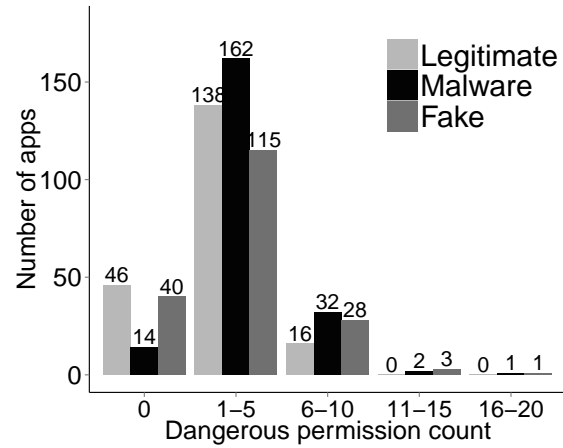
Android’s API level 22 labels 47 permissions as “dangerous”. Figure 3.10(b) compares the distributions of the number of dangerous permissions requested by the gold standard malware, fraudulent and benign apps. The most popular dangerous permissions among these apps are “modify or delete the contents of the USB storage”, “read phone status and identity”, “find accounts on the device”, and “access precise location”. While most benign apps request at most 5 such permissions, some malware and fraudulent apps request more than 10.

Upon manual inspection of several apps, we identified a new type of malicious intent possibly perpetrated by deceptive app developers: apps that seek to attract users with minimal permissions, but later request dangerous permissions. The user may be unwilling to uninstall the app “just” to reject a few new permissions. We call these *Jekyll-Hyde apps*. Figure 3.10(c) shows the dangerous permissions added during different version updates of one gold standard malware app.

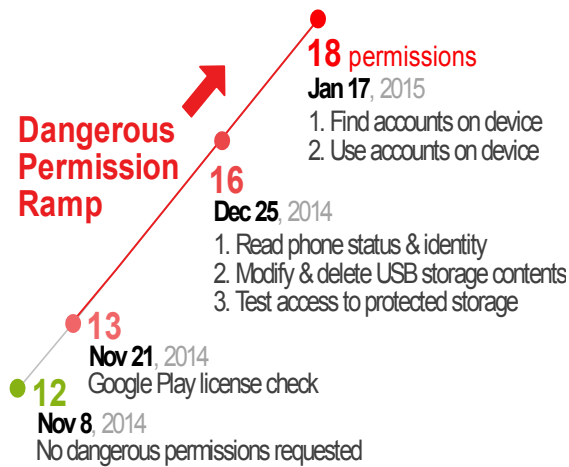
JH features. We extract the following features, (i) the total number of permissions requested by the app, (ii) its number of dangerous permissions, (iii) the app’s number



(a)



(b)



(c)

Figure 3.10: (a) Apks detected as suspicious (y axis) by multiple anti-virus tools (x axis), through VirusTotal [Vir15], from a set of 7,756 downloaded apks. (b) Distribution of the number of “dangerous” permissions requested by malware, fraudulent and benign apps. (c) Dangerous permission ramp during version updates for a sample app “com.battery.plusfree”. Originally the app requested no dangerous permissions.

of dangerous permission ramps, and (iv) its total number of dangerous permissions added over all the ramps.

3.5 Empirical Evaluation

In this section we show that Marco is scalable as well as efficient in detecting fraudulent reviews and deceptive venues. We have implemented Marco using (i) Python, to extract data from parsed pages and compute the proposed features, (ii) the statistical tool R, to classify reviews and venues. We used MySQL to store collected data and features.

We have implemented FairPlay using Python to extract data from parsed pages and compute the features, and the R tool to classify reviews and apps. We have set the threshold density value θ to 3, to detect even the smaller pseudo cliques.

We have used the Weka data mining suite [Wek] to perform the experiments, with default settings. We experimented with multiple supervised learning algorithms. Due to space constraints, we report results for the best performers: Multi-Layer Perceptron (MLP) [Gal90], Decision Trees (DT) (C4.5) and Random Forest (RF) [Bre01], using 10-fold cross-validation [Koh95a]. We use the term “positive” to denote a fraudulent review, fraudulent or malware app; FPR means *false positive rate*. Similarly, “negative” denotes a genuine review or benign app; FNR means *false negative rate*.

3.5.1 Review Classification in Yelp.

We investigated the ability of the FRI module to classify reviews, when using 5 machine learning tools: Bagging, k -Nearest Neighbor (kNN), Random Forest (RF), Support Vector Machines (SVM) and C4.5 Decision Trees (DT). We used 10-fold

Classifier	TPR(%)	FPR(%)	FNR(%)	Acc(%)
Random Forest	94.71	7.0	5.29	93.83
Bagging	93.45	6.28	6.55	93.59
Decision tree	91.44	5.07	8.56	93.22
SVM	89.92	9.66	6.04	92.11

Table 3.4: Review classification: comparison of machine learning algorithms. RF performs best, at 93.83% accuracy.

Compared Classifiers	χ^2 value	p-value
Bagging-DT	11.6452	0.0006437
RF-DT	13.5	0.0002386
Bagging-RF	0.0476	0.8273
RF-SVM	4.8983	0.0268
Bagging-SVM	5.2258	0.0222
DT-SVM	5.1142	0.0237

Table 3.5: Significance test: pairwise comparison of machine learning algorithms using McNemar’s test. With the exception of the (Bagging, RF) pair, for all other pairs McNemar’s test produces a χ^2 value with 1 degree of freedom, highly significant with a confidence level of more than 95.0%.

cross-validation over the fraudulent and 410 genuine reviews of Section 3.3.1. Figure 3.5 shows the receiver operating characteristic (ROC) curve for the top 3 performers: RF, Bagging and DT.

The overall accuracy ($\frac{TPR+TNR}{TPR+TNR+FPR+FNR}$) of RF, Bagging and DT is 93.8%, 93.6% and 93.2% respectively. TPR is the true positive rate, TNR is the true negative rate, FPR the false positive rate and FNR the false negative rate. The (FPR, FNR) pairs for RF, Bagging and DT are (7.0%,5.3%),(6.3%,6.6%) and (5.1%,8.6%) respectively (shown in table 3.4). In the remaining experiments, the FRI module of Marco uses the RF classifier.

The top 2 most impactful features for RF are $r(U)$ and $Exp_U(V)$. Figure 3.6(a) compares the distribution of the $r(U)$ feature for the 426 fraudulent and the 410 genuine reviews. We emphasize their symmetry: few fraudulent review writers

posted a significant number of reviews, while few genuine review writers posted only a few reviews. Figure 3.6(b) compares the distribution of the $Exp_U(V)$ measure. The distributions are also almost symmetric: most writers of genuine reviews have written at least 4 reviews for other venues in the vicinity of the venue of their selected review.

Furthermore, we tested the null hypothesis that the classifiers used in review classification are equivalent i.e. the difference in performance metrics of different classifiers is not significant. As the classifiers are trained and tested on the same dataset, we used McNemar’s test which tabulates the outcomes of every two classifiers used for review classification. The results are shown in Table 3.5. With the exception of the test that compares Bagging and RF, all other tests produce a χ^2 value with 1 degree of freedom, highly significant with a confidence level of more than 95.0% (the p -value is <0.05). Thus, we reject the null hypothesis, which means that the differences in performance metrics of DT, RF, Bagging and SVM models are statistically significant.

3.5.2 Venue Classification in Yelp.

We have used 10-fold cross-validation to evaluate the ability of Marco to classify the 90 deceptive and 100 legitimate venues of Section 3.3.1. Figure 3.11 shows the ROC curve for Marco when using the RF, Bagging and C4.5 DT classifiers on the features listed in Table 3.3. The overall accuracy for RF, Bagging and DT is 95.8%, 93.7% and 95.8% respectively, with the corresponding (FPR,FNR) pairs being (5.55%,3%),(8.88%,4%) and (5.55%,3%) respectively.

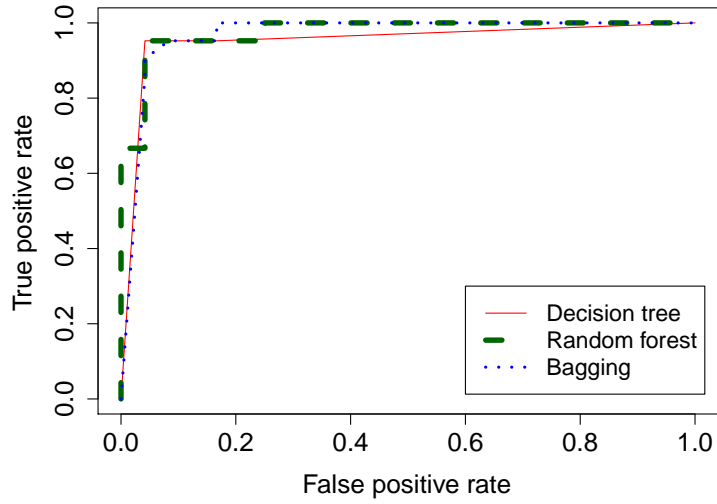


Figure 3.11: ROC plot of RF, Bagging and C4.5 DT for the 90 deceptive/100 legitimate venue datasets. RF and DT are tied for best accuracy, of 95.8%.

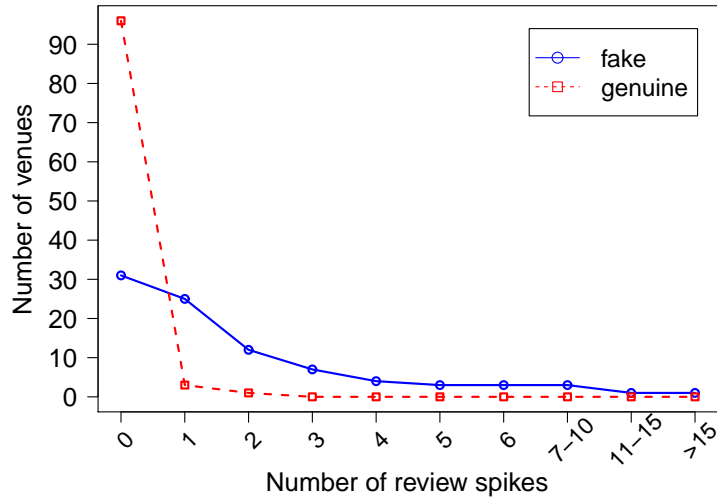


Figure 3.12: Distribution of $SC(V)$, for the 90 deceptive and 100 legitimate venues. 60 deceptive venues have at least one review spike. 1 legitimate venue has 1 spike.

Strategy	FPR	FNR	Accuracy
Marco/RF	5/90 = 0.055	3/100 = 0.3	95.8%
<i>avg</i> Δ	33/90 = 0.36	31/100 = 0.31	66.3%
<i>dist</i> Φ	28/90 = 0.31	25/100 = 0.25	72.1%
<i>peak</i> \uparrow	41/90 = 0.45	37/100 = 0.37	58.9%

Table 3.6: Marco vs. the three deceptive venue detection strategies of Feng et al. [FXGC12]. Marco shows over 23% accuracy improvement over *dist* Φ .

Classifier	TPR(%)	FPR(%)	FNR(%)	Acc(%)
Random Forest	96.07	20.0	3.92	94.64
Bagging	94.12	20.0	5.88	92.15
Decision tree	94.12	0.0	5.88	94.64

Table 3.7: Marco performance on new, unpopular venues: comparison of machine learning algorithms. RF and DT perform the best.

Figure 3.12 shows the distribution of $SC(V)$ for the 190 venues. Only 1 legitimate venue has a review spike, while several deceptive venues have more than 10 spikes. Furthermore, 26 deceptive venues have an FRI score larger than 1; only 1 legitimate venue has an FRI larger than 1.

Comparison with state-of-the-art. We compared Marco with the three deceptive venue detection strategies of Feng et al. [FXGC12], *avg* Δ , *dist* Φ and *peak* \uparrow . Table 3.6 shows the FPR, FNR and overall accuracy of Marco, *avg* Δ , *dist* Φ and *peak* \uparrow . Marco achieves a significant accuracy improvement (95.8%) over *dist* Φ , the best strategy of Feng et al. [FXGC12] (72.1%).

Marco performance for new venues. We have also evaluated the performance of Marco to classify relatively new venues with few genuine reviews. Specifically, from our set of 90 deceptive and 100 genuine reviews, we selected 51 deceptive and 5 genuine venues that had less than 10 genuine reviews when we collected them. The overall accuracy of RF, Bagging and DT on these 56 venues is 94.64%, 92.15%

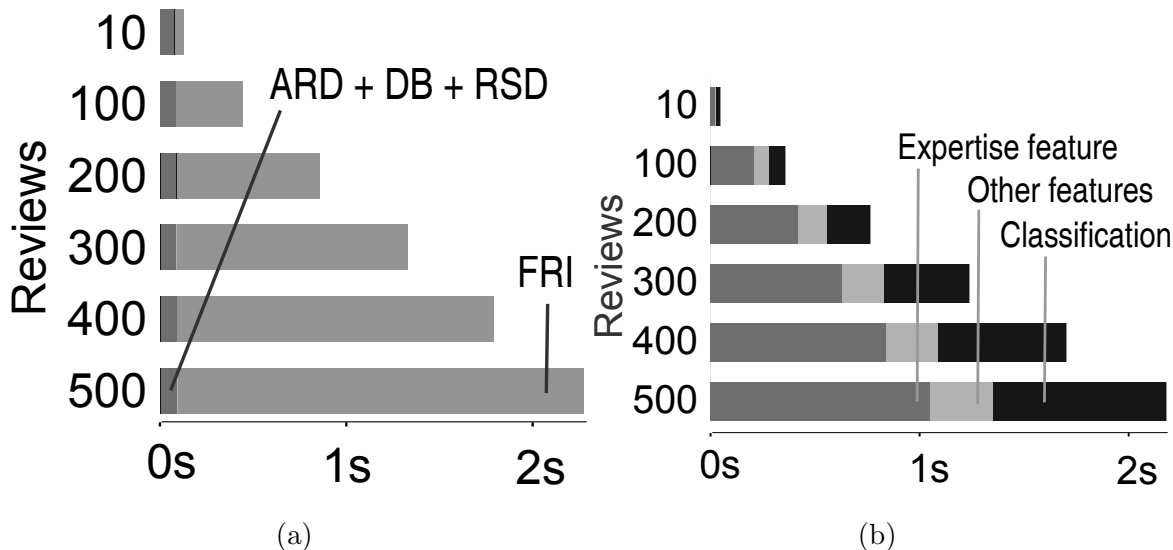


Figure 3.13: (a) Marco’s per-module overhead: FRI is the most expensive, but under 2.3s even for venues with 500 reviews. (b) Zoom-in of FRI module overhead. Computing the $Exp_U(V)$ feature takes the most time.

and 94.64% respectively. The (FPR, FNR) pairs for RF, Bagging and DT are (20.0%,3.92%),(20.0%,5.88%) and (0.0%,5.88%) respectively.

3.5.3 Marco in the Wild.

Marco takes only a few seconds to classify a venue, on a i5@2.4GHz, 4GB of RAM Dell laptop. Figure 3.13(a) shows the per-module overhead of Marco (averages over 10 experiment runs), as a function of the review count of the venue classified. While the FRI module is the most time consuming, even for venues with 500 reviews the FRI overhead is below 2.3s. The RSD and ARD modules impose only a few ms (6ms for 500 reviews), while DB access and data retrieval take around 90ms. Figure 3.13(b) zooms-in into the FRI overhead. For 500 reviews, the most time consuming components are computing the user expertise, $Exp_U(V)$ ($\approx 1.1s$), computing *all* the other features ($\approx 0.4s$) and classifying the reviews ($\approx 0.8s$).

City	Car Shop	Mover	Spa
Miami, FL	1000 (6)	348 (8)	1000 (21)
San Fran., CA	612 (59)	475 (45)	1000 (42)
NYC, NY	1000 (8)	1000 (27)	1000 (28)

Table 3.8: Collected venues organized by city and venue type. Values between parentheses show the number of venues detected by Marco to be deceptive. San Francisco has the highest percentage of deceptive venues.

In order to understand the ability of Marco to perform well when trained on small sets, we have trained it on 50 deceptive and 50 legitimate venues and we have tested it on the remaining 40 deceptive and 50 legitimate venues. On average over 10 random experiments, Marco achieved an FPR of 6.25% and an FNR of 3%.

We have used Marco to classify the 7,435 venues we collected from Miami, San Francisco and New York City. We have divided the set of 7,435 venues into subsets of 200 venues. We trained Marco on the 190 ground truth/gold standard venues and tested it separately on all subsets of 200 venues. Table 3.8 shows the total number of venues collected and the number of venues detected to be deceptive, between parentheses. San Francisco has the highest concentration of deceptive venues: Marco flags almost 10% of its car repair and moving companies as suspicious, and upon our manual inspection, they indeed seemed to engage in suspicious review behaviors. While the FRI of San Francisco’s collected genuine venues is at most 1, 60% of its deceptive venues have an FRI between 1 and 4.

3.5.4 Detecting Yelp Campaigns

We conjecture that Yelp events can be used as review campaigns. Our hypothesis is based on several observations. First, the process of choosing the venues hosting Yelp events is not public. Second, a venue hosting an event is given ample warning to

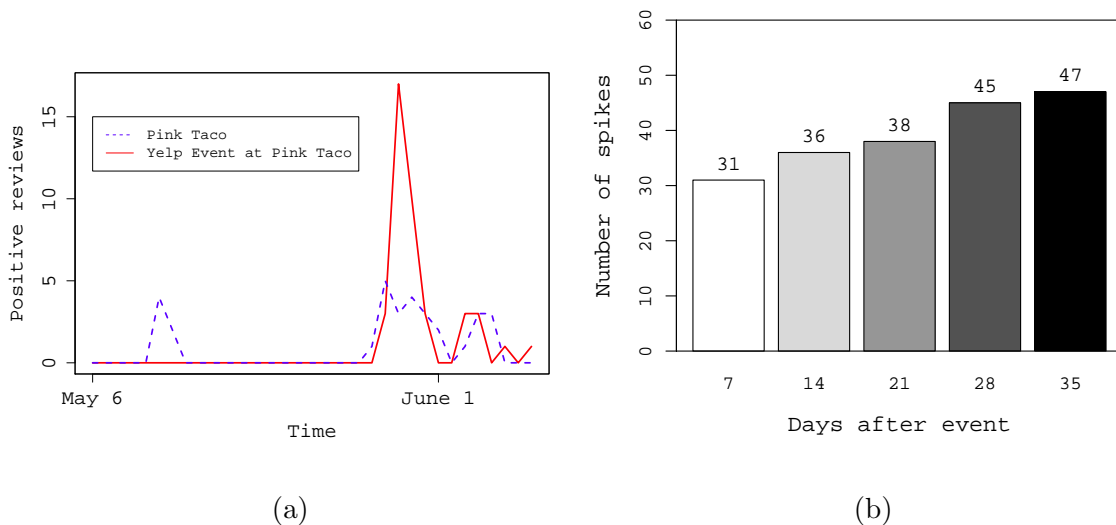


Figure 3.14: (a) The timeline of “Pink Taco 2” (Los Angeles) and of the Yelp event for this venue. Note the correlation between the two. (b) Yelp events: Positive review spike count as a function of ΔT .

organize the event. Third, only Elite yelpers attend this event. While the attendees are encouraged to review the event’s Yelp account, we have identified Yelp events that impacted the ratings of the corresponding host venues. We call such events, *Yelp campaigns*. Figure 3.14(a) shows an example of venue and event timelines, correlated in time, for the venue “Pink Taco 2” (Los Angeles). Note how the venue’s latest two spikes coincide with the spikes of the event.

To detect the correlation between Yelp events and increased review activity concerning the venues hosting the events, we use Marco’s RSD module as follows. Specifically, given a Yelp event and a time interval ΔT (system parameter), we determine if the hosting venue experiences a positive review spike within an interval ΔT of the event’s date.

For the events and hosting venues collected (see Section 3.3.1), Figure 3.14(b) plots the number of positive review spikes detected within ΔT days, when ΔT ranges

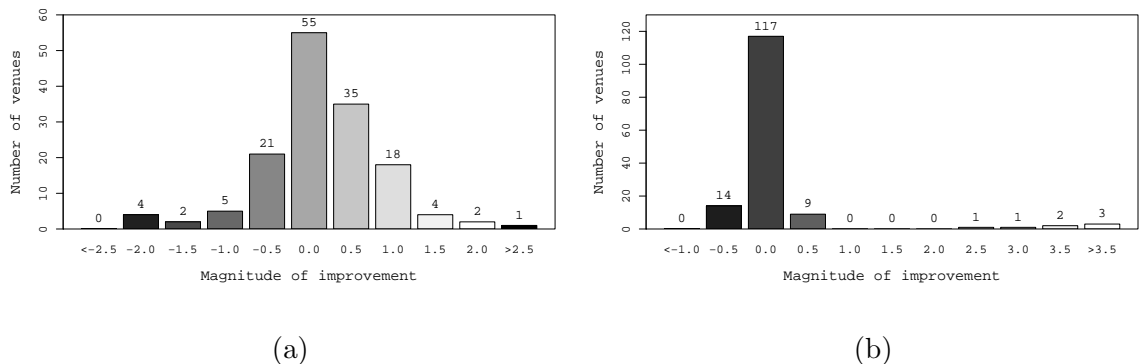


Figure 3.15: (a) Distribution of the short term impact (2 weeks) of Yelp events on venue ratings. (b) Yelp events: Distribution of the improvement due to Elite events.

from 1 to 5 weeks. For instance, when ΔT is 14 days, Marco detected 36 spikes on the 149 venues. Some venues have more than one spike within the 14 days. The total number of venues with at least one spike is 24, accounting for around 17% of the venues. While for $\Delta T = 35$ Marco detected 47 spikes, we prefer a shorter interval: the correlation between the event and spikes may fade over longer intervals. In the following we use $\Delta T=14$.

Furthermore, we focused on determining the influence of Yelp events on the overall rating of a venue. First, we computed the *2-week impact* of the Yelp event on the venue. We define the 2-week impact as the difference between the rating of the venue two weeks after the event and the rating of the venue before the event. We compute the rating of a venue at any given time T as the average over the ratings of all the reviews received by the venue before time T . Figure 3.15(a) shows the distribution of the 2-week impact of the Yelp event on the venue. While 55 (of the 149) venues show no impact, 60 venues show at least a 0.5 star improvement, with 3 at or above 2 star improvements. 32 venues are negatively impacted. Thus, almost twice as many venues benefit from Yelp events, when compared to those showing a rating decay.

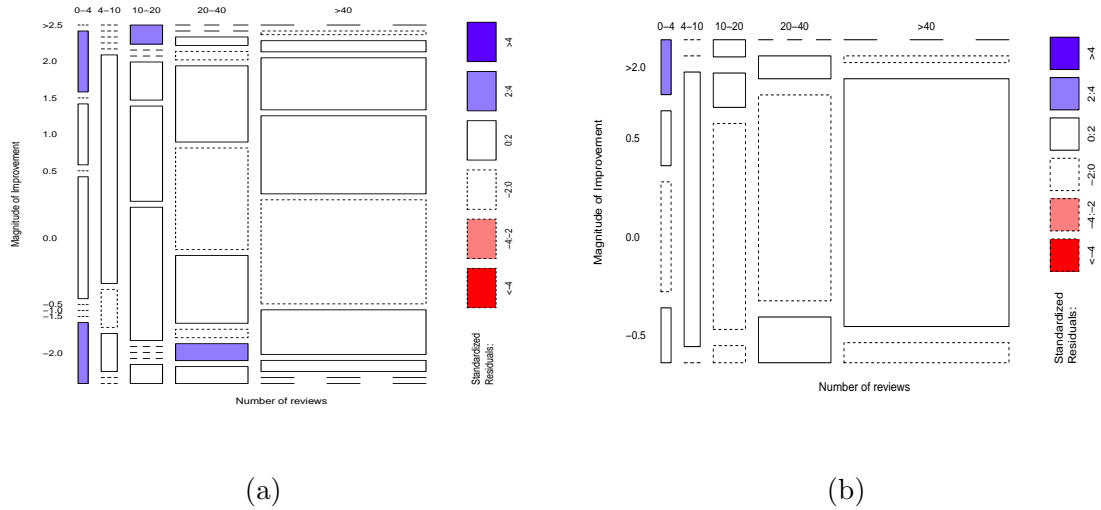


Figure 3.16: Mosaic plots: The standardized residuals indicate the importance of the rectangle in the χ^2 test. (a) The dependency between the short term rating change of venues due to events and their number of reviews. (b) The dependency between the long term rating change of venues due to events and their number of reviews.

This result raises the question of whether there exists a relation between the number of reviews of a venue and the short term impact an event has on the venue. The impact of an event is a categorical variable, as it is quantified with fractions of a star (integer). The number of reviews however is a discrete variable. Therefore, we cannot use methods for linear or non-linear association, e.g. correlation coefficient. Instead, we tested the hypothesis of independence between the rating impact and the number of reviews, using a χ^2 test [TD00]. The test produced a $\chi^2 = 58.6837$ with 36 degrees of freedom, which is highly significant (the p -value is 0.009854). Thus, we reject the independence hypothesis.

Figure 3.16(a) shows the mosaic plot depicting this relation. Each rectangle corresponds to a set of venues, that have a certain review count range (the x axis) and having been impacted by a certain measure within two weeks of an event (the y axis). The shape and size of each rectangle depict the contribution of the corresponding variables, so a large rectangle means a large count in the contingency

table. Blue rectangles indicate that they are more than two standard deviations above the expected counts. Then, the figure shows that more than half of the (149) venues have more than 40 reviews. Moreover, we notice that the venues having more than 40 reviews set the trend of Figure 3.15(a): while roughly one third of the venues show no impact, twice as many venues show a positive impact vs. a negative one.

Second, we study the long term impact of Yelp events. For this, we compare the current ratings of the 149 venues with their ratings before the events. Figure 3.15(b) shows the distribution (over the 149 venues) of the difference between the current rating of the venues and their rating before the events. 78% of venues show no improvement. Furthermore, we see a balance between the number of venues showing an improvement versus a negative impact (16 positive vs. 14 negative). However, we emphasize that the negative impact is only half a star, while the positive impact reaches up to 3.5 stars.

We conducted a χ^2 test to verify the dependence of the long term impact of events on venues on the number of ratings of the venues. The test was highly significant with $\chi^2 = 29.2038$, 12 degrees of freedom and a p -value of 0.003674. Figure 3.16(b) shows the mosaic plot: a vast majority of the venues having more than 40 reviews have no impact on the long term. This shows that review spikes have a smaller impact on constantly popular venues.

3.5.5 Review Classification in Google Play.

To evaluate the accuracy of FairPlay’s fraudulent review detection component (RF module), we used the gold standard datasets of fraudulent and genuine reviews of § 3.3.2. We used GPCrawler to collect the data of the writers of these reviews,

Strategy	FPR%	FNR%	Accuracy%
DT (Decision Tree)	2.46	6.03	95.98
MLP (Multi-layer Perceptron)	1.47	6.67	96.26
RF (Random Forest)	2.46	5.40	96.26

Table 3.9: Review classification results (10-fold cross-validation), of gold standard fraudulent (positive) and genuine (negative) reviews. MLP achieves the lowest false positive rate (FPR) of 1.47%.

Strategy	FPR%	FNR%	Accuracy%
FairPlay/DT	3.01	3.01	96.98
FairPlay/MLP	1.51	3.01	97.74
FairPlay/RF	1.01	3.52	97.74

Table 3.10: FairPlay classification results (10-fold cross validation) of gold standard fraudulent (positive) and benign apps. RF has lowest FPR, thus desirable [CNW⁺11].

including the 203 reviewers of the 406 fraudulent reviews (21,972 reviews for 2,284 apps) and the 315 reviewers of the genuine reviews (9,468 reviews for 7,116 apps). Table 3.9 shows the results of the 10-fold cross validation of algorithms classifying reviews as genuine or fraudulent. To minimize wrongful accusations, we seek to minimize the FPR [CNW⁺11]. MLP simultaneously achieves the highest accuracy of 96.26% and the lowest FPR of 1.47% (at 6.67% FNR). Thus, in the following experiments, we use MLP to filter out fraudulent reviews in the RF.1 step.

3.5.6 App Classification in Google Play

To evaluate FairPlay, we have collected all the 97,071 reviews of the 613 gold standard malware, fraudulent and benign apps, written by 75,949 users, as well as the 890,139 apps rated or played by these users.

Fraud Detection Accuracy. Table 3.10 shows 10-fold cross validation results of FairPlay on the gold standard fraudulent and benign apps (see § 3.3.2). All

Strategy	FPR%	FNR%	Accuracy%
FairPlay/DT	4.02	4.25	95.86
FairPlay/MLP	4.52	4.72	95.37
FairPlay/RF	1.51	6.13	96.11
Sarma et al. [SLG ⁺ 12]/SVM	65.32	24.47	55.23

Table 3.11: FairPlay classification results (10-fold cross validation) of gold standard malware (positive) and benign apps, significantly outperforming Sarma et al. [SLG⁺12]. FairPlay’s RF achieves 96.11% accuracy at 1.51% FPR.

classifiers achieve accuracies of around 97%. Random Forest is the best, having the highest accuracy of 97.74% and the lowest FPR of 1.01%.

Malware Detection Accuracy. We have used Sarma et al. [SLG⁺12]’s solution as a baseline to evaluate the ability of FairPlay to accurately detect malware. We computed Sarma et al. [SLG⁺12]’s RCP and RPCP indicators (see § 2.1.3) using the longitudinal app dataset. We used the SVM based variant of Sarma et al. [SLG⁺12], which performs best. Table 3.10 shows 10-cross validation results over the malware and benign gold standard sets. FairPlay significantly outperforms Sarma et al. [SLG⁺12]’s solution, with an accuracy that consistently exceeds 95%. Random Forest has the smallest FPR of 1.51% and the highest accuracy of 96.11%. This is surprising: most FairPlay features are meant to identify search rank fraud, yet they *also* accurately identify malware.

Is Malware Involved in Fraud? We conjectured that the above result is due in part to malware apps being involved in search rank fraud. To verify this, we have trained FairPlay on the gold standard benign and fraudulent app datasets, then we have tested it on the gold standard malware dataset. MLP is the most conservative algorithm, discovering 60.85% of malware as fraud participants. Random Forest discovers 72.15%, and Decision Tree flags 75.94% of the malware as fraudulent. This result confirms our conjecture and shows that search rank fraud detection can be an important addition to mobile malware detection efforts.

3.5.7 FairPlay on the Field.

We have also evaluated FairPlay on non “gold standard” apps. For this, we have collected a set of apps, as follows. First, we selected 8 app categories: Arcade, Entertainment, Photography, Simulation, Racing, Sports, Lifestyle, Casual. We have selected the 6,300 apps from the longitudinal dataset of the 87K apps, that belong to one of these 8 categories, and that have more than 10 reviews. From these 6,300 apps, we randomly selected 200 apps per category, for a total of 1,600 apps. We have then collected the data of all their 50,643 reviewers (not unique) including the ids of all the 166,407 apps they reviewed.

We trained FairPlay with Random Forest (best performing on previous experiments) on all the gold standard benign and fraudulent apps. We have then run FairPlay on the 1,600 apps, and identified 372 apps (23%) as fraudulent. The Racing and Arcade categories have the highest fraud densities: 34% and 36% of their apps were flagged as fraudulent.

Intuition. During the 10-fold cross validation of FairPlay for the gold standard fraudulent and benign sets, the top most impactful features for the Decision Tree classifier were (i) the percentage of nodes that belong to the largest pseudo clique, (ii) the percentage of nodes that belong to at least one pseudo clique, (iii) the percentage of reviews that contain fraud indicator words, and (iv) the number of pseudo clique with $\theta \geq 3$.

While not plotted here due to space constraints, we note that around 75% of the 372 fraudulent apps have at least 20 fraud indicator words in their reviews.

3.5.8 Coercive Campaign Apps in Google Play.

Upon close inspection of apps flagged as fraudulent by FairPlay, we identified apps perpetrating a new attack type. The apps, which we call *coercive campaign apps*, harass the user to either (i) write a positive review for the app, or (ii) install and write a positive review for other apps (often of the same developer). In return, the app rewards the user by, e.g., removing ads, providing more features, unlocking the next game level, boosting the user’s game level or awarding game points.

We found evidence of coercive campaign apps from users complaining through reviews, e.g., “I only rated it because i didn’t want it to pop up while i am playing”, or “Could not even play one level before i had to rate it [...] they actually are telling me to rate the app 5 stars”.

We leveraged this evidence to identify more coercive campaign apps from the longitudinal app set. Specifically, we have first manually selected a list of potential keywords indicating coercive apps (e.g., “rate”, “download”, “ads”). We then searched all the 2,850,705 reviews of the 87K apps and found around 82K reviews that contain at least one of these keywords. Due to time constraints, we then randomly selected 3,000 reviews from this set, that are not flagged as fraudulent by FairPlay’s RF module. Upon manual inspection, we identified 118 reviews that report coercive apps, and 48 apps that have received at least 2 such reviews. We leave a more thorough investigation of this phenomenon for future work.

3.6 Limitations

Our collected dataset in Yelp from Miami, San Francisco and New York City only consists of 7435 venues, their 270,121 reviews and 195,417 reviewer profiles. This dataset was not chosen randomly so that it can represent the entire dataset. Our

classification results using FairPlay to detect fraudulent reviews, malwares and fraud apps are based on our gold-standard dataset which consist of only hundreds of reviews and apps. It is one of the limitations of our work. We identified tens of coercive apps from a very small dataset of reviews (3000 reviews randomly selected from 82K reviews of the search result) due to time constraints. More thorough investigation of this phenomenon is needed to truly identify the deceptive behavior of these apps.

3.7 Summary

At first, we present Marco, a system for detecting deceptive Yelp venues and reviews, leveraging a suite of social, temporal and spatial signals gleaned from Yelp reviews and venues. We also contribute a large dataset of over 7K venues, 270K reviews from 195K users, containing also a few hundred *ground-truth* and *gold-standard* reviews (fraudulent/genuine) and venues (deceptive/legitimate). Marco is effective in classifying both reviews and venues, with accuracies exceeding 94%, significantly outperforming state-of-the-art strategies. Using Marco, we show that two weeks after an event, twice as many venues that host Yelp events experience a significant rating boost, when compared to the venues that experience a negative impact. Marco is also fast; it classifies a venue with 500 reviews in under 2.3s. We also present FairPlay, a system for detecting fraudulent and malware apps in Google’s app market. On data we collected from more than 87K Google Play apps that we monitored over more than 6 months, as well as from more than 600 gold standard datasets of fraudulent, malware and legitimate apps, FairPlay achieves an accuracy exceeding 98% in identifying malware and fraudulent apps. In addition,

we identified tens of apps in our monitored set, that coerce users into participating in search rank fraud.

CHAPTER 4

VISUAL VERIFICATION THROUGH LIVENESS ANALYSIS

In this chapter, we will focus on the problem of whether the visual stream uploaded by a user has been captured live on a mobile device, and has not been tampered with by an adversary. This problem is a cornerstone in a variety of practical applications that use the mobile device camera as a trusted witness. Examples applications include citizen journalism, where people record witnessed events (e.g., public protests, natural or man-made disasters) and share their records with the community at large. Other applications include video based proofs of physical possession of products and prototypes (e.g., for sites like Kickstarter [Kic], Amazon [Ama] and eBay [eBa]), and of deposited checks [BoA14, Fow10].

Part of the content in this section has been published during my Ph.D study, including the problem formulation, attack models and the proposed system solutions. The outline of this chapter is as follows: The motivation and challenges of this topic will be presented in Section 4.1. In Section 4.2, the system and adversary models have been described. After that, the detailed solution of this problem will be introduced in Section 4.3 and the implementation of the system has been described in Section 4.4. Data collection steps and the datasets have been explained in Section 4.5. Then the detailed evaluation of the solution through various experiments has been presented in 4.6. Finally, the limitations of the solution and the conclusion will be given in Section 4.7 and Section 4.8, respectively.

4.1 Motivation and Challenges

In response to the ubiquitous and connected nature of mobile and wearable devices, industries such as utilities, insurance, banking, retail, and broadcast news

have started to trust visual information gleaned from or created using mobile devices. Mobile apps utilize mobile and wearable device cameras for purposes varying from authentication to location verification, tracking, witnessing, and remote assistance. The citizen journalism revolution, enabled by advances in mobile and social technologies, transforms information consumers into collectors and disseminators of news. Major news outlets have started to fill out professional journalistic gaps with videos shot on mobile devices. The increasing popularity of citizen journalism is starting however to raise important questions concerning the credibility of impactful videos (see e.g., [Cit, Wit, She12, Gua14]). Videos from other sources can be copied, projected and recaptured, cut and stitched before being uploaded as genuine on social media sites.

We address the fundamental question of whether the visual stream uploaded by a user has been captured live on a mobile device, and has not been tampered with by a malicious user attempting to game the system. We refer to this problem as video “liveness” verification. This problem has several dimensions, that include assessing the location and time of capture, or the content of the video. For instance, CitizenEvidenceLab [Cit] provides tutorials to train the public to assess citizen videos from YouTube (see Figure 4.1 for a snapshot).

InformaCam [Inf] leverages the unique noise of the device camera to sign content it produces, along with the output of other sensors (e.g., GPS). This enables InformaCam to authenticate that content has been produced with a certain camera. InformaCam assumes that all sensor data is valid and has not been fabricated. It is also vulnerable to plagiarism attacks where the attacker points the camera to a projected video.

In this paper we focus on the liveness dimension of video verifications: verify that the video was captured on a mobile device, and has not been fabricated using

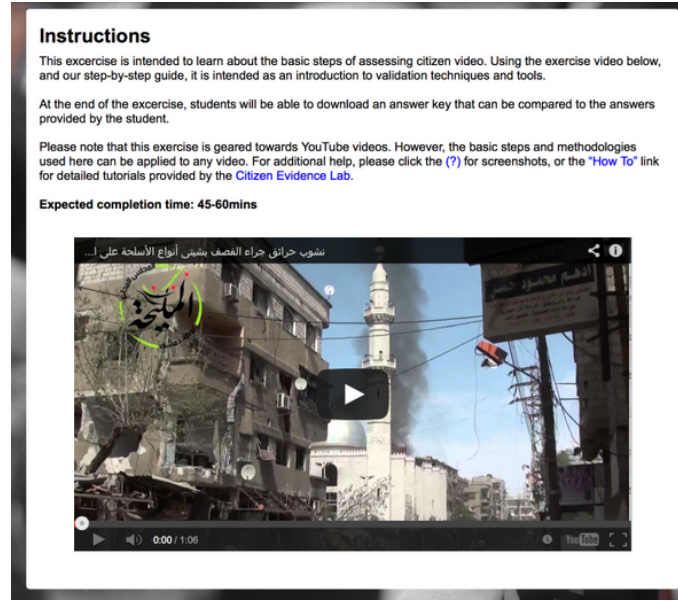


Figure 4.1: Snapshot of Citizen Evidence Lab [Cit] training session exercise. It consists of steps to verify the source of the video (i.e., account of uploader, upload time) and its content (e.g., clothes, accents, flags, landmarks).

material from other sources. Movee is vulnerable to the potent attacks that we study in this paper. For example, an attacker starts Movee and points to a portion of a target video playing on a projection screen, performs a pan motion as specified by Movee, then points the camera to the whole frame of the fraudulent video. Since Movee only uses the initial 6s chunk, the resulting sample passes Movee’s verifications. Furthermore, in Section 4.6.7 we quantitatively show the ineffectiveness of Movee for free-form movements even in 6s chunks: on the attacks we introduce, Movee’s false positive rate is as low as 38% and its false negative rate is 28%.

We introduce Vamos to address these limitations and provide the first video liveness verification system that works on unconstrained, free-form videos, does not impose a “verification” step on users, and is resilient to a suite of powerful, sensor based attacks.

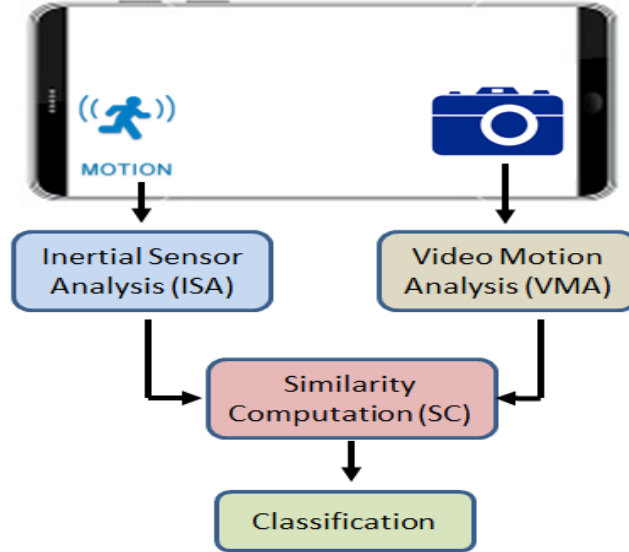


Figure 4.2: Movee uses four modules to verify a video stream: the i) Video Motion Analysis (VMA), and the ii) Inertial Sensor Motion Analysis (IMA), produce movement estimations during capture, iii) Similarity Computation extracts features, which iv) Classification uses to make the final decision.

4.2 The Model: System and Adversary

We now describe the system and adversary models that we assume in this work. We also propose a general classification of videos captured on mobile devices.

4.2.1 System Model

We consider a system that consists of a service provider, e.g. video sharing services such as Vine [Vin], YouTube [Youa] or check deposit services [BoA14, Fow10]). The provider offers an interface for subscribers to upload or stream videos they shot on their mobile devices.

We assume subscribers own mobile devices equipped with a camera and inertial sensors (i.e., accelerometers). Devices have Internet connectivity, which, for the purpose of this work may be intermittent. Subscribers need to install an application on their mobile devices, which we henceforth denote as the “client”.

A subscriber needs to use this client to capture videos. In addition to video, the client simultaneously captures the inertial sensor (accelerometer) stream from the device. The client uploads both the video and the accelerometer streams to the provider. The provider verifies the authenticity of the video by checking the consistency of the two streams. The verification is performed using limited information: the two streams are from independent sources, but have been captured at the same time on the same device.

We assume a system where the problems of establishing trust in the mobile device, operating system and associated drivers, and the mobile client are already addressed. This includes for instance a system where a chain of trust has been established [NKZS10, BDH⁺11, ULGW12]. The chain of trust ensures that the operating system, including the camera and sensor device drivers, and the installed apps, are trusted and have not been tampered with by an attacker, see e.g., [App, Arx]. A discussion of limitations is included in Section 4.7.

In the remainder of the paper we use the terms accelerometer and inertial sensor interchangeably.

4.2.2 Adversary Model

We assume that the service provider is honest. Users however can be malicious. An adversarial user can tamper with or copy video streams and inertial sensor data. The goal is to fraudulently claim ownership of videos they upload to the provider. Let V be such a video. The adversary can use a trusted device to launch the following attacks, that produce fraudulent videos or fraudulent video and acceleration data:

Copy-Paste attack. Copy V and output it.

Projection attack. Point the camera of the device over a projection of the target video. Output the result.

Random movement attack. Move the device in a random direction, and capture the resulting acceleration data. Output the video V and the captured acceleration stream.

Direction sync attack. Use the video to infer the dominant motion direction of V . Use the device to capture an acceleration sample that encodes the same motion direction. Output V and the acceleration sample.

Replay attack. Study the target video V . Then, holding a mobile device that captures acceleration data, emulate the movements observed in V . Let A' be the acceleration data captured by the device during this process. Output (V, A') .

Sandwich attack. The attacker studies the video V and emulates the observed movement. For instance, \mathcal{A} stacks two devices. The attacker plays the target video V on the top device. He then moves the device stack to emulate the movement seen on the top device. The device on the bottom records the resulting acceleration data, \overline{Acc} . \mathcal{A} outputs \overline{Acc} .

In the following, we describe the cluster attack, an automatic technique to produce fraudulent data: pair the target video with the acceleration stream copied from a “similar” but genuine sample.

Cluster attack. \mathcal{A} captures a dataset of genuine (video, acceleration) samples and stores them in $\Gamma_{\mathcal{A}}$. \mathcal{A} uses a clustering algorithm (e.g., K-means [Bis95]) to cluster the videos based on their movement. \mathcal{A} classifies the target V according to its movement and assigns it to one of the previously generated clusters: the cluster containing videos whose movement is closest to V . \mathcal{A} randomly picks one of the genuine (video, acceleration) samples in the cluster. Let (V', Acc') be the chosen sample. \mathcal{A} outputs Acc' .

Category ID	Distance to subject	User Motion	Camera Motion
1	Close	Standing	Stationary
2	Far	Standing	Stationary
3	Close	Walking	Stationary
4	Far	Walking	Stationary
5	Close	Standing	Scanning
6	Far	Standing	Scanning
7	Close	Walking	Scanning
8	Far	Walking	Scanning
9	Close	Standing	Following
10	Far	Standing	Following
11	Close	Walking	Following
12	Far	Walking	Following

Table 4.1: Video motion categories, based on (i) camera distance to the subject, (ii) the user motion and (iii) camera motion.

Next we introduce the stitch attack, that concatenates a plagiarized (video, acceleration) chunk with several genuine chunks. In Section 4.5.4 we construct stitched samples from multiple fraudulent and genuine chunks.

Stitch attack. \mathcal{A} takes as input parameters the target video V and two integers, $g > 0$ and $0 \leq k \leq g$. \mathcal{A} first creates a set of genuine video and acceleration chunks, $\Gamma_{\mathcal{A}} = \{(V_1, Acc_1), \dots, (V_1, Acc_g)\}$, e.g., by capturing them on the mobile device. \mathcal{A} uses either the cluster or the sandwich attack to fabricate \overline{Acc} , an acceleration stream for V . \mathcal{A} then “stitches” the fake chunk (V, \overline{Acc}) with the genuine chunks $\Gamma_{\mathcal{A}}$, according to the index k . Let \parallel denote the concatenation operation, applicable both to video and acceleration streams. Then, \mathcal{A} outputs (V_a, Acc_a) , where $V_a = V_1 \parallel \dots \parallel V_{k-1} \parallel V \parallel V_{k+1} \parallel \dots \parallel V_g$ and $Acc_a = Acc_1 \parallel \dots \parallel Acc_{k-1} \parallel \overline{Acc} \parallel Acc_{k+1} \parallel \dots \parallel Acc_g$.

4.2.3 A Classification of Mobile Videos

We posit that the success rate of the attacks previously introduced depends on the type of motions encoded in the video. For instance, it seems intuitive that videos where the hand-held device is stationary are easier to plagiarize. To verify our conjecture, we propose a general classification of videos captured on mobile devices, based on the following dimensions:

- **User motion:** We consider two types of recorder motions, “standing” and “walking”, but no motions such as jumping or driving.
- **Camera motion:** We consider three types of camera motions: “stationary”, “scanning” and “following”. “Scanning” means the camera moves in a direction (e.g., left to right) at a pace independent of the subject of the video. “Following” means the camera moves to maintain the subject within the confines of the video. We have not considered videos shot with head mounted cameras.
- **Distance to subject:** We consider video subjects that are either “close” or “far” to the camera. If the camera focuses on the subject of the video and only a limited area of the background is observed in the video, we say the subject is “close”. Otherwise, the subject is “far”.

Table 4.1 shows the resulting 12 mobile video categories. Figure 4.19 shows the category distribution of YouTube and free-form video sets we collected (§ 4.5.3 and § 4.6.6).

4.3 Movee: Solution Overview

We introduce Movee, a system to verify the live capture of videos uploaded from mobile devices. Movee performs an analysis based on the consistency between the

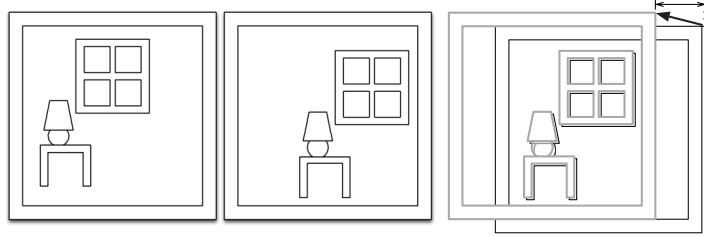


Figure 4.3: The Video Motion Analysis module processes each consecutive video frame and finds the motion vector by computing the amount of displacement that common image components have shifted between two frames.

motion inferred from the simultaneously and independently captured camera and inertial sensor streams. If the data from the inertial sensor corroborates the data from the camera, Movee concludes that the video was genuine: it has been taken by the user pointing the camera to a real scene.

The Movee client is intended to be installed in mobile devices as part of special purpose video capture apps. When the user wants to capture a video or a photo, the client performs two actions simultaneously: First, it turns the camera on and starts to capture video frames; second, it starts to collect a stream of accelerometer readings.

The data from the camera sensor is stored in periodically captured image frames. The data from the inertial sensor in most mobile devices comes in the form of periodically captured acceleration magnitudes on 3 main axes as measured by the accelerometer. Movee infers the direction and the magnitude of motion from these two different types of sensor data.

Figure 4.2 shows a diagram of Movee. The *Video Motion Analysis* (VMA) module uses an efficient image processing method to infer a motion vector over the timeline of the video from frame-by-frame progress. The *Inertial sensor Motion Analysis* (IMA) module, converts the raw inertial sensor readings into a motion vector over the same timeline. Subsequently, the *Similarity Computation* (SC) module

extracts features which represent agreements and differences between the two motion data, from the VMA and IMA modules. The final decision of whether the captured video is genuine is made in the *Classification* module. The Classification module uses trained classifiers on the data produced by the SC module to find out whether the inertial sensor data corroborates the video sensor data.

In the rest of this section, we detail each of these modules.

4.3.1 Video Motion Analysis (VMA)

The Video Motion Analysis (VMA) module takes as input the captured video stream and outputs an estimate for the direction and magnitude of the camera movement. The output of VMA is then used by the Similarity Computation module of Movee (see Figure 4.2).

VMA first retrieves the frame per second (fps) rate of the stream and each available frame. In a pre-processing step, it applies a Hamming window [Smi11] filter to eliminate noise from each frame.

For each pair of consecutive frames, VMA needs to find the movement of the camera. It is possible to perform this operation manually: First, print the photos on transparency films. Second, *shift* one sheet placed on top of the other and keep *comparing* the two prints until they line up with minimal difference. The amount that the edges of one sheet overhang the other represents the offset between the photos (see Figure 4.3). The common optical mice [opt] use this principle to determine pointer movement from a stream of images taken with a low resolution optical sensor mounted to their bottom side. The movement inferred from this analysis will be limited to two axes, i) horizontal along the X axis, and ii) vertical along the Y axis.

Algorithm 1 Pseudocode of the Video Motion Analysis (VMA) module. The `videoShift` operation computes and returns the total displacement on one axis, as computed from video frames.

```
1. Object implementation VMA;
2. Operation double videoShift(Video V)
3.   int N := V.getFrameCount();
4.   double totalShift := 0;
5.   h := createHammingWindow();
6.   for i := 1 to N - 1 do
7.     Frame f1 := V.getFrame(i);
8.     Frame f2 := V.getFrame(i + 1);
9.     totalShift += phaseCorrelate(f1, f2, h); od
10.  return totalShift;
11. end
```

Phase Correlation. VMA uses the *shift* and *compare* principle as well, by applying it on all consecutive frames of the video (see Algorithm 1). The result is a frame-by-frame displacement vector. However, it would have been prohibitively expensive to compute the differences between two frames for all possible pixel shifts, especially considering how large each frame is.

Instead, we use *Phase Correlation* [DCM87] to find the shift that minimizes the difference, by carrying the computation into the frequency domain. Phase correlation is an image processing technique that computes the spatial shift between two similar images (or sub-images). It is based on the Fourier shift property: a shift in the spatial domain of two images results in a linear phase difference in the frequency domain of the Fourier Transform (FT) [FF09]. It performs an element-wise multiplication of the transform images. It then computes the inverse Fourier transform (IFT) of the result, and finds the shift that corresponds to the maximum amplitude. This yields the resultant displacement. The maximum amplitude can be defined in

Direction	Video Shift	Sensor Shift
Up	Y++	X++
Down	Y- -	X- -
Left	X++	Y++
Right	X- -	Y- -

Table 4.2: Camera motion inference based on cumulative shifts along X and Y axes inferred from the video and inertial sensor streams. The device is considered to be in landscape orientation. X++ (and X- -) denote positive (and negative) X axis shifts that dominate shifts along other axes.

the two-dimensional surface with delta functions (colloquially referred to as *peaks*) at the positions corresponding to spatial shifts between the two images.

Then, for each pair of consecutive frames, VMA applies the phase correlation method to obtain linear shifts between images in both X and Y directions (see Algorithm 1). It then computes the *cumulative shift* along the X and Y axes by adding up the linear shifts for all consecutive frames retrieved from that video. Let $VS_{x,i}$ and $VS_{y,i}$ denote the cumulative video shifts of the i -th frame on the X and Y axes. We use $VS_{x,i}$ and $VS_{y,i}$ as feature descriptors (see Section 4.3.4).

Extract motion direction from video. Based on the computed cumulative shift along the X and Y axes, VMA infers the camera direction of movement. For instance, if the camera is in landscape orientation, and the cumulative shift over the X axis is negative, $VS_{x,i} < 0$, and dominates the one over the Y axis, (i.e., $|VS_{x,i}| \gg |VS_{y,i}|$), the video motion direction is to the right. Table 4.2 (first and second columns) summarizes the direction inference process. We use the notation X- - and X++ to denote negative and positive cumulative shifts along the X axis that dominate shifts along other axes.

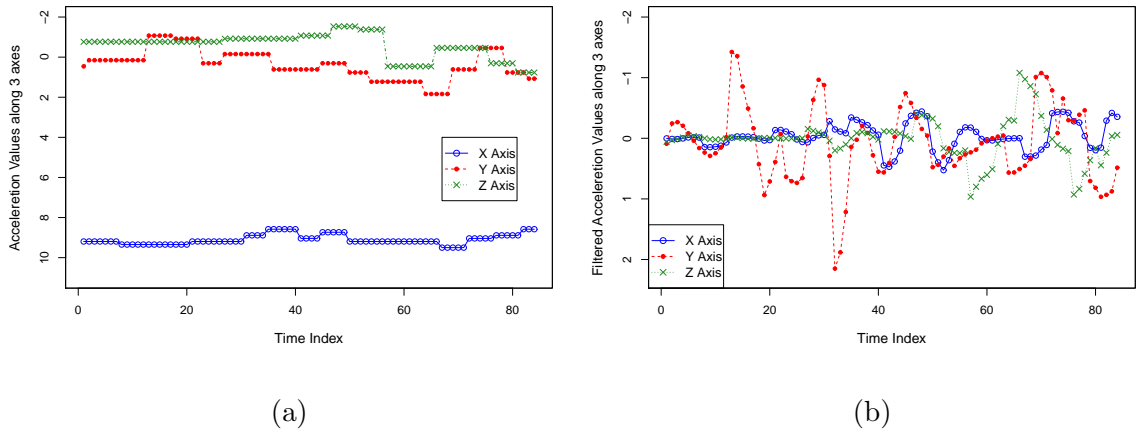


Figure 4.4: (a) Raw Accelerometer Data. (b) Filtered Accelerometer Data. The Y axis is the dominant axis for the direction and orientation of the device.

Algorithm 2 Pseudocode of the Inertial sensor Motion Analysis (IMA) module. The `sensorShift` operation computes and returns the total displacement on one axis, as computed from instantaneous accelerometer readings.

```

1. Object implementation IMA;
2. double Th;           #Threshold
3. Direction dr;       #movement direction
4. Operation double sensorShift(sensorData S)
5.     double totalShift := 0;
6.     int N := S.getSensorLogCount();
7.     for i := 1 to N do
8.         if (i = 1) then
9.             totalShift+ = dIntegral(S[i], 0);
10.        else
11.            totalShift+ = dIntegral(S[i], S[i - 1]);
13.    return totalShift;
14. end

```

4.3.2 Inertial Sensor Motion Analysis (IMA)

The Inertial Sensor Motion Analysis (IMA) module (see Figure 4.2) relies on the accelerometer sensor widely available in mobile devices. The IMA processes the data from the accelerometer in order to produce a motion direction and magnitude which is then compared in the Similarity Computation module with the output from the VMA module.

The inertial sensor coordinate system is defined relative to the screen of the phone in portrait orientation. The X axis is horizontal and points to the right, the Y axis is vertical and points up and the Z axis points towards the outside of the front face of the screen (coordinates behind the screen have negative Z values). Let $\{(A_{x,i}, A_{y,i}, A_{z,i}) | i = 1..m\}$ denote the accelerometer trace, recorded every T seconds. $(A_{x,i}, A_{y,i}, A_{z,i})$ is the i -th sample, containing accelerometer readings on the three axes. Instead of faster accelerometer sampling modes (e.g., “fastest” or “game” modes) we chose the slower but less noisy 16Hz mode.

Filtering step. In a pre-processing step, at each sampling time T , IMA removes duplicate (noise) acceleration values that occur (i) at time T , phenomenon that frequently occurs at the beginning of the capture interval, and (ii) within interval $[T - 10ms, T + 10ms]$. Furthermore, IMA uses a combination of low-pass and high-pass filters to remove the effects of gravity from the recorded acceleration stream. In a first, low-pass filter, let $G_{a,i}$ be the filtered gravity value on the a axis ($a \in \{X, Y, Z\}$) in the i -th sample and let $G_{a,i+1}$ be the gravity value to be filtered in the current, $(i + 1)$ -th sample. $A_{a,i+1}$ is the acceleration reading on the a axis for the $i + 1$ -th sample. Then, $G_{a,i+1} = \alpha G_{a,i} + (1 - \alpha)A_{a,i+1}, \forall a \in \{x, y, z\}$. We have experimented with values of α ranging between 0.6 and 0.95. We have found the value $\alpha = 0.8$ to perform best.

Subsequently, IMA passes the result through a high-pass filter, $FA_{a,i+1} = A_{a,i+1} - G_{a,i+1}$, where $FA_{a,i+1}$ denotes the filtered acceleration value on the a axis, $\forall a \in \{x, y, z\}$, for the $(i + 1)$ -th sample.

Figure 4.4(b) shows the effects of filtering for the sample raw acceleration of Figure 4.4(a)), where the phone was held in landscape orientation. Figure 4.4(a) shows that the gravity primarily influences the X axis, with acceleration values being changed around 9.19 (g value for the phone when in fixed position). However the gravity value also affects the acceleration readings on other axes. The filtering method eliminates the gravity affect.

Extract motion direction from acceleration data. Figure 4.4(b) shows that the Y axis movement is dominant (the highest translation/shift variations), thus the direction of movement is to the right in landscape orientation of the phone. Table 4.2 (first and third columns) summarizes the direction inference process also for acceleration data. For instance, given the cumulative shifts, AS , on all acceleration axes, if the device is in landscape orientation, $AS_{y,i} < 0$ and $|AS_{y,i}| > thr \times AS_{x,i}$ (for a given threshold thr), the accelerometer motion direction is to the right. We denote this situation through the notation Y-. We chose the thr value experimentally to be larger than 1.5, after analyzing sample data for different directions .

Extract motion distance from acceleration data. The *dIntegral* function used in Algorithm 2 uses the acceleration data to infer the displacement, as follows. Given acceleration data on each axis, $A_{a,1}, \dots, A_{a,m}$, where $a \in \{X, Y, Z\}$, captured every T seconds, IMA computes the position (relative to the starting point) using a double integral. We adopt the trapezoidal rule [Hil87] to approximate the definite integral $\int_c^d f(x)dx$, representing the area below the curve. The integration step is first applied to obtain velocity ($vel_{a,i} = vel_{a,i-1} + \frac{A_{a,i} + A_{a,i-1}}{2} * T$). In a second

application, the integration retrieves the position ($pos_{a,i} = pos_{a,i-1} + \frac{vel_{a,i} + vel_{a,i-1}}{2} * T$). $vel_{a,i}$ and $pos_{a,i}$, $i = 1..m$, denote the velocity and position at the i -th sample on the axis a . The resulting position shifts are combined to obtain the cumulative shift, $AS_{x,i}$, $AS_{y,i}$, $AS_{z,i}$, along each axis. $AS_{x,i}$, $AS_{y,i}$, $AS_{z,i}$ are then used as feature descriptors (see Section 4.3.4).

4.3.3 Similarity Computation (SC)

The Similarity Computation (SC) module compares the two motion sequences computed by the VMA and the IMA modules. It returns a set of features that summarize the nature of the similarity between the two sequences. The features are then used by the Classification module (see Section 4.3.4) to decide whether the two motion sequences corroborate each other, thereby concluding whether the video is genuine or not. The video motion and inertial sensor streams encode the same user hand movement, which are processed by the VMA and IMA modules respectively (see Figure 4.2) to each yield a motion stream.

To compute their similarity, we use a well-known sequence similarity measurement method from speech and pattern recognition, called Dynamic Time Warping (DTW) [ANCT09, SC07]. Similar to the well-known string edit distance, DTW is a dynamic programming solution to find the minimum cost set of operations that converts one sequence to the other.

In this subsection, we describe how we adapted the DTW algorithm to the practical issues in comparing the two motion sequences from the VMA and IMA modules. The two sequences differ in their number of samples, and have different magnitudes due to the nature of their source sensors. The VMA sequence length is proportional to the number of video frames, whereas the IMA sequence length is proportional to the product of the sample rate of the inertial sensor and the length

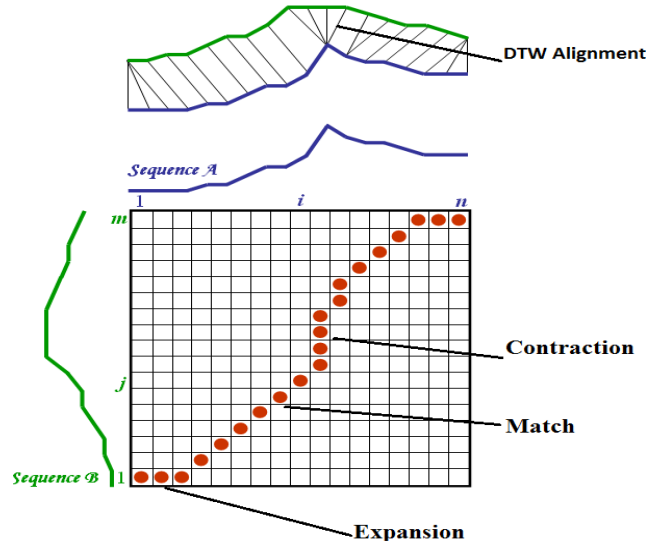
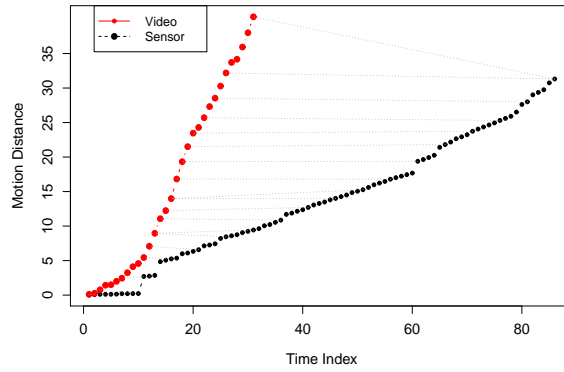


Figure 4.5: Illustration of DTW alignment for two time-dependent sequences. The red dots show the optimal warping path. A diagonal (match) move is a match between the two sequences. An expansion duplicates one point of one sequence and a contraction eliminates one of the points.

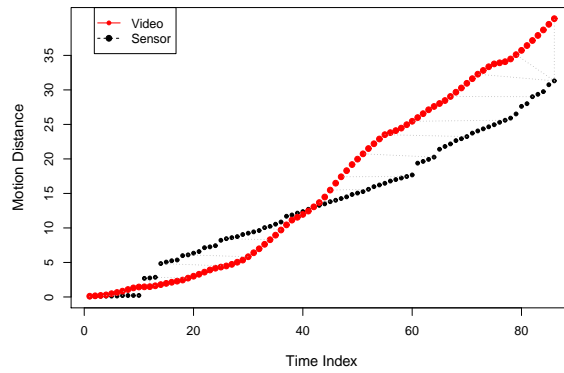
of the recording interval. Thus, we first perform a *stretching* step to make sure that the sequences output by VMA and IMA are of same length.

Furthermore, the motion sequence that the VMA infers from the video stream does not take into account the distance of objects into the camera. This may result in the same motion being registered as faster when the objects are close to the camera, and slower when the objects are far. To address this problem, in a second step we perform a *calibration* process: compute a coefficient to match the average speed of the motion the video stream to that of the inertial sensor stream.

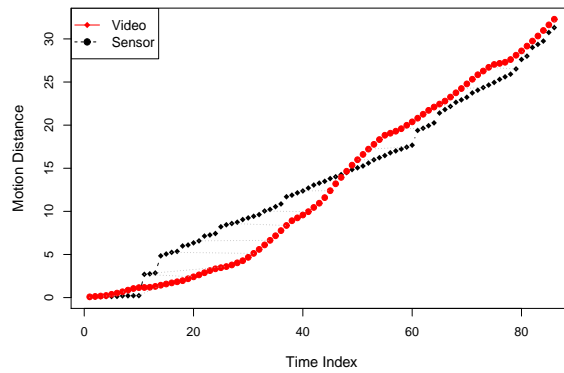
In the rest of this subsection, we first briefly detail the DTW algorithm, then present the stretching and calibration processes. We provide justification to the use of these methods with observed improvements in the resulting accuracy that the system gains after processing the features in the Classification module.



(a)



(b)



(c)

Figure 4.6: Example alignment of video and inertial motion streams extracted from the same experiment: (a) when using only DTW. (b) when stretching the shorter vector and applying DTW. (c) after stretching and calibration and applying DTW.

Dynamic Time Warping (DTW)

Let \mathcal{F} be a feature space. Let $X = (x_1, x_2, \dots, x_n)$ and $Y = (y_1, y_2, \dots, y_m)$, $n, m \in \mathbb{N}$, be time-dependent vectors, $x_i, y_j \in \mathcal{F}$, $i = 1..n, j = 1..m$. DTW computes the (n, m) -warping path of X and Y , that is a sequence $P(X, Y) = (p_1, \dots, p_L)$, where $p_l = (i, j) \in [1 : n] \times [1 : m]$, $\forall l \in [1 : L]$. The warping path satisfies boundary, monotonicity and step size conditions. At each step, DTW has the option to perform one of the following three *moves*, illustrated in Figure 4.5: a diagonal (or match) move, an expansion move, or a contraction move. The cost of a warping path $P(X, Y)$ is defined as the sum over the costs of all the moves in the path. The goal of DTW is to find a warping path of minimal cost among all possible warping paths.

Movee uses a variation of the DTW algorithm: the Variable Penalty Dynamic Time Warping (VPdtw) [CS12]. This is because the process of expanding and contracting the time axis of a sensor stream can produce a very high quality alignment to a video stream. However, excessive numbers of expansions and/or contractions can often result in matches at random parts of the streams and appear artificial rather than catching the genuine common movement patterns. VPdtw uses a penalty to constrain the use of expansions and/or contractions. This penalty is incurred whenever a non-diagonal (i.e., expansion or contraction) move is taken (see Figure 4.5).

Let L denote the length of the longer sequence between the video and inertial sensor sequences for each sample. We extract several characteristics of the computed DTW alignment as feature descriptors, to be used by the Classification module (see Section 4.3.4). First, the *normalized penalty cost*, defined as the penalty cost divided by L . Second, the *ratio of overlap points*, which is the number of overlap points between the two streams, divided by L . Third, the *ratio of diagonal moves*, which is the number of diagonal moves divided by L . Fourth, the *ratio of expansion moves*, which is the number of expansion moves divided by L . Finally, the *ratio of*

contractions moves, which is the number of contraction moves divided by L . The normalization to L ensures that the values are independent of the sample length.

Stretching

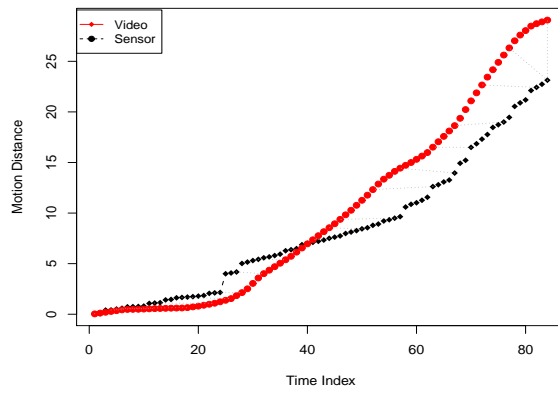
The sensor and video streams are sampled at different rates, thus the two vectors are of different length. The stretching step extends the shorter sequence (length s) to the length of the longer sequence (l). We use linear interpolation to compute $l - s$ new points for the shorter sequence. In Sections 4.6.3 and 4.6.4 we show that depending on the attack type, the use of stretching improves the accuracy of Movee in differentiating fraudulent from genuine videos by a rate of 5-12%. This result is illustrated in Figure 4.6(b), where the use of stretching significantly improves the ability of the DTW procedure to align the video and inertial sensor movement streams when compared to Figure 4.6(a).

Calibration

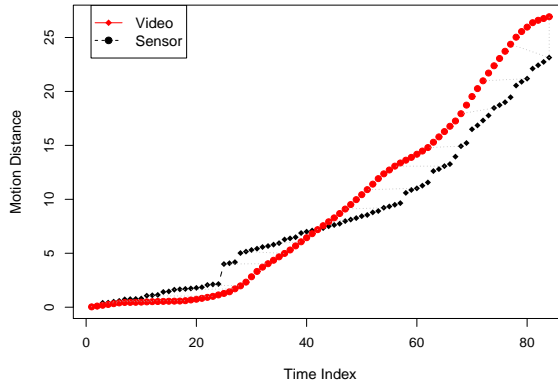
An artifact of the method used in the Video Motion Analysis module is that the same motion pattern can be registered as faster when the objects in the view are close to the camera, and slower when the objects are far. In order to compensate for this artifact, we calibrate the speed of the video motion vector with a coefficient to match that of the speed of the inertial sensor motion vector.

The goal is to compute a calibration factor CF , that is used to multiply all the points in the video stream. We have explored several calibration methods, including mean based and linear curve fitting. We provide details however only on the two methods that performed the best in our experiments, *truncated mean* and *polynomial curve fitting*.

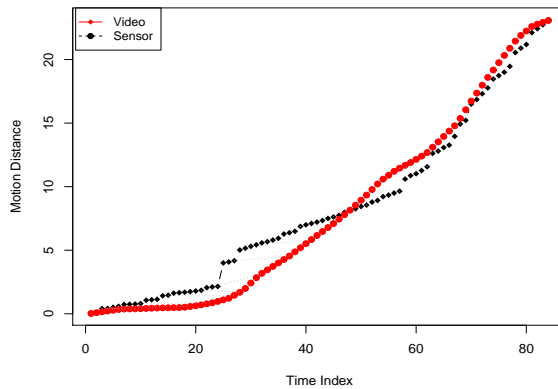
Truncated mean. The truncated mean computes the mean after discarding the high and low ends of the probability distribution (see Figure 4.7(b)). We apply this



(a)



(b)



(c)

Figure 4.7: Effect of calibration in the similarity computation. (a) No calibration. (b) Truncated mean calibration. (c) Polynomial curve fitting calibration.

concept as follows: For each pair of points in the sensor and video vectors, compute their ratio and add it to a *ratio vector*. Compute the truncated mean of the ratio vector, discarding 12.5% from both the low and the high ends of the distribution.

Polynomial curve fitting. Polynomial curve fitting [Coo93] constructs the polynomial that has the best fit to a series of data points (see Figure 4.7(c)). To compute the coefficients that best fit the curve to the given data, we use the least squares method [Coo93] to minimize the error between the data and the fitted polynomial [Coo93]. Let SP_s denote the average value over the points on the fitted curve for the sensor stream and let SP_v denote the average value of the points on the curve of the video stream. Compute the calibration factor as $CF = \frac{SP_s}{SP_v}$.

Figures 4.7(b) and 4.7(c) show sample calibration outputs for these two methods, when compared to the uncalibrated version shown in Figure 4.7(a).

Example Alignment

To illustrate the need for the DTW, stretch and calibration steps previously described, we provide here experimental results of their use on a genuine sample of video and inertial sensor streams, captured using Movee (see Section 4.4 for implementation details). Figure 4.6(a) shows the alignment between the video and inertial sensor streams when only DTW is used. Figure 4.6(b) shows the resulting alignment when DTW and stretching are applied. Finally, Figure 4.6(c) shows the alignment achieved when DTW is applied along with stretching and calibration. The experiment shows that stretching is vital to achieve a good alignment, while calibration further improves the quality of the alignment.

4.3.4 Classification

The Similarity Computation module produces 14 features that represent the nature of the similarity between the motion information inferred from the video stream

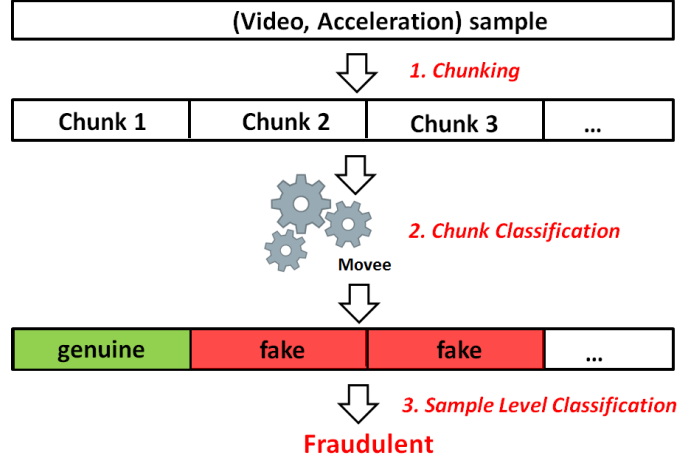


Figure 4.8: Illustration of the Vamos architecture and operation. Vamos consists of three steps, (i) “chunking”, to divide the (video, acceleration) sample, (ii) chunk level classification, and (iii) sample level classification.

and the one observed from the inertial sensor data. The features are: (1) the movement direction of the target from the center of the screen (see Section 4.4), (2-5) the cumulative shift of the video and accelerometer on the x and y axes (4 descriptors), (6) the video motion direction, (7) the sensor motion direction, (8) the DTW distance after stretching and calibration steps, (9) the calibration factor, CF , (10) the normalized penalty cost, (11) the ratio of overlap points, (12) the ratio of diagonal moves, (13) the ratio of expansion moves and (14) the ratio of contractions moves.

The Classification module runs trained classifiers over these features to determine whether there is sufficient evidence that the video stream is genuine. Section 4.6.1 describes the classifiers used in our experiments.

4.3.5 Vamos: Video Accreditation Through Motion Signatures

In this section we introduce Vamos (Video Accreditation Through Motion Signatures) an un-constrained video liveness analysis system. The verifications of Vamos

leverage the entire video and acceleration sample. This is in contrast with Movee, that relies only on the initial section of the sample. Vamos consists of the three step process illustrated in Figure 4.8. First, it divides the input sample into equal length chunks. Second, it classifies each chunk as either genuine or fraudulent. Third, it combines the results of the second step with a suite of novel features to produce a final decision for the original sample. In the following, we detail each of these steps.

Chunk Extraction

The “chunking” process divides a video and acceleration sample $S = (V, Acc)$ into fixed length chunks. We consider a 1s granularity of division. While 6s is the chunk length we use in the experiments, we consider here a parameter l to denote the length in seconds of the chunks. We call a *transition point* (TP) to be the time when the sample transitions from one video motion category to another (e.g., from category 4 to category 8). Let a *transition chunk*, denote a l second chunk that contains parts that belong to multiple video categories. Let $V[s, t]$ and $Acc[s, t]$ denote a segment of V and Acc that starts at second s and ends at second t . The chunking process produces a set C of chunks, initially empty. Let L denote the length of the (V, Acc) sample. We propose three chunking techniques, illustrated in Figure 4.9:

Sequential chunking. Divide (V, Acc) into sequential chunks, starting with the beginning. Let $n = |C| = \lfloor L/l \rfloor$. Then, $C = \{(V[0, l - 1], Acc[0, l - 1]), (V[l, 2l - 1], Acc[l, 2l - 1]) \dots (V[l(c - 1), lc], Acc[l(c - 1), lc])\}$.

Segment based chunking. Identify the transition points of the sample (V, Acc) . Let a sample *segment* denote the part of a sample between either (i) the beginning of the sample and the first transition point, (ii) two transition points, or (iii) the last transition point and the end of the sample. Discard all segments of (V, Acc)

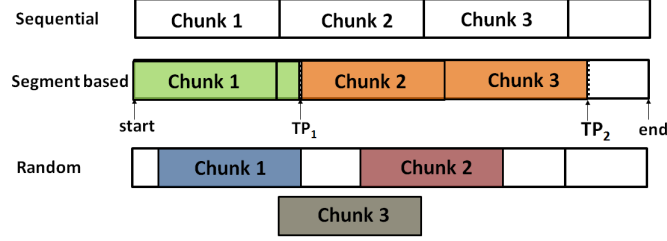


Figure 4.9: Chunk extraction illustration. For segment based chunking, the first segment produces a single usable chunk. For random chunking, chunk 3 overlaps both chunks 1 and 2.

whose length is less than l . Divide remaining segments according to the sequential chunking described above.

Randomized chunking. Produces k chunks, $0 < k \leq L$, where k is an input argument, as follows. Generate k different index values within the sample, $0 \leq i_1, \dots, i_k \leq L$ such that for any s and t , $1 \leq s, t \leq k$, $i_s + l \neq i_t$. For each i_j , $j = 1..k$, if $i_j \leq L - l$, then $C = C \cup (V[i_j, i_j + l - 1], Acc[i_j, i_j + l - 1])$. Otherwise, $C = C \cup (V[i_j - l, i_j], Acc[i_j - l, i_j])$.

Sequential chunking may produce transition chunks, that contain one or more transition points. Segment based chunking will not produce transition chunks. However, segment based chunking requires a mechanism to identify transition points. Randomized chunking can produce transition chunks and also overlapping chunks. Sequential and segment based chunking produce strictly non-overlapping chunks.

CL-Vamos: Chunk Level Verification

In the second step, Vamos classifies each chunk produced by the first step, as either genuine or fraudulent. While Movee [RTC13] works on fixed length chunks, it is limited to video and inertial sensor streams that encode one of 4 movements (up, down, to the left, or to the right). Specifically, 3 of the 14 features of Movee are (i) the placement of the bullseye relative to the center of the screen, (ii) the dominant video motion direction and (iii) the dominant sensor motion direction.

We introduce here CL-Vamos, the first liveness verification solution that works on free form chunks, that encode unrestricted movements. Similar to Movee, CL-Vamos analyzes the consistency of the inferred motion from the simultaneously and independently captured video and acceleration streams. First, it uses an efficient image processing method to infer a motion vector over the timeline of the video from frame-by-frame progress. Second, it converts the raw inertial sensor readings into a motion vector over the same timeline. Subsequently, CL-Vamos uses the Dynamic Time Warping algorithm (DTW) [M107] to find the set of operations that minimizes the cost of converting one vector to the other.

CL-Vamos is not restricted to the dominant direction of movement and removes the features extracted from it. Instead, we have investigated a wide range of features on both the x and y axes. Due to lack of space we report and evaluate here (see Section 4.6) the feature combination that achieved the best performance.

Specifically, CL-Vamos computes the DTW between the motion vectors extracted from the projections of the video and acceleration streams on both the x and y axes. For each axis, DTW returns the number of diagonal, expansion and contraction moves that convert one vector to the other, and the cost of the resulting transformation. CL-Vamos uses this information to generate the following features, for both the x and y axes:

- The DTW distance (transformation cost) between the video frame shift and acceleration streams.
- The ratio of overlap points: the number of overlapping points in the two motion vectors divided by the length of the vectors.
- The ratio of diagonal, expansion and contraction moves to the number of points in the vectors.

CL-Vamos uses these features, along with other Movee features (e.g., the cumulative shift of the video and accelerometer on the x and y axes), with supervised learning to train classifiers. For each chunk C_i in C , let $c_i \in \{genuine, fake\}$ denote the classification produced by CL-Vamos, and let $a_i \in \{genuine, fake\}$ denote the actual status of the chunk. We consider a “positive” to denote a fake chunk, and a “negative” to denote a genuine chunk.

We observe that the false positive rate of CL-Vamos, $FPR = Pr(c_i = fake|a_i = genuine)$. That is, the false positive rate denotes the probability that a chunk is classified as fake (positive), given that the chunk is in fact genuine. Similarly, the false negative rate is $FNR = Pr(c_i = genuine|a_i = fake)$, the true positive rate is $TPR = Pr(c_i = fake|a_i = fake)$ and the true negative rate is $TNR = Pr(c_i = genuine|a_i = genuine)$.

Vamos: Whole Video Classification

Let us assume that for a sample $S = (V, Acc)$, f chunks in C have been classified as fraudulent and g chunks have been classified as genuine. Let $n = f + g = |C|$. We say S is genuine iff $\forall i = 1..n, a_i = \text{“gen”}$. S is fake if $\exists i, i = 1..n$, s.t., $a_i = \text{“fake”}$. We can write the probability that the sample $S = (V, Acc)$ is fake, $Pr(S = fake)$, given the above classification result, as

$$\begin{aligned} Pr[S = fake | \bigwedge_{i=1}^g (c_i = gen), \bigwedge_{i=g+1}^n (c_i = fake)] &= \\ &= 1 - \prod_{i=1}^g Pr(a_i = gen | c_i = gen) \times \\ &\quad \prod_{i=g+1}^n Pr(a_i = gen | c_i = fake). \end{aligned}$$

Let $\alpha = Pr(a_i = gen | c_i = gen)$, for any of the chunks C_i in C . Similarly, let $\beta = Pr(a_i = gen | c_i = fake)$. Then, we have that $Pr(S = fake) = 1 - \alpha^g \times \beta^f$.

Now, based on Bayes' theorem, we have that

$$\alpha = \frac{TNR \times Pr(a_i = \text{genuine})}{TNR \times Pr(a_i = \text{genuine}) + FNR \times Pr(a_i = \text{fake})}.$$

Similarly, we have that $\beta = \frac{FPR \times Pr(a_i = \text{genuine})}{FPR \times Pr(a_i = \text{genuine}) + TPR \times Pr(a_i = \text{fake})}$. We can compute thus α and β as a function of $Pr(a_i = \text{genuine})$ and $Pr(a_i = \text{fake})$. We obtain these probability values statistically, based on the performance of CL-Vamos on a large number of chunks. Specifically, $Pr(a_i = \text{fake}) = \frac{\text{Nr. of fake chunks}}{\text{Total nr. of chunks}}$ and $Pr(a_i = \text{genuine}) = \frac{\text{Nr. of genuine chunks}}{\text{Total nr. of chunks}}$, see Section 4.6.9.

We introduce several mechanisms to classify samples as genuine or fraudulent. First, we propose a majority voting approach, where a sample $S = (V, Acc)$ is classified as fraudulent if more than a threshold of the chunks of S have been classified by CL-Vamos as fraudulent: $\frac{f}{f+g} > thr$. The threshold thr is a parameter that will be determined experimentally. Second, we consider a probabilistic approach that labels a sample as fake if $Pr(S = \text{fake}) = 1 - \alpha^g \times \beta^f$ is larger than a threshold value. We experiment with threshold values in Section 4.6.9. Third, we propose a classifier based approach, that uses the following novel features:

- **Results of CL-Vamos:** The number of fraudulent chunks, f and the number of genuine chunks g . The classification results $c_i, \forall i = 1..n$. The probability that the sample S is fake, $Pr(S = \text{fake})$.
- **Aggregate features:** For each of the 18 features of CL-Vamos, compute the minimum, maximum, average and standard deviation of the feature's values over $c_i, \forall i = 1..n$, as new features.

Vamos uses these features with supervised learning to train classifiers for samples of arbitrary length and encoding arbitrary motions.



Figure 4.10: Movee in action on smartphone: Target icon (bullseye) at the bottom of the screen shows the direction in which the user needs to move the camera.

4.4 Movee Implementation

We have implemented a Movee client using Android and a server component using C++, R and PHP. We used the OpenCV (Open Source Computer Vision) library [Ope] for the video motion analysis. The client allows users to capture movies and simultaneously provide proofs of liveness. Figure 4.10 shows a snapshot of Movee in action, on a smartphone. When starting the client, the user is presented with an initial screen that instruct her to hold the device firmly before pressing the start button. This is done to prevent initial accelerometer reading errors. Once the user presses the start button, a target appears (bullseye). The user is instructed to move the camera in the direction of the target. Once the user starts to move the camera toward the target, the target begins to move toward the center of the screen. The target moves at a speed that ensures that the process takes at least 6s.

We call this 6s long process, the *verification* interval. During the verification interval, the Movee client captures the video stream and logs the accelerometer data. After the verification interval, the user can continue capturing the intended

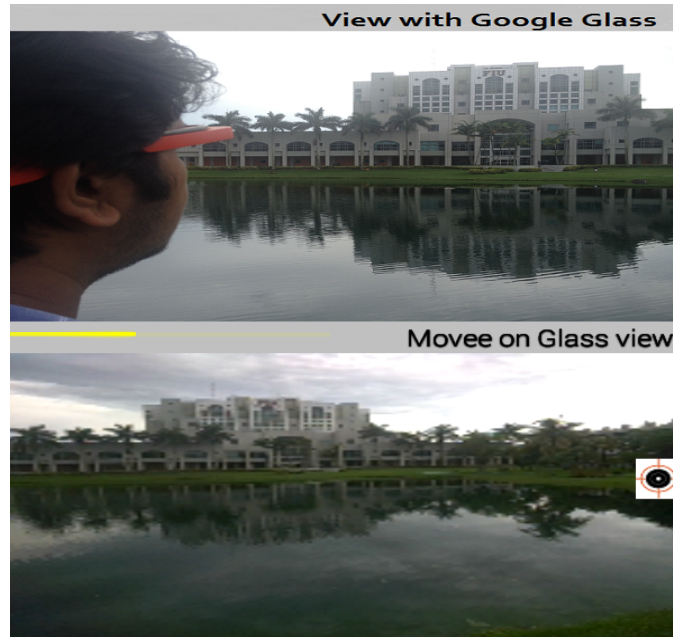


Figure 4.11: Movee on Google Glass: Top snapshot shows view with Google Glass. Bottom snapshot shows view from Movee on Glass perspective. Target icon (bullseye) at the right of the screen shows the direction in which the user needs to move the head mounted glass.

scenes. We were inspired by Vine¹ to choose the verification interval to be 6s. This choice presents the additional advantage that it keeps the size of the video file small (around 150 KB in the Samsung Admire Phone), and reduces the communication overhead.

We have also implemented MoveeG, a Movee app variant for the Google Glass. We have used the Glass Development Kit (GDK) to build MoveeG as a glassware that runs directly on Glass (around 700 lines of code). MoveeG starts and stops by voice command or through a tap based menu. Since the built-in camera activity has limited functionality, we have built our own logic with the Android Camera API [cam], to capture videos. Once the video capture is completed, MoveeG sends

¹Vine [Vin] is an application that allows users to create and post (on Twitter, Facebook) video clips.

the captured video and accelerometer streams to a server over the Glass WiFi connectivity, using HTTP POST requests.

4.5 Data Collection

We have used the implemented Movee and MoveeG applications to collect video and acceleration samples from real life users. We have worked with the Institutional Review Board (protocol number IRB-13-0582) at FIU to ensure an ethical interaction with the users during the collection process.

We used the 3D accelerometers, available in most recent smartphones, tablets and Google Glass, to acquire motion acceleration data. We have collected smartphone data using a Samsung Admire with a fps (frames per second) rate of 14, that samples accelerometer readings at 16.67Hz [Sen] mode. We have collected Google Glass data using a Glass device that samples accelerometer readings at 50Hz [Sen].

We have collected data from a total of 13 users, in multiple rounds. We have selected participants from FIU’s student body and campus visitors. 9 of the users are males and 4 are females. Their age ranges between 23 and 32 yo., and their occupations include biologist, fashion designer, housewife, software, civil and electrical engineers.

Ethical considerations. We have used the Vamos application to collect video and acceleration samples from real life users. We have worked with the Institutional Review Board (protocol number IRB-13-0582) at FIU to ensure an ethical interaction with the users and collection of the (video, acceleration) samples.

4.5.1 Smartphone Data Collection

Random and direction sync attack datasets. We have first collected “genuine” video and accelerometer data from a subset of 10 participants (7 male, 3 female). Each participant was asked to use Movee, following the instructions shown on the

screen (see Section 4.4). We have collected 10 well defined (6s long) samples from each user; the total of 100 samples are stored in a “genuine” dataset.

We have used the genuine dataset to generate random and direction sync attack datasets as follows. Each of these datasets contains an equal number of genuine and fraudulent video and acceleration samples. The “random attack” dataset consists of 50 video and corresponding acceleration samples from the genuine dataset, and 50 fraudulent samples created according to the Random attack. Specifically, each fraudulent “random” video and acceleration sample is created from one genuine sample, by coupling its video with the acceleration data of another, randomly chosen sample.

Similarly, the “direction sync attack” dataset contains the other 50 genuine samples from the genuine set, and 50 fraudulent samples created according to the Direction Sync attack. That is, each fraudulent sample couples the video of one genuine sample with the acceleration data of another genuine sample, with the same direction of movement.

Cluster attack dataset. We have generated the “cluster attack” dataset from data we collected from a subset of 12 participants (8 male, 4 female). The participants were given the freedom to move (themselves and the smartphone) in any direction for at least 15s but no more than 30s, while using Movee. We have collected a total of 141 samples of 15-30s long video and acceleration streams.

Similar to the random and direction sync attack datasets, we have built the cluster attack dataset to consist of an equal number of genuine and fraudulent video and corresponding acceleration samples. For this, we introduced the following variation of the Cluster attack introduced in Section 4.3. First, we divided each of the 141 samples into 6s long chunks. Second, we ran K-means clustering [Bis95] at the chunk level, to cluster the chunks according to the motion determined by

the VMA module (see the cluster attack in Section 4.3). This resulted in 423 genuine chunks of video and corresponding acceleration stream. We applied the v-fold cross-validation [Sta10] algorithm to automatically determine the number of motion clusters in the data. The v-fold cross-validation step produced $K=6$ for our cluster dataset. Then, for each of the 423 genuine chunks, we randomly chose another chunk from the same motion cluster. Third, we coupled the video from the first chunk with the acceleration stream of the randomly selected chunk. We added each such fraudulent sample to the “cluster attack” dataset. Thus, this dataset contains 423 genuine and 423 fraudulent (video, acceleration) chunks.

Replay attack dataset. We have also collected a “replay attack” dataset built according to the attack described in Section 4.3. Similar to the cluster attack dataset, the replay attack dataset also consists of 423 genuine and 423 fraudulent chunks.

4.5.2 Google Glass Data Collection

We performed a similar data collection process for a Google Glass device. First, we collected data from a subset of 5 users (3 male, 2 female): 20 well defined 6s long samples from each user. The total of 100 samples form a “genuine” dataset. Similar to Section 4.5.1, the “random attack” dataset for Glass, contains 50 genuine samples and 50 fraudulent samples created according to the Random attack. The “direction sync attack” dataset for Glass, contains the other 50 genuine samples and 50 fraudulent samples created according to the Direction Sync attack.

For the cluster and replay attack datasets we have collected 84 genuine samples (30s long each), resulting in a total of 420 chunks of 6s each. Then, each fraudulent cluster and replay chunk is created as specified in Section 4.3. Thus, each of the Glass cluster and replay attack datasets contains 420 genuine and 420 fraudulent chunks.

Category	Chunk count	Category	Chunk count
1	26	8	28
2	50	9	26
3&7	82	10	35
4	18	11	28
5	44	12	22
6	42		

Table 4.3: Number of chunks of the free-form dataset, per category. Details in Section 4.6.6.

We have collected datasets of citizen journalism videos from YouTube and of free-form (video, accelerometer) samples from real users. We have also created datasets of fraudulent samples following the attacks introduced in Section 4.3. In the following we detail each dataset.

4.5.3 YouTube Video Collection

We have collected 150 random citizen journalism videos from YouTube, in the following manner. First, we have identified relevant topics using Wikipedia’s “Current Events” site [Wik], BBC [BBC] and CNN [CNN]. They include political events (e.g., Ukraine, Venezuela, Middle East), natural disasters (e.g., earthquakes, tsunamis, meteorite landing), extreme sports and wild life encounters. We have used keywords from such events to identify videos in YouTube that have been captured by a regular person, using a mobile camera. We have discarded videos shot by a professional cameraman or using a head mounted camera. We collected the 150 videos from 139 users accounts. We have made public this list of videos [youub]. The total length of the 150 videos is 13,107 seconds. We analyze this dataset in Section 4.6.6.

Free-form data set. We have collected data from 16 users². Each user was asked to use Vamos, following the instructions shown on the screen: move the device

²11 are males and 5 females, aged 23-32, occupation including biology, fashion design, unemployed, and software, civil and electrical engineering

in any direction to capture videos. Each user contributed 10 free-form videos (and associated accelerometer data), producing a free-form dataset of 160 videos. We have manually annotated the free-form dataset video samples according to the categories described in Table 4.1.

We have divided each sample of the free-form dataset into 6s chunks, using segment based chunking (see Section 4.3.5), producing a total of 401 genuine chunks. Table 4.3 shows the distribution of the chunks into categories. We have made the free-form dataset publicly available [ffd].

4.5.4 Attack Datasets

Sandwich attack dataset. Two skilled users have performed the sandwich attack on the 160 free-form video dataset. We have used the following procedure, for each whole video (not at chunk level). The attacker watches the target video an unlimited number of times. The attacker stacks two phones. The attacker plays the target video on the top device. The bottom device records the acceleration readings during the session. The attacker can shoot any number of takes, until satisfied with the result.

We combine the original video with the resulting attack acceleration sample to produce a “sandwich sample”. We used the segment based chunking method to divide each sandwich sample into 6s (video,acceleration). Thus, each sandwich chunk corresponds to one of the free-form chunks. The sandwich chunk dataset contains thus also 401 chunks.

Cluster attack dataset. We ran K-means clustering [Bis95] on the free-form chunk dataset, to cluster the chunks according to their motion (see Cluster attack). We applied the v-fold cross-validation algorithm [Ar107] to determine the optimal number of clusters in our dataset. The outcome was $K = 6$. The cluster attack dataset

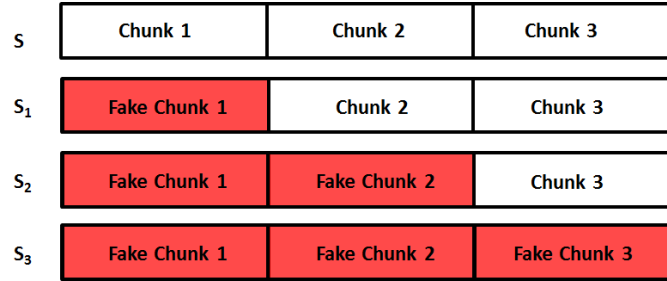


Figure 4.12: Stitch attack example. For a genuine sample of 3 chunks, the attacker produces 3 fake samples, with 1 to 3 fake (red) chunks. The genuine chunks are copied from the genuine sample.

consists of two subsets, of genuine and fraudulent (video, acceleration) chunks. We used the free-form chunk dataset as the genuine data. To create the fraudulent subset, for each genuine chunk, we randomly chose another chunk from the same (motion) cluster. We then coupled the video from the first chunk with the inertial sensor data of the randomly selected chunk. Thus, the genuine and fraudulent subsets of the cluster attack dataset each contain 401 chunks.

Stitch attack datasets. We have built two stitch attack datasets, one based on the fake cluster chunks and one on the fake sandwich chunks of the previous two attack datasets. The construction process is the following. First, we discarded 4 out of the 160 free-form samples, as they do not have a 6s chunk belonging to a single category. We then discarded 43 samples that have only one chunk. For each of the 113 remaining samples (that has at least 2 chunks), we construct 3 fraudulent samples. For instance, for a 2 chunk genuine sample, we create a fraudulent sample having the first chunk fake, the second genuine, one where the first chunk is genuine, but the second is fake, and one where both chunks are fake. For samples with more than 3 chunks, the position of the fake chunks in any of the 3 created fake samples is randomly selected. The fake chunks are from either the sandwich or the cluster chunk datasets.

Figure 4.12 illustrates the generation of fraudulent samples given a genuine free-form sample of 3 chunks. The reason for dropping samples with less than 2 chunks is that we need to create the same number of fake samples given any genuine sample (3 fakes per genuine sample). Samples with 1 chunk cannot produce 3 fake stitch samples, thus had to be discarded. The resulting stitch datasets based on the cluster and sandwich attacks have thus each 339 fake samples $((160 - 4 - 43) \times 3)$.

4.6 Evaluation

In this section we present experimental results for Movee and MoveeG. We first describe the experimental setup. Second, we evaluate the overhead of the liveness analysis on the server. Furthermore, we evaluate the performance of Movee and MoveeG on the attack datasets introduced in Sections 4.5.1 and 4.5.2. Finally, we evaluate the impact of MoveeG on the battery lifetime of a Google Glass device.

4.6.1 Experiment Setup

The Classification module (see Section 4.3.4) runs trained classifiers to determine whether there is sufficient evidence that a video stream is genuine. We have used several classifiers, including Multilayer Perceptron (MLP) [Gal90], Decision Tree (C4.5), Random Forest (RF) [Bre01], Bagging and Random Tree [AG97].

We have applied 10-fold cross-validation tests [Koh95b] to assess how the results of the statistical analysis will generalize to an independent data set. Specifically, the ground truth data set is randomly partitioned into k equal sized subsets. $k-1$ subsets are used for training the model and the last subset is used for testing the model. This process is repeated k times (the folds), with each of the k subsets used exactly once for validation. The k results from the folds are averaged to produce a single estimation. The advantage of this method is that all observations are used for both training and validation, and each observation is used for validation exactly

once. We have used the Weka version 3.7.9 data mining suite [Wek] to perform the experiments, with default settings: For the backpropagation algorithm of the MLP classifier, we set the learning rate to 0.3 and the momentum rate to 0.2.

Metrics. We briefly define the metrics we use to evaluate the accuracy of Movee. The TPR (True Positive Rate) metric denotes the fraction of videos correctly identified as genuine, the FPR (False Positive Rate) denotes the fraction of videos incorrectly identified as genuine and the FNR (False Negative Rate) denotes the fraction of videos incorrectly identified as fraudulent. The Receiver Operating Characteristic (ROC) curve [ROC] is a visual characterization of the trade-off between the False Positive Rate (FPR) and the False Negative Rate (FNR). The Equal Error Rate (EER) [TB04] is the rate at which both accept and reject errors are equal. A lower EER denotes a more accurate solution. The area under the ROC curve (AUC) is equal to the probability that a classifier will rank a randomly chosen genuine sample higher than a randomly chosen fraudulent one. An area of 1 represents a perfect test; an area of 0.5 represents a worthless test.

During the experiments, we have tested Movee on a Samsung Admire smartphone running Android OS Gingerbread 2.3 with an 800MHz CPU. We have tested MoveeG on a a Google Glass running Android OS KitKat 4.4.2 with OMAP 4430 dual-core ARM Cortex-A9 CPU and 682 MB of RAM. We have used a Dell laptop equipped with a 2.4GHz Intel Core i5 processor and 4GB of RAM for the server.

4.6.2 Server Overhead

Figure 4.13 shows the overhead (divided into modules) of the Movee liveness analysis on the server, running on the Dell laptop, for 6s videos. The values shown are an average over 10 experiment runs. It shows that the VMA is the most time consuming module, slightly exceeding 1s. The IMA and Classification components (running the

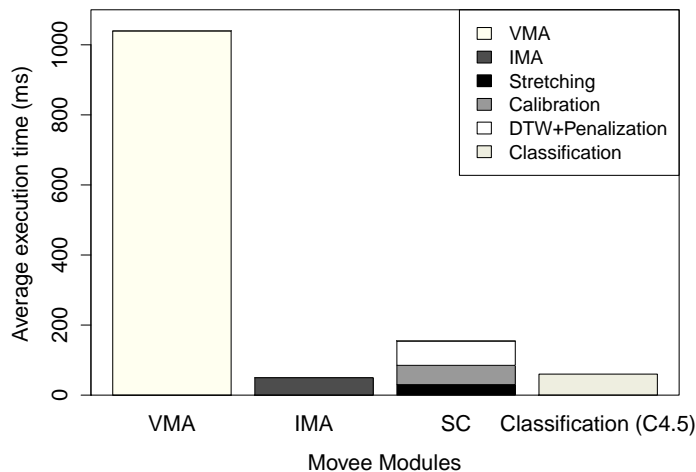


Figure 4.13: Movee (per-module) server side overhead: video processing is the most expensive. The total cost is however under 1.3s.

C4.5 classifier) impose the smallest overheads, of 110ms taken together. MLP takes an average of 940 ms and Random Forest an average of 140 ms. The overhead of the SC module is around 150ms, with the smallest cost imposed by the stretching step and the highest cost by the penalty based DTW.

4.6.3 Movee Attack Detection

Movee prevents the Copy-Paste attack of Section 4.3: no sensor stream exists. Movee also detects the Projection attack: the motion registered in the projected video is likely inconsistent with the acceleration data of the device capturing the video. We now evaluate the ability of Movee to detect random, direction sync, cluster and replay attacks, on a smartphone. The next subsection studies the performance of MoveeG on the same attacks, but executed on a Google Glass.

We focus first on the random and direction sync attacks on the smartphone, using the corresponding attack datasets described in Section 4.5.1. Details of the 3 best performing classifiers, including TPR, FPR and FNR values are shown in

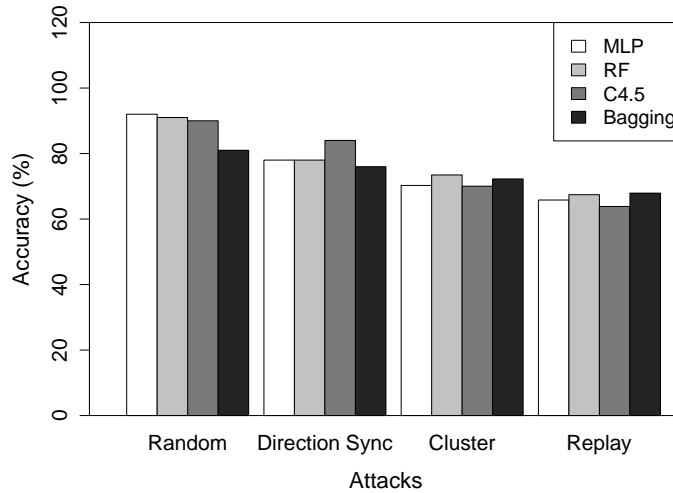


Figure 4.14: Summary of Movee accuracy on smartphone random, direction sync, cluster and replay attack datasets. The accuracy (y axis) labels exceed 100% to fit the legend.

Attack	Classifier	Acc(%)	TPR(%)	FPR(%)	FNR(%)
Random	MLP	92	93.33	9.09	6.67
	RF	91	88.89	7.84	11.11
	C4.5	90	91.11	10.9	8.89
	Bagging	81	82.0	20.0	18.0
Dir sync	MLP	78	85.71	31.8	14.29
	RF	78	82.14	27.27	17.86
	C4.5	84	82.14	13.63	17.86
	Bagging	76	82.14	31.81	17.86

Table 4.4: Detailed accuracy results of Movee on the smartphone random and direction sync attacks. For the random attack, Movee with MLP achieves an accuracy of 92%. For the more effective direction sync attack, Movee using Decision Tree (C4.5) achieves an 84% accuracy.

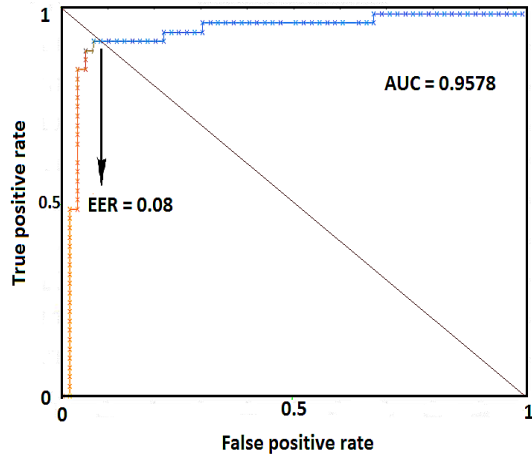
Attack	Classifier	Acc(%)	TPR(%)	FPR(%)	FNR(%)
Cluster	MLP	70.26	74.46	34.08	25.54
	RF	73.46	78.14	31.39	21.86
	Bagging	72.25	77.27	32.95	22.72
	C4.5	70.04	74.24	34.30	25.76
Replay	MLP	65.81	70.93	39.9	29.07
	RF	67.44	72.03	37.68	27.97
	Bagging	67.91	71.81	38.42	28.19
	C4.5	63.84	69.61	42.61	30.39

Table 4.5: Detailed accuracy results of Movee on the smartphone cluster and replay attack datasets. For both the cluster and replay attacks, Bagging achieves the best accuracy of 73% and 68% respectively.

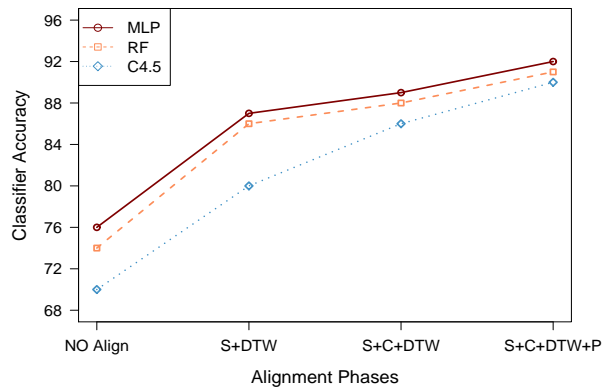
Table 4.4. For the random attack dataset, the three classifiers perform similarly, with MLP being the top performer (92% accuracy). For the direction sync attack dataset, Decision Tree (C4.5) outperforms MLP and RF with 84% accuracy.

Figure 4.15(a) shows the ROC curve and the computed EER and AUC values for the MLP classifier on the random dataset. The EER value of MLP is as small as 0.08 (maximum is 0.5) and the area under the curve exceeds 0.95, denoting an accurate classifier.

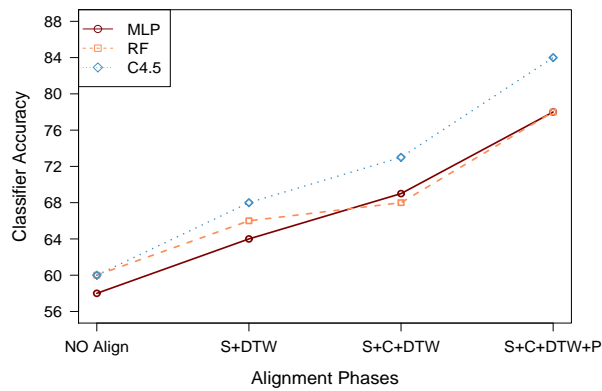
We have also evaluated the impact of each step of the SC module on the accuracy of Movee, for the random and direction sync attack datasets. For each dataset, we measured the accuracy of the three classifiers when (i) no alignment phase was applied, (ii) when stretching and DTW were applied, (iii) for stretching, calibration and DTW, and (iv) for stretching, calibration and penalty based DTW. Figure 4.15(b) shows the results for the random attack dataset and Figure 4.15(c) shows the results for the direction sync attack datasets. The stretching step contributes the most to the accuracy of Movee for the random attack while the penalization step contributes the most for the direction sync attack (around 12%).



(a)



(b)



(c)

Figure 4.15: Smartphone data evaluation. (a) ROC curve on random dataset for Movee (using MLP). (b) Impact of SC steps on Movee’s accuracy for the random attack. (c) Impact of SC steps on Movee’s accuracy for the direction sync attack.

Attack	Classifier	Acc(%)	TPR(%)	FPR(%)	FNR(%)
Random	MLP	90	87.5	7.7	12.5
	RF	90	89.6	9.6	10.41
	RT	91	88.89	6.52	11.11
Dir sync	MLP	72	74.5	31.1	25.4
	RF	74	61.4	16.1	38.6
	RT	79	68.2	12.5	31.8

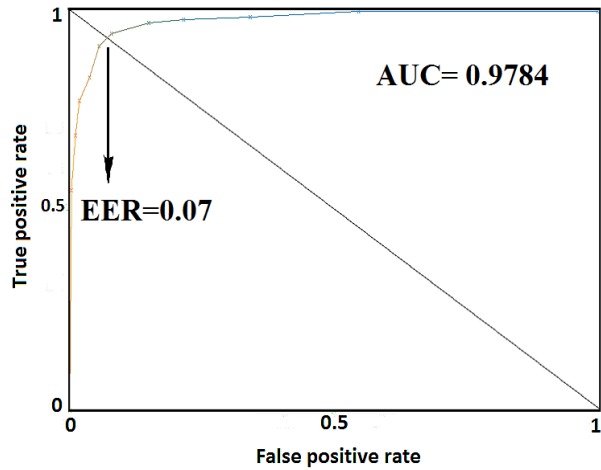
Table 4.6: Detailed accuracy parameters of Movee Glassware (using different classifiers) for all three attack datasets. “Acc” denotes the accuracy of the classifier.

Table 4.5 shows the results of our experiments on the cluster and replay attack datasets. The replay attack is more effective. The Bagging algorithm achieves the best performance on both attacks, of 73% for the cluster and 68% for the replay attack. Figure 4.14 summarizes the performance of the best 4 classifiers on the 4 attacks considered, on smartphone captured data. The more complex cluster and replay attacks are more efficient.

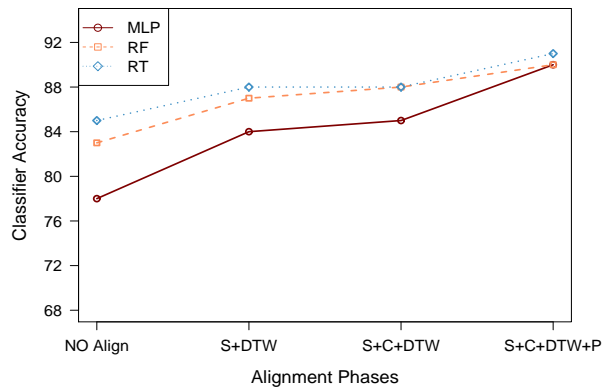
4.6.4 MoveeG Attack Detection

We first investigate the accuracy of MoveeG running on Google Glass to classify genuine and fraudulent video samples, on the random and direction sync attack. Table 4.6 shows the detailed results of the 3 best performing classifiers (RF, RT and MLP). RT performs best on the random and direction sync attack datasets. Figure 4.16(a) shows the ROC curve and the computed EER value for the RF classifier and the random attack dataset for MoveeG. The EER value of RF is 0.07 and the area under the curve (AUC) exceeds 0.97.

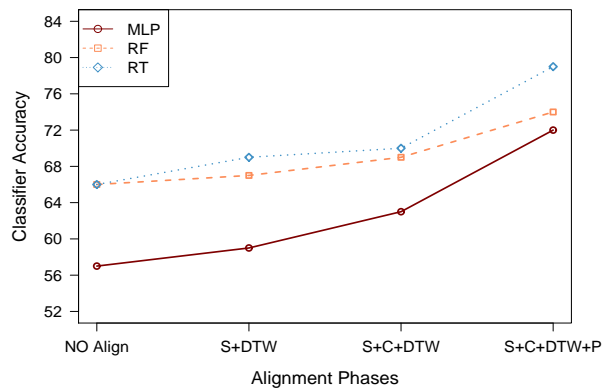
Figure 4.16(b) shows the impact of the steps in the SC module for the random attack dataset and Figure 4.16(c) shows their impact for the direction sync attack dataset. The stretching step contributes the most to the accuracy of MoveeG for the



(a)



(b)



(c)

Figure 4.16: (a) ROC curve (using MLP) on random attack dataset for MoveeG. (b) The impact of SC steps on the MoveeG accuracy for the random attack. (c) The impact of SC steps on the MoveeG accuracy for the direction sync attack.

Attack	Classifier	Acc(%)	TPR(%)	FPR(%)	FNR(%)
Cluster	RF	78.36	79.3	22.5	20.7
	MLP	72.45	75.1	30.0	24.9
	RT	76.34	72.0	19.6	28.0
Replay	RF	77.86	81.8	26.0	18.2
	MLP	62.76	67.7	42.2	32.3
	RT	74.74	78.1	28.6	21.9

Table 4.7: Detailed accuracy results of MoveeG on the Glass cluster and replay attack datasets. Random Forest achieves best accuracy for both cluster and replay attacks. Movee is more accurate on Glass than on smartphone.

random attack (5%). For the direction sync attack, the penalization step contributes the most (around 8%).

We now evaluate Movee on the cluster and replay attacks performed on the data captured on the Google Glass device (see Section 4.5.2). Table 4.7 shows the results of the experiments. Similar to the smartphone investigation, the replay attack is slightly more effective than the cluster attack. Both attacks are more effective than the random and direction sync attacks. The Random Forest classifier achieves best accuracy both for the cluster (78%) and for the replay attack (77%). We observe a surprising result: Movee has higher accuracy on the cluster and replay attacks performed on Glass videos when compared to the smartphone videos. We conjecture that for the replay attack, the reason for this stems from the fact that head mounted cameras capture more complex motions, that are harder to emulate.

Figure 4.17 summarizes the accuracy of Movee on all 4 attacks performed on Google Glass data. Different from the smartphone data investigation, where the cluster and replay attacks are more efficient, we note that on Google Glass data, MoveeG with Random Forest achieves higher accuracy on the replay attack than on the direction sync attack.

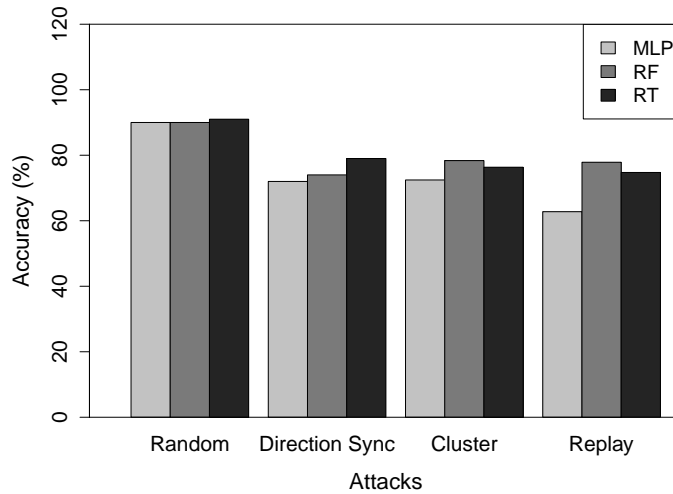


Figure 4.17: Summary of Movee accuracy on data collected from Google Glass, when different classifiers are used.

4.6.5 MoveeG Battery Impact

MoveeG impacts the battery lifetime of the Google Glass device. To properly evaluate and profile the components responsible, we first turned off the wireless interfaces and stopped all applications running on the Google Glass. We then performed the following experiments, each starting with a full battery charge. First, we ran MoveeG until the device ran out of battery. Second, we separately ran only the video capture, and third, we ran only the acceleration stream capture components of MoveeG. We have logged the battery level of the Google Glass device during these experiments.

Figure 4.18 shows our results: when recording only acceleration data, the battery lasts 209 minutes, when recording only video the battery lasts 102 minutes, and when recording both (MoveeG), it lasts 86 minutes. As a baseline, we also ran the experiment with no application running on the Glass: on average, the battery lasted 3034 minutes. We note that the device turns itself to sleep when not in use.

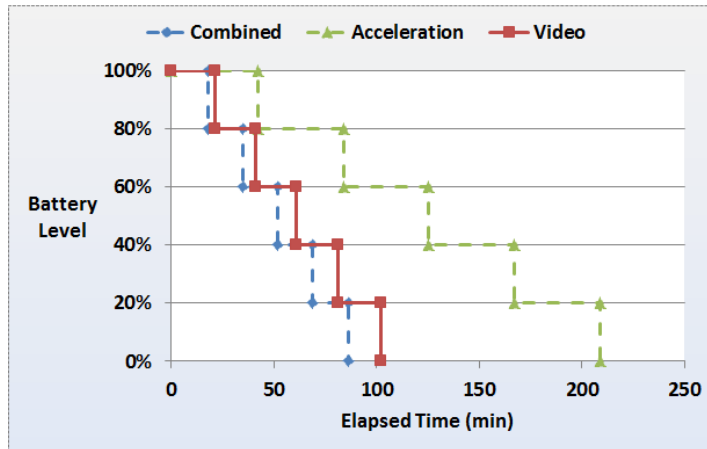


Figure 4.18: Impact of video and acceleration recording on Google Glass battery lifetime. The video recording activity halves the battery lifetime when compared to the acceleration recording activity.

This experiment shows that keeping the device continuously active even when only capturing acceleration data, significantly reduces the battery lifetime to 6%. The video capture activity further reduces the battery lifetime by 50% when compared to the acceleration capture activity.

We first report our experience in classifying the collected video datasets. We then evaluate the ability of CL-Vamos to classify 6s chunks from the free-form dataset, as either genuine or fraudulent. Finally, we evaluate the performance of Vamos on the whole length samples from the free-form dataset.

4.6.6 Video Dataset Classification

Two users (paper authors) have manually annotated the YouTube and free-form datasets based on the criteria described in Section 4.2.3. Since a single video can include sections belonging to different motion categories, the result of the annotation process consists of tuples of the form $(start_time, end_time, category_id)$, where the first two fields denote the start and end time of a video section (measured in seconds) and the last field denotes the id of the video category (integer ranging from 1 to

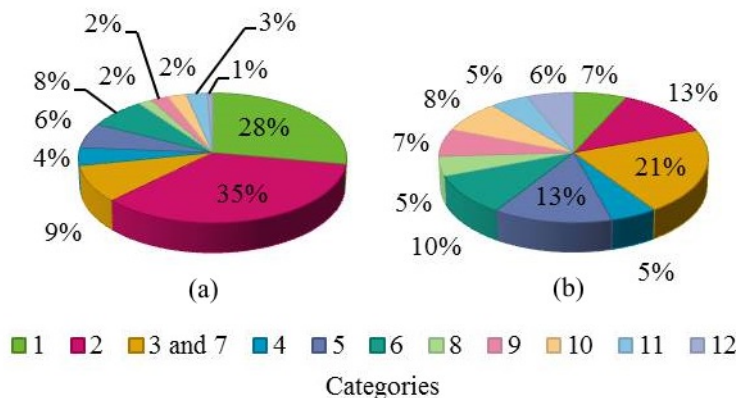


Figure 4.19: (a) Motion category distribution for YouTube dataset. (b) Distribution for free-form dataset. Table 4.1 defines the 12 categories.

12). At the end of the process, we have computed a tally of the number of seconds of video belonging to each of the 12 video motion categories.

We have noticed discrepancies between the annotations of the two users: a walking user recording a nearby scene without moving the camera, produces a video that can be (visually) categorized as either 3 or 7. We have labeled these video with a category denoted 3&7.

Figure 4.19(a) shows the resulting category distribution of the YouTube dataset. The motion categories 2 and 1 have the largest representation, whereas category 12 has the smallest representation. Figure 4.19(b) shows the category distribution of the free-form dataset, and Table 4.3 shows the category distribution of the free-form chunks. The difference in distributions of the YouTube and free-form datasets is likely due to the fact that the free-form video collection scenarios have different dynamics from citizen journalism scenarios.

4.6.7 CL-Vamos on Motion Categories

Experiment setup. CL-Vamos and Vamos use trained classifiers to determine if a video is genuine. We have experimented with several classifiers, including Mul-

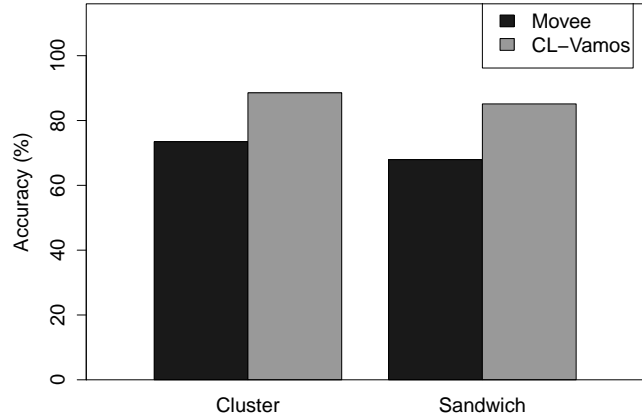


Figure 4.20: Accuracy of CL-Vamos and Movee. CL-Vamos improves by more than 15% on Movee, for both cluster and sandwich attacks.

tiLayer Perceptron (MLP) [Gal90], Decision Trees (DT) (C4.5), Random Forest (RF) [Bre01] and Bagging [Bre96].

We have used the Weka data mining suite [Wek] to perform the experiments, with default settings. For the backpropagation algorithm of the MLP classifier, we set the learning rate to 0.3 and the momentum rate to 0.2.

CL-Vamos vs. Movee. In a first experiment we compare the efficacy of CL-Vamos and Movee [RTC13] using the following variation of cross validation. For each category c , we split the data into 10 equal sized folds. Then, in each of 10 iterations, we train the classifier on 9 folds from all the categories, and test on the data of the remaining fold from c . Repeat this operation 10 times, ensuring each fold appears once in the test dataset.

Figure 4.20 summarizes our results. On the cluster attack, CL-Vamos achieves 88% accuracy when using MLP (Random Forest 83%, Bagging 81%, Decision Trees 78% and SVM 80%). Movee achieves highest accuracy when using the Random Forest (73%). On the sandwich attack, CL-Vamos achieves 85% accuracy when

Category	TPR(%)	FPR(%)	FNR(%)	Acc(%)
1	75.0	16.67	25.0	80.0
2	82.13	16.67	17.87	83.33
3&7	87.97	27.0	12.03	77.08
4	75.0	25.0	25.0	75.0
5	80.0	2.86	20.0	83.33
6	68.33	13.9	31.67	79.17
8	75.0	0.0	25.0	80.0
9	77.66	16.67	22.34	80.0
10	91.67	38.86	8.33	75.0
11	83.25	6.25	16.75	85.0
12	75.0	25.0	25.0	81.25

Table 4.8: CL-Vamos accuracy on cluster attack is as high as 85% (on category 11.)

using Random Forest (MLP 71%, Bagging 78%, Decision Trees 74% and SVM 80%). Movee achieves the highest, 67% accuracy, when using either Random Forest or Bagging classifiers. This substantial improvement of CL-Vamos corresponds to an FNR of 6-14% and FPR of 15-17% on these attacks. In contrast, Movee’s FNR is between 21-28% and FPR is between 31-38%.

CL-Vamos: per-category efficacy. Table 4.8 shows the TPR, FPR, FNR and accuracy results for CL-Vamos on the cluster chunk dataset, on each of the 11 motion categories. The accuracy ranges between 75% and 85%. Table 4.9 shows the per-category TPR, FPR, FNR and accuracy results of CL-Vamos on the sandwich chunk dataset. The accuracy ranges from 68% to 88%. We conjecture that its good accuracy for several video categories is due to the difficulty for a human attacker to correctly emulate the movement of the camera, including to estimate distances, observed in a video.

Category relevance. We now verify the intuition that the variation in FNR, FPR and accuracy of CL-Vamos is due to its dependence on the video motion categories. We have performed both Pearson’s χ^2 and Fisher’s exact test on the

Category	TPR(%)	FPR(%)	FNR(%)	Acc(%)
1	75.0	10.0	25.0	84.0
2	66.67	16.67	33.33	73.33
3&7	81.34	22.58	18.66	78.75
4	83.34	12.5	16.66	80.0
5	66.0	31.32	34.0	68.75
6	69.34	28.0	30.66	70.0
8	86.0	6.66	14.0	88.0
9	74.34	20.0	25.66	84.0
10	66.67	8.0	33.33	77.14
11	83.34	27.34	16.66	72.0
12	76.68	0.0	23.32	85.0

Table 4.9: CL-Vamos accuracy on sandwich attack ranges from 68% to 88%.

results CL-Vamos for the sandwich attack dataset. The null hypothesis is that the true positive, false positive, true negative and false negative values are independent of the proposed video categories. The χ^2 's p-value is 0.0001166 and Fisher's p-value is 0.00015. Thus, we reject the null hypothesis and conclude that the performance of CL-Vamos depends on the video motion category.

Experiment conclusion. While we expected that certain motion categories are easier to plagiarize, our results are surprising: CL-Vamos does not perform worst on categories 1 and 2, captured by a standing user with a stationary camera. Based on observations from our experiments, we believe that CL-Vamos exploits the ability of accelerometers to capture the small, involuntary hand shakes that occur during video capture sessions. Instead, CL-Vamos has high FPR values for the sandwich attack on categories 5, 6 and 11. This shows that in our experiments, humans are better at plagiarizing videos shot while scanning or following subjects. In Section 4.6.9 we show that Vamos' overall accuracy exceeds 93% even for the sandwich attack.

4.6.8 CL-Vamos on Citizen Journalism

Current YouTube videos do not have acceleration data. CL-Vamos however only works for video chunks for which we have acceleration data. We propose to use the classification of the collected YouTube videos (see Section 4.6.6) and the performance of CL-Vamos on the free-form video and acceleration samples (see Section 4.6.7) to predict its performance on fixed length chunks of citizen journalism videos from YouTube.

Let $Acc(i, FreeForm, AT)$ denote the accuracy of CL-Vamos on videos from the i -th category ($i=1..11$) of the free-form dataset, for a given attack type AT . We define the predicted accuracy of CL-Vamos for YouTube and the attack type AT , $Acc_p(YouTube, AT)$, as the weighted sum of its per-category accuracy on the free-form dataset:

$Acc_p(YouTube, AT) = \sum_{i=1}^{11} w_i \times Acc(i, FreeForm, AT)$. We define the weight w_i to be the percentage of chunks of category i in the YouTube dataset, as shown in Figure 4.19(a). In the YouTube dataset categories 1 and 2 have the highest weight. The predicted accuracy of CL-Vamos for the cluster attack on the YouTube dataset is then 80.9%, and for the sandwich attack is 77.19%.

4.6.9 Vamos Evaluation

We now evaluate the performance of Vamos on entire video and acceleration samples. We note that a sample can consist of multiple chunks that belong to different motion categories. We have performed experiments using the stitch attack datasets (based on chunk-level cluster and sandwich attacks) described in Section 4.5.4. The stitch attack datasets consist of both genuine and fraudulent free-form samples.

Vamos makes the sample level classification decision based on the classification of the chunks of the sample. In order to avoid a case where the same chunk appears

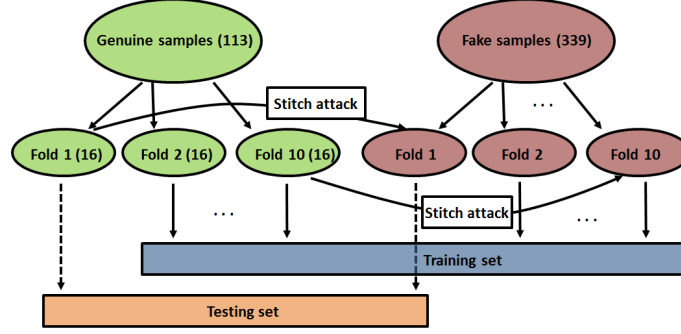


Figure 4.21: Setup of Vamos experiment. Each genuine fold produces a stitch attack fold. In each experiment, 9 genuine folds and the corresponding stitch attack folds are used for training. The rest are used for testing.

in both training and testing sets, we propose the following experimental design, illustrated in Figure 4.21.

First, divide the dataset of 113 samples of at least 2 chunks each, into k folds, $gen.fold(i)$, $i = 1..k$, and the corresponding 339 attack sample dataset (either cluster or sandwich attack based) into k folds, $attack.fold(i)$, $i = 1..k$. The split takes place such that the samples from the $gen.fold(i)$ were used to generate the attack samples from

$attack.fold(i)$. Then, for each $i = 1..k$, pick all the samples from $gen.fold(j)$ and $attack.fold(j)$, $j = 1..k$, $j \neq i$, and use their chunks to train CL-Vamos. Run the trained CL-Vamos on all the chunks from $gen.fold(i)$ and $attack.fold(i)$. Given the classified chunks of the samples from $gen.fold(i)$ and from $attack.fold(i)$, run the sample level classification step to classify the samples. For instance, to compute $Pr(S = fake)$ for a sample S , compute $Pr(a_i = genuine)$ and $Pr(a_i = fake)$ based on the number of fake and genuine chunks in the training folds (see Section 4.3.5). In our experiments, we set k to 10.

Experiment results. Table 4.10 reports the performance of Vamos on the cluster based stitch attack dataset. We have experimented with multiple threshold values. For majority voting, a threshold of 0.1 performed best: both FPR and FNR values

Algo	TPR(%)	FPR(%)	FNR(%)	Acc(%)
Maj. Vote	91.69	7.95	8.31	91.78
Prob.	91.69	7.95	8.31	91.78
Bagging	97.35	5.08	2.65	95.53

Table 4.10: Vamos efficacy on cluster based stitch attack. The classifier approach performs best.

Algo	TPR(%)	FPR(%)	FNR(%)	Acc(%)
Maj. Vote	74.19	35.83	25.81	71.69
Prob.	69.95	32.50	30.05	69.34
Bagging	83.7	3.63	16.3	93.199

Table 4.11: Vamos performance on sandwich/stitch attack. The classifier approach performs best.

are under 9%. For the probabilistic approach, a threshold of 0.7 achieves similar performance. However, we note that the classifier approach, when using the Bagging algorithm, significantly outperforms the other solutions, with an FPR of around 5% and an FNR of 2.65%.

Table 4.11 shows the performance of the majority voting, probabilistic and classifier based approaches of Vamos, on the sandwich based stitch attack dataset. For majority voting and probabilistic approaches, the sandwich based stitch attack is significantly more efficient: The majority voting has no threshold where both FPR and FNR are below 35%. The probabilistic approach achieves its optimum for a threshold of 0.8, when its FPR and FNR values are barely under 35%. In contrast, the classifier approach, again when using Bagging, exhibits a significantly improved performance, with an FPR of under 4% and an FNR of 16.3%. Figure 4.22 summarizes the accuracy of the three approaches of Vamos for the cluster and sandwich based stitching attacks.

We emphasize the importance of this result: while the Movee [RTC13] algorithm exhibits an accuracy of 60-70% on fixed length chunks, Vamos achieves an accuracy

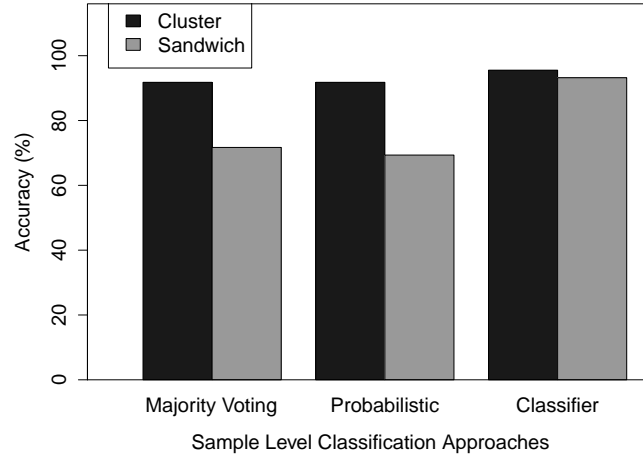


Figure 4.22: Vamos accuracy on stitch attacks. Even for the sandwich stitch attack, the classifier approach (using Bagging) exceeds 93% accuracy.

that exceeds 93% even on arbitrary length videos, under a *combination* of potent attacks.

4.7 Limitations

Vamos is designed to work with video streams captured by directly pointing the camera to the scene. Vamos’s liveness verifications will likely not work for videos captured in indirect ways (e.g., through reflection or refraction), or videos encoding fast motion. Vamos will have difficulty extracting accurate motion information from video frames containing occlusions, illumination changes and blur.

We have also not experimented with very short videos (less than 6s) or with videos shot in unusual circumstances: involving very high accelerometer activity, e.g., running, or when the user is in a moving vehicle. Due to the lack of gyroscope sensors in the Samsung Admire device, we have not integrated gyroscope readings to verify camera rotation movements.

Furthermore, we have not experimented with doctored video and accelerometer streams. For instance, given an input video, the attacker can use the work of Davison et al. [DRMS07] to recover the 3D trajectory of the camera. Then, given root access (e.g., using [Unl, Roo], create a corresponding accelerometer sample and feed it to Vamos, e.g., using a solution similar to [XPr]. We defer the task of providing trust for the integrity of the mobile app, as well as the trust for the integrity of the device’s connection to its camera and accelerometer sensors to the providers of Vamos. Establishing the integrity of a mobile platform and mobile apps is currently an active area of research and is also being addressed by products readily available in the market [SFE10, Arx, Six11].

As mentioned in Section 4.3, we assume a trusted device, operating system and device drivers (e.g., accelerometer), achieved e.g., using chain of trust solutions. This assumes an attacker with reasonable limitations. That is, circumventing the defenses would require the attackers to incur overwhelming time and effort costs. A successful attacker to a trusted app inside a trusted system satisfying our assumptions would have to successfully inject unsigned code into an operating system which uses chain of trust (e.g. iOS [App]) and modify an app which is protected by cryptographic mobile app verification (e.g. Arxan [Arx]) at the same time. Alternatively, a successful attacker who can produce both the fraudulent video and the corresponding acceleration streams would have to use a computer generated (CGI) video stream. Vamos would not be effective in either scenario.

We have not experimented with “green screen” attacks, where the attacker captures a video with a portion of the scene being a green screen. Following the video capture, the attacker overlays additional video footage or static images on the green section. We note that Vamos raises the bar here: an attacker needs to invest in ad-

ditional equipment to thwart the defenses of Vamos. The quality of the equipment determines the (in)ability of a human observer to detect the attack.

We have not evaluated Vamos against a sandwich attack variant, where a robotic arm [Ban, Rob] holding the mobile device is used to reproduce the motion observed in a target video. While low cost, easy to program robotic arms do exist, they are only capable of jerky, robot-like movements. Such movements are likely different from the fluid movements encoded in human captured video streams; they can thus be detected by Vamos. Therefore, in order for such a robot based attack to bypass Vamos, the adversary needs to invest in a system able to fluidly replicate the wide variety of whole body movements. However, robotic arms that are capable of fluid, human-like movements are significantly more expensive. Thus, Vamos raises the bar for attackers, that need to invest in expensive components.

Furthermore, we have introduced and evaluated Vamos against manual, automatic and mixed attacks. We leave the exhaustive exploration of the attack space for future work.

4.8 Summary

In this chapter we have introduced the concept of “liveness” analysis, of verifying that a video has been shot live on a mobile device. We have proposed Movee, a system that relies on the accelerometer sensors ubiquitously deployed on most recent mobile devices to verify the liveness of a simultaneously captured video stream. We have implemented Movee, and, through extensive experiments, we have shown that (i) it is efficient in differentiating fraudulent and genuine videos and (ii) imposes reasonable overheads on the server.

We also proposed Vamos, the first length and motion un-constrained video liveness verification system. Vamos uses the entire video and acceleration streams to

identify video fraud. We introduced a motion based classification of videos. We evaluated Vamos on data collected from a user study and on citizen journalism videos from YouTube. Vamos has an accuracy exceeding 93% on novel, complex attacks.

CHAPTER 5

SECURE MANAGEMENT OF DATA STORAGE AND COMMUNICATION IN SOCIAL SENSOR NETWORKS

In this chapter, we will address the problem of securing data storage and communication in wearable fitness trackers being used in social sensor networks. During my Ph.D study, part of the following content has been published.

In the following, the motivation and challenges to this problem will be introduced in Section 5.1. In Section 5.2, the system components, general assumptions about the adversaries and reverse-engineering of Fitbit and Garmin’s protocols will be described briefly. In Section 5.3, the discovered security and privacy vulnerabilities during reverse-engineering of Fitbit and Garmin’s protocol will be described and then in Section 5.4, the proposed solution framework and the detailed solution will be presented. In Section 5.5, several discussions about the security aspects of our solution and its practical feasibility will be presented. After that, the efficiency and efficacy of the solution will be demonstrated and a prototype system based on our solution will be described in Section 5.6. One limitation of our work will be mentioned in Section 5.7. Finally, the summary of this chapter will be given in Section 5.8.

5.1 Motivation and Challenges

The increasing popular interest in personal telemetry, also called the Quantified Self or “lifelogging”, has induced a popularity surge for wearable personal fitness trackers. Fitness trackers automatically collect sensor data about the user throughout the day, and integrate it into social network accounts. This data-centric lifestyle, “lifelogging” is now producing massive amounts of intimate personal data. For instance, BodyMedia [Bod] has created one of the world’s largest libraries of raw and

real-world human sensor data, with 500 trillion data points [BMD]. This data is becoming the source of privacy and security concerns: information about locations and times of user fitness activities can be used to infer surprising information, including the times when the user is not at home [Ple], and company organizational profiles [TKS13].

We demonstrate vulnerabilities in the storage and transmission of personal fitness data in popular trackers from Fitbit [Fit] and Garmin [For]. Vulnerabilities have been identified for similar systems, including pacemakers (e.g., Halperin et al. [HHBR⁺08]) and glucose monitoring and insulin delivery systems (e.g., Li et al. [LRJ11]). The differences in the system architecture and communication model of social sensor networks enable us to identify and exploit different vulnerabilities.

The attacks are fast, thus practical even during brief encounters. We believe that, the vulnerabilities that we identified in the security of Fitbit and Garmin are due to the many constraints faced by solution providers, including time to release, cost of hardware, battery life, features, mobility, usability, and utility to end user. Unfortunately, such a constrained design process often puts security in the back seat. Solution providers have to strike a balance between many constraints, leading to a design process that often puts security in the back seat.

These systems need a solution for secure fitness data storage and transmission which protects not only against inspect and inject attacks, but also against attackers that physically capture and read the memory of trackers. The solution's hardware and computation requirements should be minimal, just enough to perform low-cost operations on the tracker and the solution should avoid imposing storage overhead on trackers.

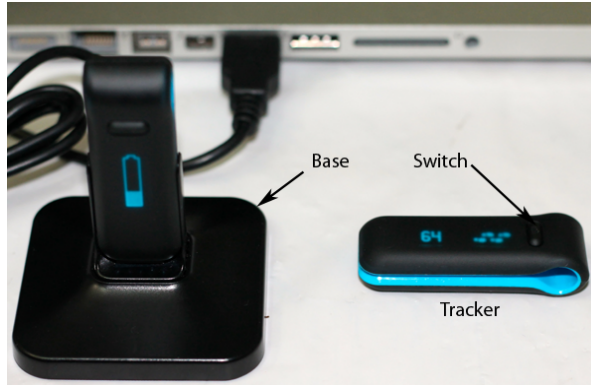
To help address these constraints, in this paper we introduce SensCrypt, a protocol for secure fitness data storage and transmission on lightweight personal trackers.

We leverage the unique system model of social sensor networks to encode data stored on trackers using two pseudo-random values, one generated on the tracker and one on the providing server. SensCrypt thwarts not only the attacks we introduced, but also defends against powerful JTAG Read attacks. SensCrypt’s hardware and computation requirements are minimal, just enough to perform low-cost symmetric key encryption and cryptographic hashes. SensCrypt does not impose storage overhead on trackers and ensures an even wear of the tracker storage, extending the life of flash memories with limited program/erase cycles.

SensCrypt is applicable to a range of sensor based platforms, that includes a large number of popular fitness [Fit, For, Jaw, Mota, Bas] and home monitoring solutions [Nes, WeM, Motb], as well as scenarios where the sensors need to be immobile and operable without network connectivity (e.g., infrastructure, traffic, building and campus monitoring solutions). In the latter case, the bases through which the sensors sync with the webserver are mobile, e.g., smartphones of workers, who may become proximal to the sensors with the intention of data collection or as a byproduct of routine operations.

We have developed Sens.io, a \$52 tracker platform built on Arduino Uno, of similar capabilities with current solutions. On Sens.io, SensCrypt (i) imposes a 6ms overhead on tracker writes, (ii) reduces the end-to-end overhead of data uploads to 50% of that of Fitbit, and (iii) enables a server to support large volumes of tracker communications. While SensCrypt’s defenses may not be immediately adopted by existing products ¹, this paper provides a foundation upon which to create, implement and test new defensive mechanisms for future tracker designs.

¹We have contacted Fitbit and Garmin with our results. While interested in the security of their users, they have declined collaboration.



(a)



(b)

Figure 5.1: System components: (a) Fitbit: trackers (one cradled on the base), the base (arrow indicated), and a user laptop. The arrow pointing to the tracker shows the switch button, allowing the user to display various fitness data. (b) Garmin: trackers (the watch), the base (arrow indicated), and a user laptop.

5.2 System Model, Attacker and Background

5.2.1 System Model

We consider a general system consisting of tracker devices, base stations and an online social network. We exemplify the model components using Fitbit Ultra [Fit] and Garmin Forerunner [For], two popular health centric social sensor networks (see Figures 5.1(a) and 5.1(b)). For simplicity, we will use “Fitbit” to refer the Fitbit Ultra and “Garmin” to denote the Garmin Forerunner 610 solution.

The tracker. The *tracker* is a wearable device that records, stores and reports a variety of user fitness related metrics. We focus on the following trackers:

- **The Fitbit tracker** measures the daily steps taken, distance traveled, floors climbed, calories burned, the duration and intensity of the user exercise, and sleep patterns. It consists of four IC chips, (i) a MMA7341L 3-axis MEMS accelerometer, (ii) a MEMS altimeter to count the number of floors climbed and (iii) a MSP 430F2618 low power TI MCU consisting of 92 KB of flash and 96 KB of RAM. The user can switch between displaying different real-time fitness information on the tracker, using a dedicated hardware *switch* button (see the arrow pointing to the switch in Figure 5.1(a)).
- **The Garmin tracker** records data at user set periodic intervals (1-9 seconds). The data includes a timestamp, exercise type, average speed, distance traveled, altitude, start and end position, heart rate and calories burned during the past interval. The tracker has a heart rate monitor (optional) and a 12 channel GPS receiver with a built-in SiRFstarIII antenna. that enables the user to tag activities with spatial coordinates.

Both Fitbit and Garmin trackers have chips supporting the ANT protocol, with a 15ft transmission range for Fitbit and 33ft for Garmin. Each tracker has a unique id, called the *tracker public id* (TPI). Trackers also store profile information of their users, including age, gender and physiological information such as height, weight and gait information.

The base and agent module. The base connects with the user’s main computing center (e.g., PC, laptop) and with trackers within transmission range (15ft for Fitbit and 33ft for Forerunner) over the ANT protocol. The user needs to install an “agent module”, a software provided by the service provider (Fitbit, Garmin) to run on the base. The agent and base act as a bridge between the tracker and the online social

network. They upload information stored on the tracker to its user account on the webserver, see Figures 5.1(a) and 5.1(b) for system snapshots.

Tracker to base pairing. Fitbit trackers communicate to any base in their vicinity. However, tracker solutions like Garmin Forerunners allow trackers to communicate only through bases to which they have been previously “paired” or “bonded”. Garmin’s pairing procedure works in the following manner. The agent running on the base searches for available ANT enable devices. Each tracker periodically sends broadcast beacons over the ANT interface. If the agent discovers a tracker, it extracts its unique id (TPI). The agent uses one of two methods of authentication: initial *pairing* or *passkey*. The agent verifies if it already stores an authfile for this TPI. If no such file exists (i.e., this is the first time the tracker is pairing with the base), the agent uses the *pairing* method and sends a bind request to the tracker. When prompted, the user needs to authenticate the operation, through the push of a button on the tracker. The agent then retrieves a factory embedded “passkey” from the tracker. It then stores the pair $\langle TPI, passkey \rangle$ in a newly created authfile. During subsequent authentications, the agent uses the *passkey* method: it recovers the passkey corresponding to the TPI from the authfile and uses it to authenticate the tracker.

The system model considered can be extended to cover the case of fitness tracking solutions that turn the user’s mobile device into a base, e.g., [Jaw, Nik, Bas]. In such systems, the agent module is a mobile app running on the mobile device. The tracker communicates with the smartphone over existing network interfaces, e.g., Bluetooth or NFC. We note that Naveed et al. [NZD⁺14] identified an intriguing vulnerability of Android smartphones bonded to health trackers. The vulnerability stems from the fact that the bonding occurs at smartphone device level not at the

app level. This effectively leaves the health data vulnerable to rogue apps with Bluetooth permissions.

The webservice. The online social network webservice (e.g., fitbit.com, connect.garmin.com), allows users to create accounts from which they befriend and maintain contact with other users. Upon purchase of a tracker and base, the user binds the tracker to her social network account. Each social network account has a unique id, called the *user public id* (UPI). When the base detects and sets up a connection with a nearby tracker, it automatically collects and reports tracker stored information (step count, distance, calories, sleep patterns) with temporal and spatial tags, to the corresponding user's social network account. In the following, we use the term *webservice* to denote the computing resources of the online social network.

Tracker-to-base communication: the ANT protocol. Trackers communicate to bases over ANT, a 2.4 GHz bidirectional wireless Personal Area Network (PAN) ultra-low power consumption communication technology, optimized for transferring low-data rate, low-latency data.

Data conversion. The Fitbit tracker relies on the user's walk and run stride length values to convert the step count into the distance covered. It then extrapolates the user's Basal Metabolic Rate (BMR) [HE04] values and uses them to convert the user's daily activities into burned calories values. The Garmin tracker uses the GPS receiver to compute the outdoor distance covered by the user. It then relies on the Firstbeat[Fir05] algorithm to convert user data (gender, height, weight, fitness class) and the captured heart rate information to estimate the user's Metabolic Equivalent (MET), which in turn is used to retrieve the calories burnt.

5.2.2 Attacker Model

We assume that the webserver is honest, and is trusted by all participants. We assume adversaries that are able to launch the following types of attacks:

Inspect attacks. The adversary listens on the communications of trackers, bases and the webserver.

Inject attacks. The adversary exploits solution vulnerabilities to modify and inject messages into the system, as well as to jam existing communications.

Capture attacks. The adversary is able to acquire trackers or bases of victims. The adversary can subject the captured hardware to a variety of other attacks (e.g., Inspect and Inject) but cannot access the memory of the hardware. We assume that in addition to captured devices, the adversary can control any number of trackers and bases (e.g., by purchasing them).

JTAG attacks. JTAG and boundary scan based attacks (e.g., [Bre06]), extend the Capture attack with the ability to access the memory of captured devices. We focus here on “JTAG-Read” (JTAG-R) attacks, where the attacker reads the content of the *entire* tracker memory.

5.2.3 Reverse Engineering Fitbit and Garmin

Our goal in reverse-engineering the Fitbit Ultra and Garmin Forerunner protocols was dual, (i) to understand the source(s) of vulnerabilities and (ii) to develop security solutions that are interoperable with these protocols. Sec. 103(f) of the DMCA (17 U.S.C. 1201 (f)) [Reva] states that a person who is in legal possession of a program, is permitted to reverse-engineer and circumvent its protection if this is necessary in order to achieve “interoperability”.

To log communications between trackers and webserver, we wrote USB based filter drivers and ran them on a base. We have used Wireshark to capture all

wireless traffic between the agent software and the webserver. To reverse engineer Fitbit, we exploited (i) the lack of encryption in all its communications and (ii) libfitbit [lib], a library built on ANT-FS [ANT] for accessing and transferring data from Fitbit trackers. Unlike Fitbit, Garmin uses HTTPS with TLS v1.1 to send user login credentials. However, similar to Fitbit, all other communications are sent over plaintext HTTP.

Fitbit and Garmin bases both use *service logs*, files that store information concerning communications involving the base. Garmin’s logs consist of an “authfile” for each tracker that was paired with the base, and .FIT files. The authfile contains authentication information for each tracker. Forerunner maintains 20 types of .FIT files, each storing a different type of tracker data, including information about user activities, schedules, locations and blood pressure readings. On the Windows installation of the Fitbit software, daily logs are stored in cleartext in files whose names record the hour, minute and second corresponding to the time of the first log occurrence. Each request and response involving the tracker, base and social network is logged and sometimes even documented in the archive folder of that log directory.

In the following, we first focus on Fitbit’s tracker memory organization and communication protocol.

Fitbit: Tracker memory organization. A tracker has both *read banks*, containing data to be read by the base and *write banks*, containing data that can be written by the base. The read banks store the daily user fitness records. The write banks store user information specified in the “Device Settings” and “Profile Settings” fields of the user’s Fitbit account. The tracker commits sensor values (step, floor count) to the read bank once per minute. The tracker can store 7 days worth of 1-per-minute sensor readings [Fit13].

The webservice communicates with the tracker through XML blocks, that contain base64 encoded commands, or *opcodes*. Opcodes are 7 bytes long. We briefly list below the most important opcodes and their corresponding responses. The opcode types are also shown in Figure 5.2.

- **Retrieve device information (TRQ-REQ):** opcode [0x24,000000]. Upon receiving this opcode from the webservice (via the base), the tracker sends a reply that contains its serial number (5 bytes), the hardware revision number, and whether the tracker is plugged in on the base.
- **Read/write tracker memory (READ-TRQ/WRITE).** To read a memory bank, the webservice needs to issue the READ-TRQ opcode, [0x22, *index*,00000], where *index* denotes the memory bank requested. The response embeds the content of the specified memory bank. To write data to a memory bank, the webservice issues the WRITE opcode [0x23, *index*, *datalen*,0000]. The payload data is sent along with the opcode. The value *index* denotes the destination memory bank and *datalen* is the length of the payload. A successful operation returns the response [0x41,000000].
- **Erase memory: (ERASE)** opcode [0x25, *index*, *t*, 0]. The webservice specifies the *index* denoting the memory bank to be erased. The value *t* (4 bytes, MSB) denotes the operation deadline - the date until which the data should be erased. A successful operation returns the response [0x41,000000].

Fitbit: The communication protocol. The communication between the webservice and the tracker through the base, is embedded in XML blocks, that contain base64 encoded opcodes: commands for the tracker. All opcodes are 7 bytes long and vary according to the instruction type (e.g., TRQ-REQ, READ-TRQ, WRITE, ERASE, CLEAR). The system data flow during the data upload operation is shown in Figure 5.2.

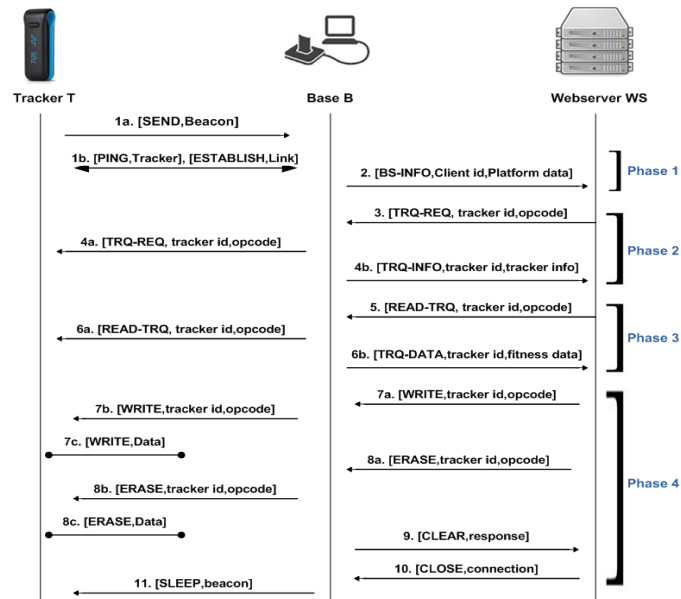


Figure 5.2: Fitbit Upload protocol. Enables the tracker to upload its collected sensor data to the user’s social networking account on the webservice. SensCrypt’s Upload protocol extends this protocol, see Section 5.4.

1. Upon receiving a beacon from the tracker, the base establishes a connection with the tracker.
2. **Phase 1:** The base contacts the webservice at the URL `HOME/device/tracker/uploadData` and sends basic client and platform information.
3. **Phase 2:** The webservice sends the tracker id and the opcode for retrieving tracker information (TRQ-REQ).
4. The base contacts the specified tracker, retrieves its information TRQ-INFO (serial number, firmware version, etc.) and sends it to the webservice at `HOME/device/tracker/dumpData/lookupTracker`.
5. **Phase 3:** Given the tracker’s serial number, the webservice retrieves the associated tracker public id (TPI) and user public id (UPI) values. The webservice

sends to the base the TPI/UPI values along with the opcodes for retrieving fitness data from the tracker (READ-TRQ).

6. The base forwards the TPI and UPI values and the opcodes to the tracker, retrieves the fitness data from the tracker (TRQ-DATA) and sends it to the webserver at `HOME/device/tracker/dumpData/dumpData`.
7. **Phase 4:** The webserver sends to the base, opcodes to WRITE updates provided by the user in her Fitbit social network account (device and profile settings, e.g., body and personal information, time zone, etc). The base forwards the WRITE opcode and the updates to the tracker, which overwrites the previous values on its write memory banks.
8. The webserver sends opcodes to ERASE the fitness data from the tracker. The base forwards the ERASE request to the tracker, who then erases the contents of the corresponding read memory banks.
9. The base forwards the response codes from the tracker to the webserver at the address
`HOME/device/tracker/dumpData/clearDataConfigTracker`.
10. The webserver replies to the base with the opcode to CLOSE the tracker.
11. The base requests the tracker to SLEEP for 15 minutes, before sending its next beacon.

5.2.4 Crypto Tools

We use a symmetric key encryption system. We write $E_K(M)$ to denote the encryption of a message M with key K . We also use cryptographic hashes that are pre-image, second pre-image and collision resistant. We use $H(M)$ to denote the hash of message M . We also use hash based message authentication codes [BCK96]:

```
log-20120728203216.txt - Notepad
File Edit Format View Help
07/29 04:37:38 Reset channel.
07/29 04:37:38 Starting session in pairing mode...
07/29 04:37:38 [CTX] CommunicationManager::ResetSession: setting context 00000000
07/29 04:37:38 [CTX] CommunicationManager::StartSession: setting context 00EF6D08
07/29 04:37:38 Processing request...
07/29 04:37:38 Connecting [89]: POST to http://client.fitbit.com:80/device/tracker/uploadData with data: p%5fbc
07/29 04:37:39 Processing action 'http'...
07/29 04:37:39 Sending 8487 bytes of HTML to UL...
07/29 04:37:39 Processing request...
07/29 04:37:39 Waiting for minimum display time to elapse [1000ms]...
07/29 04:37:40 Waiting for form input...
07/29 04:38:00 UI [\\pipe\Fitbit\vallagenah]: F
07/29 04:38:00 Processing action 'form'...
07/29 04:38:00 Received form input: email=networkcrazy13@gmail.com&password=shashi_13&login=%3CSP/
07/29 04:38:00 Connecting [90]: POST to http://client.fitbit.com:80/device/tracker/pairing/signupHandler with da
07/29 04:38:01 Processing action 'http'...
07/29 04:38:01 Sending 26882 bytes of HTML to UL...
07/29 04:38:01 Processing request...
07/29 04:38:01 Waiting for minimum display time to elapse [1000ms]...
```

Figure 5.3: Fitbit service logs: Proof of login credentials sent in cleartext in a HTTP POST request sent from the base to the webserver.

we write $Hmac(K, M)$ to denote the authentication code of message M with key K .

5.3 Security and Privacy Attacks

During the reverse engineering process, we discovered several fundamental vulnerabilities, which we describe here. We then detail the attacks we have deployed to exploit these vulnerabilities, and their results.

5.3.1 Vulnerabilities

Fitbit: cleartext login information. During the initial user login via the Fitbit client software, user passwords are passed to the webserver in cleartext and then stored in log files on the base. Figure 5.3 shows a snippet of captured data, with the cleartext authentication credentials emphasized. Garmin uses encryption only during the login step.

Fitbit and Garmin: cleartext HTTP data processing. For both Fitbit and Garmin, the tracker’s data upload operation uses no encryption or authentication. All the tracker-to-webserver communications take place in cleartext.

Garmin: faulty authentication during Pairing. The authentication in the Pairing procedure of Garmin assumes that the base follows the protocol and has not

been compromised by an attacker. The authentication process is not mutual: the tracker does not authenticate the base.

5.3.2 The FitBite and GarMax Tools

We have built FitBite and GarMax, tools that exploit the above vulnerabilities to attack Fitbit Ultra and Garmin Forerunner. FitBite and GarMax consist of separate modules for (i) discovering and binding to a nearby tracker, (ii) retrieving data from a nearby tracker, (iii) injecting data into a nearby tracker and (iv) injecting data into the social networking account of a tracker owner. We have built FitBite and GarMax over ANT-FS, in order to connect to and issue (ANT-FS) commands to nearby trackers. The attacker needs to run FitBite or GarMax on a base he controls.

The time required to search and bind to a nearby tracker varies significantly, but is normally in the range of 3-20 seconds. On average, the time to query a tracker is 12-15s. More detailed timing information is presented for the attacks presented in the following. We conclude that these attacks can be performed even during brief encounters with victim tracker owners.

5.3.3 Attacks and Results

Tracker Private Data Capture (TPDC). FitBite discovers tracker devices within transmission range and captures their fitness information: Fitbit performs no authentication during tracker data uploads. We exploit Garmin’s assumption of an

Type of data	FitBite	GarMax
Device info	✓	✓
User profile, schedules, goals	✓	✓
Fitness data	✓	✓
(GPS) Location history	✗	✓

Table 5.1: Types of data harvested by FitBite and GarMax from Fitbit and Garmin. Garmin provides GPS tagged fitness information, which GarMax is able to collect.

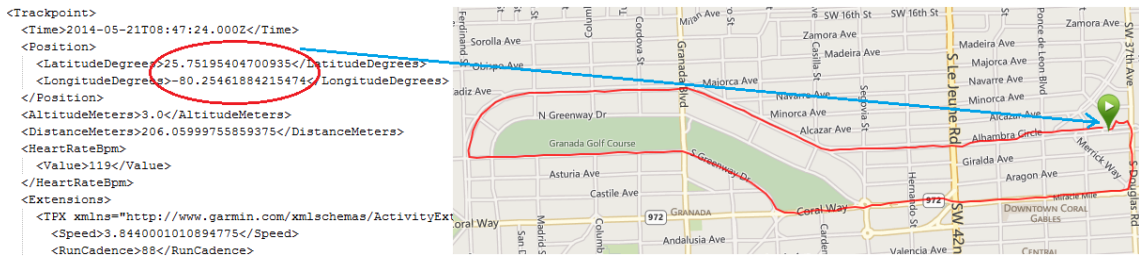


Figure 5.4: TPDC outcome on Garmin: the attacker retrieves the user’s exercise circuit on a map (shown in red on the right side), based on individual fitness data records (shown on the left in XML format). The data record on the left includes both GPS coordinates, heart rate, speed and cadence.

honest base to use GarMax, running on a corrupt base, to capture data from nearby trackers. We show how GarMax binds a “rogue” base agent to Garmin trackers of strangers within a radius of 33ft. GarMax exploits the authentication vulnerability of Garmin’s Pairing procedure (see Section 5.3.1).

During the tracker authentication and passkey retrieval step of the Pairing procedure (see Section 5.2.3), GarMax running on an attacker controlled base, retrieves the TPI of the nearby victim tracker. It then creates a directory with the TPI name and creates an auth file with a random, 8 byte long passkey. GarMax verifies the tracker’s serial number and other ANT parameters, then reads the passkey from the auth file. Instead of running the *passkey* authentication method, GarMax directly downloads fitness information (to be stored in .FIT files) from the tracker. This is possible since the tracker assumes the base has not been corrupted, and thus does not authenticate it.

TPDC can be launched in public spaces, particularly those frequented by fitness users (e.g., parks, sports venues, etc) and takes less than 13s on average. It is particularly damaging as trackers store sensor readings (i) with high frequency (1-9 seconds for Garmin, 1 minute for Fitbit), and (ii) for long intervals: up to 7 days of fitness data history for Fitbit and up to 1000 laps and 100 favorite locations for



Figure 5.5: Outcome of Tracker Injection (TI) attack on Fitbit tracker: The daily step count is unreasonably high (167,116 steps).

Garmin. The data captured contains sensitive user profile information and fitness information. For Garmin this information is tagged with GPS locations. Table 5.1 summarizes the information captured by FitBite and GarMax.

Figure 5.4 shows the reconstructed exercise circuit of a victim, with data we recovered from a TPDC attack on Garmin. The GPS location history can be used to infer the user's home, locations of interest, exercise and travel patterns.

Tracker Injection (TI) Attack. FitBite and GarMax use the reverse engineered knowledge of the communication packet format, opcode instructions and memory banks, to modify and inject fitness data on neighboring trackers. On average, this attack takes less than 18s, for both FitBite and GarMax. Figure 5.5 shows a sample outcome of the TI attack on a victim Fitbit tracker, displaying an unreasonable value for the (daily) number of steps taken by its user.

User Account Injection (UAI) Attack. We used FitBite and GarMax to report fabricated fitness information into our social networking accounts. We have successfully injected unreasonable daily step counts, e.g., 12.58 million in Fitbit, see Figure 5.6. Fitbit did not report any inconsistency, especially as the corresponding distance we reported was 0.02 miles! The UAI attack takes only 6s on average.

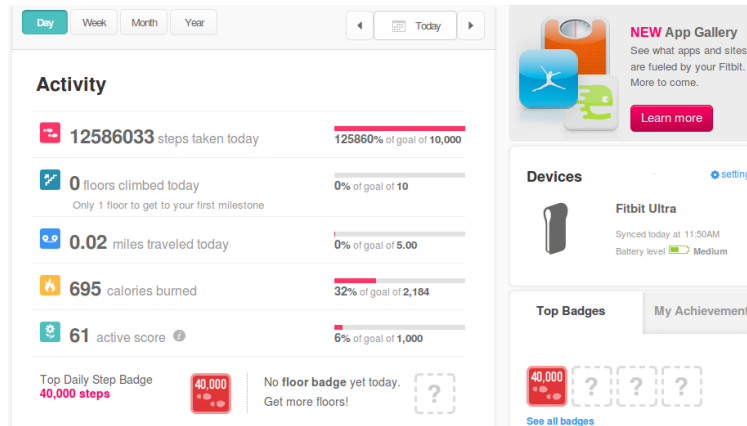


Figure 5.6: Snapshot of Fitbit user account data injection attack. In addition to earning undeserved badges (e.g., the “Top Daily Step”), it enables insiders to accumulate points and receive financial rewards through sites like Earndit [Ear].

Similarly, GarMax fabricates an activity file embedding the attacker provided fitness data in FIT/TCX [TCX] format. The simplest approach is to copy an existing activity file of the same or another user (made publicly available in the Garmin Connect website) and modify device and user specific information. We have used GarMax to successfully inject “running” activities of 1000 miles each, the largest permissible value, while keeping the other parameters intact.

Free Badges and Financial Rewards. By successful injection of large values in their social networking accounts, FitBite and GarMax enable insiders to achieve special milestones and acquire merit badges, without doing the required work. Figure 5.6 shows how in Fitbit, the injected value of 12.58 million steps, being greater than 40,000, enables the account owner to acquire a “Top Daily Step” badge. Furthermore, by injecting fraudulent fitness information into Earndit [Ear], an associated site, we were able to accumulate undeserved rewards, including 200 Earndit points, redeemable for a \$20 gift card.

Battery Drain Attack. FitBite allows the attacker to continuously query trackers in her vicinity, thus drain their batteries at a faster rate. To understand the efficiency

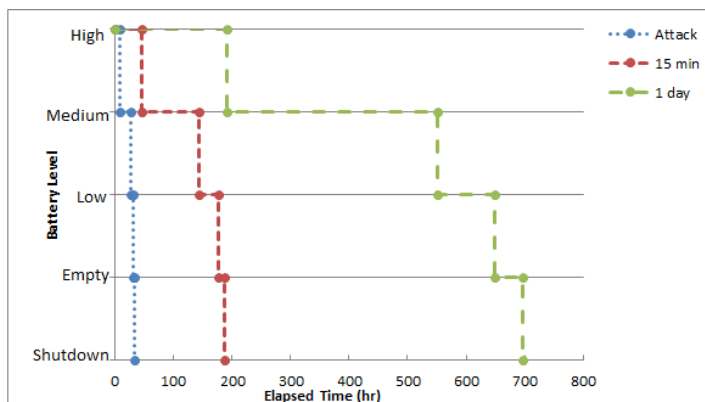


Figure 5.7: Battery drain for three operation modes. The attack mode drains the battery around 21 times faster than the 1 day upload mode and 5.63 times faster than the 15 mins upload mode.

of this attack, we have experimented with 3 operation modes. First, the *daily upload* mode, where the tracker syncs with the USB base and the Fitbit account once per day. Second, the *15 mins upload* mode, where the tracker is kept within 15 ft. from the base, thus allowing it to be queried once every 15 minutes. Finally, the *attack* mode, where FitBite’s TM module continuously (an average of 4 times a minute) queries the victim tracker. To avoid detection, the BM module uploads tracker data into the webserver only once every 15 minutes. Figure 5.7 shows our battery experiment results for the three modes: FitBite drains the tracker battery around 21 times faster than the 1 day upload mode and 5.63 times faster than the 15 mins upload mode.

In the daily upload mode, the battery lasted for 29 days. In the 15 mins upload mode, the battery lasted for 186.38 hours (7 days and 18 hours). In the attack mode, the battery lasted for a total of 32.71 hours. While this attack is not fast enough to impact trackers targeted by casual attackers, it shows that FitBite drains the tracker battery around 21 times faster than the 1 day upload mode and 5.63 times faster than the 15 mins upload mode.

Denial of Service. FitBite’s injection attack can be used to prevent Fitbit users from correctly updating their real-time statistics. The storage capacity of the Garmin tracker is limited to 1000 laps. Thus, an attacker able to injects a number of fake laps exceeding the 1000 limit, can prevent the tracker from recording the user’s valid data. A Fitbit tracker can display up to 6 digit values. When the injected value exceeds 6 digits, the least significant digits can not be displayed on the tracker. This prevents the user from keeping track of her daily performance evolution. In addition, for both Fitbit and Garmin, the attacker can render part of the recorded data useless, by injecting incorrect user profile information. For instance, by modifying user profile information (e.g., height, weight, see Section 5.2.1), the attacker corrupts information built based on it, e.g., “calories burnt”.

5.4 A Protocol for Lightweight Security

5.4.1 Solution Requirements

We aim to develop a solution for low power fitness trackers that satisfies the following requirements:

1. **Security.** Defend against the attacks described in Section 5.2.2.
2. **Minimal tracker overhead.** Minimize the computation and storage overheads imposed on the resource constrained trackers.
3. **Flexible upload.** Allow trackers to securely upload sensor information through multiple bases.
4. **User friendly.** Minimize user interaction.
5. **Level tracker memory wear.** Extend memory lifetime by leveling the wear of its blocks.

5.4.2 Public Key Cryptography: A No Go

We propose first FitCrypt, a solution to explore the feasibility of public key cryptosystems to efficiently secure the storage and communications of trackers. In FitCrypt, each tracker stores a public key. The corresponding private key is only known by the webserver. Each sensor data record is encrypted with the public key before being stored on the tracker. RSA with a 2048 bit key imposes a 4-fold storage overhead on Fitbit (each record of 64B is converted into a 256B record) and a 3.2-fold overhead on Garmin. We also consider ECIES (Elliptic Curve Integrated Encryption Scheme), an elliptic curve crypto (ECC) solution that uses a 224 bit key size, the security equivalent of RSA with 2048 bit modulus. ECIES imposes a storage overhead of $224 + 3r$ bits, where $r = 112$ is the size of a security parameter. Thus, the storage overhead is 165% for Fitbit and 150% for Garmin).

When run on an Arduino Uno board, FitCrypt-RSA takes 2.3s and FitCrypt-ECC takes 2.5s to encode a single sensor record (see Table 5.5, Section 5.6). Garmin records sensor data with a frequency as high as one write per second. FitCrypt imposes a 250% overhead on the sensor recording task (of 2.5s every 1s interval), thus does not satisfy the second requirement of Section 5.4.1. To address this issue, in the following we introduce SensCrypt, a lightweight and secure solution for wearable trackers.

5.4.3 SensCrypt

We introduce SensCrypt, a lightweight protocol for providing secure data storage and communication in fitness centric social sensor networks.

Protocol overview. Let U denote a user, T denote her tracker, B a base and W the webserver. T 's memory is divided into records, each storing one snapshot of sensor data. The memory is organized using a circular buffer structure, to ensure

Notation	Definition
U, T, B, W	user, tracker, base and webserver
id_U, id_T, id_B	unique identifiers of U, T and B
$dirty$	pointer to first written record
$clean$	pointer to first available record
$start, end$	pointers to memory bounds
K_W	symmetric key maintained by W for T
K_T	symmetric key shared by W and T
ctr	counter shared by W and T
Map	data base of W for users and trackers
mem	memory of a tracker

Table 5.2: Symbol definitions.

an even wear. T shares a symmetric key K_T with W . W also maintains a unique secret key K_W for each tracker T .

To prevent Inject attacks, all communications between T and W are authenticated with K_T . To prevent Inspect, Capture and JTAG-R attacks, we encode each tracker record using two pseudo-random numbers (PRNs). One PRN is generated by W using K_W and written on T during data sync protocols. The other PRN is generated by T using K_T at the time when the record is written on its memory. Both PRNs can later be reconstructed by W . This approach significantly increases the complexity of an attack: the attacker needs to capture the encoded data and both PRNs to recover the cleartext data.

5.4.4 The SensCrypt Protocol

Let id_U, id_B , and id_T denote the public unique identities of U, B , and T . U has an account with W . W manages a database Map that has an entry for each user and tracker pair: $Map[id_U, id_T] = [id_U, id_T, K_T, K_W, ctr]$. Each tracker is factory initialized with a symmetric key K_T and a counter ctr initialized to 1. K_T and ctr are also stored in $Map[id_U, id_T]$. K_W is a per-tracker symmetric key, kept secret by W . Table 5.2 summarizes these symbols for easy access.

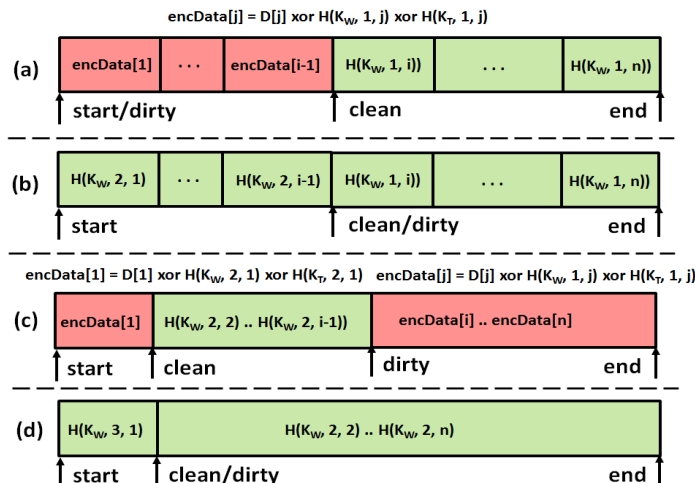


Figure 5.8: Example SensCrypt tracker memory (*mem*). Light green denotes “clean”, unwritten areas. Red denotes areas that encode tracker sensor data. (a) After $(i-1)$ records have been written. The ctr is 1. (b) After Upload occurs at the state in (a). The ctr becomes 2, to enable the creation of fresh PRNs, overwritten on the former red area. (c) After $n-i+2$ more records have been written from state (b), leading to the clean pointer cycling over from the start of the memory. (d) After Upload occurs at the state in (c).

SensCrypt consists of 2 procedures, *RecordData* and *Upload*. *RecordData* is invoked by T to record new sensor data; *Upload* allows it to sync its data with W . We now describe the organization of the tracker memory.

Tracker Memory Organization. Let mem denote the memory of T . mem is divided in “records” of fixed length (e.g., 64 bytes for Fitbit, 80 bytes for Garmin). Each record stores one report from the tracker’s sensors (see Section 5.2.1). We organize time into fixed length “epochs” (e.g., 2s long for Fitbit, 1-9s long for Garmin). *RecordData* records sensor data once per epoch. mem is organized using a circular buffer. The *dirty* pointer is to the location of the first written record, and the *clean* pointer is to the location of the first record available for writing. When reaching the end of mem , both records “circle” over to the *start* pointer. Figure 5.8 illustrates the SensCrypt tracker storage organization, after the execution of various *RecordData* and *Upload* procedures. Algorithm 3 shows the pseudo-code of the procedures.

During *Upload*, each previously written tracker record is reset by W to store a pseudo-random value (line 18 and lines 21-29 of Algorithm 3). That is, the i -th record of the tracker’s memory is set to hold $E_{K_W}(ctr, i)$, where K_W is the secret key W stores for T . The index i ensures that each record contains a different value. ctr counts the number of times mem has been completely overwritten; it ensures that a memory record is overwritten with a different encrypted value.

The RecordData Procedure. Commit newly recorded sensor data D to mem , in the next available record, pointed to by $clean$. T generates a new pseudo-random value, $E_{K_T}(ctr, clean)$, and xors it into place with $mem[clean] = E_{K_W}(ctr, clean)$ and D (see line 10 of algorithm 3):

$$mem[clean] = D \oplus E_{K_T}(ctr, clean) \oplus E_{K_W}(ctr, clean).$$

The $clean$ pointer is then incremented (line 11). When reaching the *end* of mem , $clean$ circles back to *start*(lines 12,13). We call “red” the written records and “green” the records available for write. *dirty* and *clean* enable us to reduce the communication overhead of *Upload* (see next): instead of sending the entire mem , T sends to W only the red records.

The Upload Procedure. We present the SensCrypt *Upload* as an extension of the corresponding Fitbit protocol illustrated in Figure 5.2. In the following, each message M sent between T and W is accompanied by an authentication value $Hmac(K_T, M)$, where Hmac is a hash based message authentication code [BCK96]. The receiver of the message uses K_T to verify the authenticity of the sender and of the message. For simplicity of exposition, in the following we omit the Hmac value.

Upload extends steps 6b and 7 of the Fitbit *Upload*. Specifically, when T receives the READ-TRQ command (step 6a), it compares the *dirty* and *clean* pointers. If

$dirty < clean$ (see Figure 5.8(a)), T sends to W , through B ,

$$T \rightarrow B \rightarrow W : \text{TRQ} - \text{DATA}, \text{id}_T, \text{mem}[dirty..clean],$$

where $\text{mem}[dirty..clean]$ denotes T 's red memory area. For each record i between $dirty$ and $clean$, W uses keys K_T and K_W and the current value of ctr to recover the sensor data: $D[i] = \text{mem}[i] \oplus E_{K_T}(ctr, i) \oplus E_{K_W}(ctr, i)$ (see lines 21-23 and line 16). Then, in step 7 of Upload (see Figure 5.2), W sends to T :

$$W \rightarrow B \rightarrow T : \text{WRITE}, \text{id}_T, E_{K_T}(ctr + 1, E_{K_W}(ctr + 1, i)),$$

$\forall i = dirty..clean$. T uses K_T to decrypt each $E_{K_T}(ctr + 1, E_{K_W}(ctr + 1, i))$ value. If the first field of the result equals $ctr + 1$, T overwrites $\text{mem}[dirty + i]$ with $E_{K_W}(ctr + 1, i)$ (see line 18), then sets $dirty = clean$ (line 30). Thus, $\text{mem}[dirty..clean]$ becomes green. The case where $clean < dirty$, occurring when $clean$ circles over, past the memory end, is handled similarly, see lines 24-29 of Algorithm 3 and Figure 5.8(c) and (d). We eliminate the ERASE communication (steps 8 and 9 in Figure 5.2) from the Fitbit protocol.

5.5 Analysis

5.5.1 SensCrypt Advantages

SensCrypt ensures an even wear of tracker memory: the most overwritten memory record has at most 2 overwrites more than the least overwritten record. To see why this is the case, consider that once written, a record is not overwritten until a next *Upload* takes place. The circular buffer organization of the memory ensures that all the memory records of the tracker are overwritten, not just the ones at the start of the memory. Using the example illustrated in Figure 5.8(d), notice that the first record, has been overwritten twice since the subsequent green blocks: once with $\text{encData}[1]$, see Figure 5.8(c), and once with the new $E_{K_W}(3, 1)$ received from W .

Capabilities	SensCrypt	FitCrypt
Inspect	TPDC, TI, UAI	TPDC, TI, UAI
Inject	TPDC, TI, UAI	TPDC, TI, UAI
Capture	TPDC, TI, UAI	TPDC, TI, UAI
JTAG-R	TPDC, TI, UAI	TPDC, TI, UAI
JTAG-RW	TPDC	TPDC
JTAG-R + Inspect	TI, UAI	TPDC, TI, UAI
JTAG-R + Inject	TI	TPDC, TI
Double JTAG-R	TI, UAI	TPDC, TI, UAI

Table 5.3: Comparison of defenses provided by SensCrypt and FitCrypt against the types of attacks described in Section 5.3.3 when the adversary has a combination of the capabilities described in Section 5.2.2. Each element in the table describes which attacks are thwarted by the corresponding solution.

By preventing excessive overwriting of records at the beginning of the memory, SensCrypt extends the life of trackers. This is particularly important for flash memories, that have a limited number of P/E (program/erase) cycles.

SensCrypt is user friendly, as the user is not involved in *Upload* and *RecordData* procedures. The SensCrypt base is thin, required to only setup standard secure SSL connections to W , and forward traffic between T and W . SensCrypt imposes no storage overhead on trackers: sensor data is xor-ed in-place in *mem*.

5.5.2 Security Discussion

Consider the life cycle of record i , R_i , on T . After the execution of the first *Upload*, R_i is initialized with $E_{K_W}(ctr, i)$. When R_i is overwritten with sensor data, it contains $encData[i] = D[i] \oplus E_{K_T}(ctr, i) \oplus E_{K_W}(ctr, i)$. Subsequently, R_i is not touched until an Upload takes place. During Upload, the (encoded) content of $mem[i]$ is sent to W , who subsequently overwrites R_i with a new value: $E_{K_W}(ctr + 1, i)$.

The base does not contribute to the messages it forwards between T and W . Hence, the base does not need to be authenticated. The use of the $ctr + 1$ value in communications through the base ensures message freshness.

Without $E_{K_T}(ctr, i)$, an Inspect adversary capturing communications between T and W cannot recover $mem[i]$. The use of HMACs with the key K_T to authenticate communications between T and W prevents Inject attacks: an attacker that modifies existing messages or injects new messages cannot create valid HMAC values.

An attacker that launches a Capture attack against a victim tracker or base, cannot recover information from them and thus has no advantage over general Inspect and Inject attacks. An adversary that captures a tracker T and launches a JTAG-R attack can either read $E_{K_W}(ctr, i)$ or $D[i] \oplus E_{K_W}(ctr, i) \oplus E_{K_T}(ctr, i)$, but not both. The use of the $E_{K_T}(ctr, i)$ value prevents an attacker from recovering $D[i]$. A JTAG-R attack against a captured, trusted base of tracker T offers no advantage over Inspect and Inject attacks: in SensCrypt, the base only forwards traffic between T and W . Similar to JTAG-R, a JTAG-RW attack against a captured tracker cannot decode previously encoded sensor data; it can however encode fraudulent data on the tracker (TI attack) and thus also inject data into W (UAI attack).

An adversary able to perform Inspect, Capture and JTAG-R attacks can gain access to $E_{K_W}(ctr, i)$ when sent by W , then use JTAG-R to read T 's K_T , compute $E_{K_T}(ctr, i)$ and learn $D[i]$ (TPDC attack). We note the complexity of this attack. If able to further implement Inject attacks, the adversary can also succeed in a UAI attack.

Furthermore, SensCrypt is vulnerable to an adversary able to capture T twice, at times t_1 and t_2 . At time t_1 the adversary uses JTAG-R to read $E_{K_W}(ctr, i)$. At time t_2 , assuming T has already written record i , the adversary uses JTAG-R to read $mem[i]$ and K_T and recover $D[i]$. This *double JTAG-R attack* is significantly

more complex than a single JTAG-R attack. In addition, this attack is further complicated by time constraints: At t_1 , record i has not yet been written, and at t_2 it has been written *but* an Upload has not yet been executed. An Upload procedure before t_2 would overwrite record i with $E_{K_W}(ctr + 1, i)$, effectively thwarting this attack.

FitCrypt is resilient to TPDC attacks launched by adversaries capable of performing JTAG-R and Inspect, Inject and double JTAG-R attacks: T 's records encrypted with the public key can only be decrypted by W . Table 5.3 summarizes the comparison of SensCrypt and FitLock defenses. While providing more defenses (i.e., against TPDC for several attacker capabilities), FitLock is not a viable solution on most of the available trackers (see Section 5.6).

5.5.3 Applications

SensCrypt can be applied to a range of sensor based platforms, where resource constrained sensors are unable to directly sync their data with a central webserver and need to use an Internet connected base. This includes a large number of popular fitness and home monitoring solutions. Table 5.4 summarizes several such platforms, including the communication and storage capabilities of their sensors.

SensCrypt can also be used in applications where the sensors need to be immobile, while being able to operate without network connectivity. Examples include health, infrastructure, traffic, building and campus monitoring solutions. The bases through which the sensors sync with the webserver are mobile, e.g., smartphones of workers, who may become proximal to the sensors with the intention of data collection or as a byproduct of routine operations.

SensCrypt can also secure the data and communications of platforms for social psychological studies. One such example is SociableSense [RMMR11], a smartphone

Platform	Type of data	Comm	Coverage	Memory
Fitbit [Fit]	user profile, fitness, sleep data	ANT+, BT	5-50m	96 KB RAM, 112 KB flash
Garmin FR610 [For]	fitness data, heart rate, location	ANT+	10-20m	1 MB
Nike+ [Nik]	profile, fitness data	BT	50m	Flash 256KB, RAM 32 KB
Jawbone Up [Jaw]	fitness, sleep data	BT	50m	128KB Flash, 8KB RAM
Motorola MotoActv [Mota]	fitness data, user profile	ANT+, BT, Wi-Fi	35m	16 GB
Basis B1 [Bas]	fitness, sleep data, heart rate	BT	50m	7 days of data
Mother [Motb]	motion, fitness, proximity	915-MHz	30m	32 KB RAM
Nest [Nes]	utility data	Wi-Fi	35m	512Mb DRAM, 2 Gb flash
Belkin WeMo [WeM]	home electronics	Wi-Fi	35m	RAM 32 MB, Flash 16 MB

Table 5.4: SensCrypt applicability: fitness trackers, home monitoring solutions.

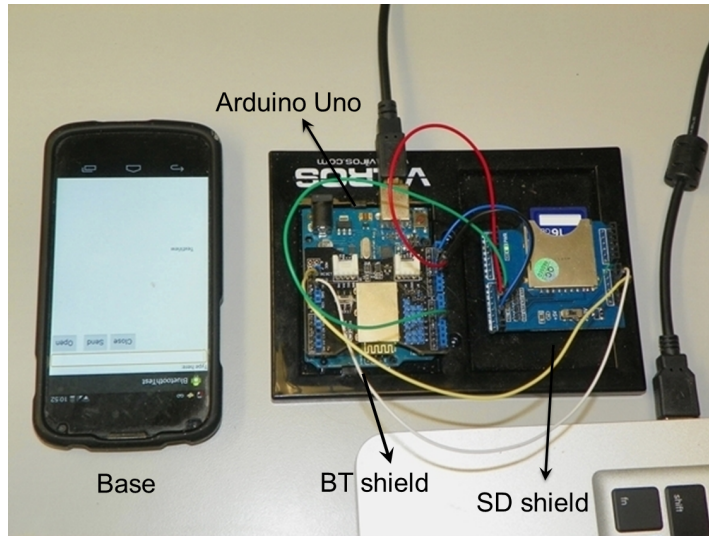


Figure 5.9: Testbed for SensCrypt. Sens.io is the Arduino Uno device equipped with Bluetooth shield and SD card is the tracker. Nexus 4 is the base.

solution that captures sensitive user behaviors (including co-location), processes the information on a remote server, and provides measures of user sociability.

5.6 Evaluation

We used Sens.io for the tracker, Android Nexus 4 with 1.512 GHz CPU for the base, and a 2.4GHz Intel Core i5 Dell laptop with 4GB of RAM for the webserver. We used Bluetooth for tracker to base communications and Wi-Fi for the connectivity between the base and the webserver. Figure 5.9 illustrates our testbed.

5.6.1 Sens.io: The Platform

We have built Sens.io, a prototype tracker, from off-the-shelves components. It consists of an Arduino Uno Rev3 [ardb] and external Bluetooth (Seeduino V3.0) and SanDisk card shields. The Arduino platform is a good model of resource constrained trackers: its ATmega328 micro-controller has a 16MHz clock, 32 KB Flash memory, 2 KB SRAM and 1KB EEPROM. The Bluetooth card has a default baud rate of

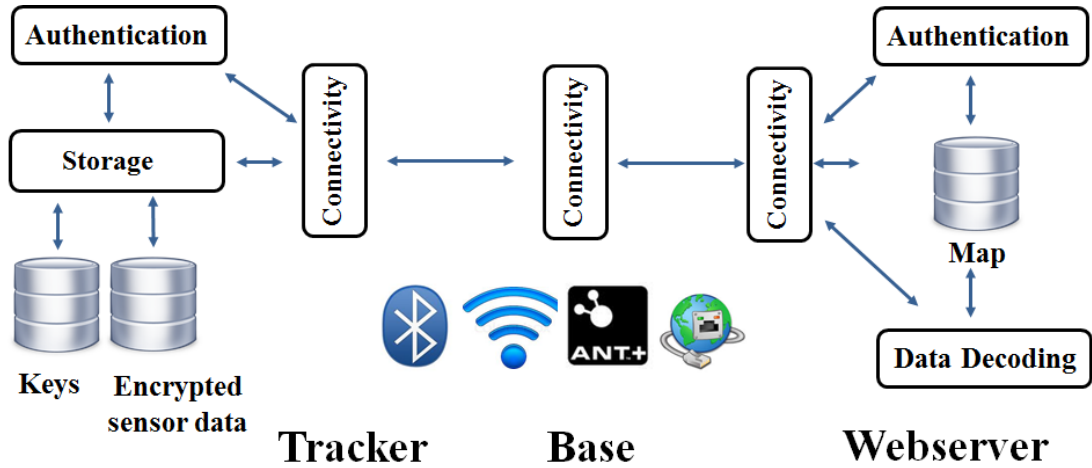


Figure 5.10: SensCrypt architecture. The tracker relies on locally stored key K_T to authenticate webservice messages and encode sensor data. The webservice manages the *Map* structure, to authenticate and decrypt tracker reports.

38,400 and communication range up to 10m. Since the Arduino has 2 KB SRAM, it can only rely on 1822 bytes to buffer data for transmissions. The SD card (FAT 16) can be accessed at the granularity of 512 byte blocks.

The cost of Sens.io is \$52 (\$25 Arduino card, \$20 Bluetooth shield, \$2.5 SD Card shield, \$4 SD card, see Figure 5.9), a fraction of Fitbit's (\$99) and Garmin's (\$299) trackers.

SensCrypt. We have implemented a general, end-to-end SensCrypt architecture, as illustrated in Figure 5.10. We have implemented the tracker both in Arduino's programming language (a Wiring implementation [Arda]), and, for generality, in Android. The base component (written exclusively in Android) is a simple communication relay. We implemented the webservice using Apache Tomcat 7.0.52 and Apache Axis2 Web services engine. We used the MongoDB 2.4.9 database to store the *Map* structure. We implemented a Bluetooth [SIG01] serial communication protocol between the tracker and the base.

Platform	SensCrypt	FitCr-RSA	FitCr-ECC
Fitbit	6.02	2300	2520
Garmin	6.06	2300	2520

Table 5.5: RecordData: computation overhead in ms. FitCrypt-RSA 2048 bit is not viable on Arduino (2.3s). FitCrypt-ECC 224 bit (equivalent of RSA 2048 bit) is even less efficient. SensCrypt is 2-3 orders of magnitude more efficient.

The testbed. We used Sens.io for the tracker, an Android Nexus 4 with 1.512 GHz CPU for the base, and a 2.4GHz Intel Core i5 Dell laptop with 4GB of RAM for the webserver. We used Bluetooth for tracker to base communications and Wi-Fi for the connectivity between the base and the webserver. Figure 5.9 illustrates our testbed.

In the following, we report evaluation results, as averages taken over at least 10 independent protocol runs.

5.6.2 Tracker: RecordData Overhead

We have investigated the overhead of the *RecordData* procedure on Sens.io. Table 5.5 compares the performance of SensCrypt and FitCrypt, with times shown in milliseconds. We have explored two versions of FitCrypt, using RSA and ECC. FitCrypt-RSA with a 1024 bit modulus takes more than 500ms, but is currently obsolete. FitCrypt-RSA with a 2048 bit modulus hangs on Sens.io due to its low (2KB) RAM. The value shown in Table 5.5 is from [GPW⁺04], where a similar platform was used. FitCrypt-ECC uses ECIES, an elliptic curve cryptography solution, with a 224 bit key size, the security equivalent of RSA with 2048 bit modulus. FitCrypt-RSA 2048 and FitCrypt-ECC are not viable alternatives, imposing an overhead of 230% for 1 per sec. RecordData frequency. SensCrypt imposes however an overhead of less than 1% (6ms for each 1s interval between RecordData runs).

5.6.3 Webserver: Storage Overhead

The webserver maintains a data structure, *Map*, with a record for each user and tracker pair. The entry consists of user, tracker and bases ids (8 byte long each), a salt (16B), password hash (28B), 2 symmetric keys (32B each) and a counter (1B). Assuming a single base in the *Bases* list, a *Map* entry stores 133 bytes. For a 1 million user base, the webserver needs to store a *Map* structure of 127MB. The average time to retrieve a record from a 1 million user *Map* is 158ms.

5.6.4 Upload: End-to-end Overhead

We consider a “Fitbit” scenario where the Upload procedure runs once every 15 minutes when in the vicinity of a base. Assuming a *RecordData* frequency of once every 2s (usual in Garmin), and a record size of 64B, *SensCrypt* uploads and overwrites 71 blocks of 512B each. The tracker side of the *SensCrypt* Upload procedure takes 502ms, dominated by the cost to read and write 71 blocks of data from/to the SD card. A single core of the Dell laptop can support 5 Uploads per second. The server cost is dominated by the 158ms cost of retrieving a record from a 1 million entry *Map*. The Upload/s rate of the webserver can be improved by caching the least recently accessed or most popular records of *Map*. Even though transferring over Bluetooth, the communication cost of *SensCrypt*’s Upload is 153ms. This is due to the low RAM available on Arduino for buffering packets (2KB).

SensCrypt’s total Upload time of 845ms is 400ms less than *FitCrypt*’s, assuming Fitbit’s memory size. We note however that Fitbit records data only once per minute, a rate at which *SensCrypt* would perform significantly faster. *SensCrypt* is 13 times faster (by more than 11s) than *FitCrypt* when considering Garmin’s memory (2000 blocks of 512B). This gain is due to *SensCrypt*’s optimization of only uploading the red, written blocks, instead of the entire memory.

Solutions	T	W	Comm
SensCrypt	502.13	190.4	153
FitCrypt (Fitbit)	904.56	177.36	162
FitCrypt (Garmin)	9366	322	1686

Table 5.6: Upload: comparison of tracker, webserver and communication delays (shown in ms) of SensCrypt and FitCrypt. FitCrypt (RSA or ECC) is shown both for the Fitbit (96KB) and Garmin (1MB) memory size. The delay of SensCrypt is independent of *mem* size, and significantly shorter.

Furthermore, even on the communication restricted Sens.io, SensCrypt reduces the upload operation of the *real* Fitbit equipment (1481ms on average) by 43%.

5.6.5 Battery Impact

To evaluate the impact of SensCrypt on the battery lifetime, we powered the Sens.io device using a 9V alkaline battery [dur]. In a first experiment, we evaluated the ability of SensCrypt to mitigate the effects of the battery drain attack. For this, we used the Bluetooth enabled Sens.io device to establish a connection with an Android app running on a Nexus 4 base. We investigated and compared two scenarios. In the first scenario, the Bluetooth enabled Sens.io runs the Fitbit protocol to process and respond to requests received every 15s. In the second scenario, the Sens.io device runs SensCrypt to process the same requests. Each scenario is performed using a fresh 9V battery.

When running Fitbit, the Sens.io device runs out of battery after 484 minutes. When running SensCrypt, the Sens.io device lasts for a total of 821 minutes. Thus, SensCrypt extends the battery lifetime of Sens.io under the battery depletion attack by 69%.

In a second experiment, we compared the impact of the periodic SensCrypt, FitCrypt-RSA-256 and Fitbit sensor data recording operations on the Sens.io battery lifetime. In the experiment, we considered a 2s interval between consecutive

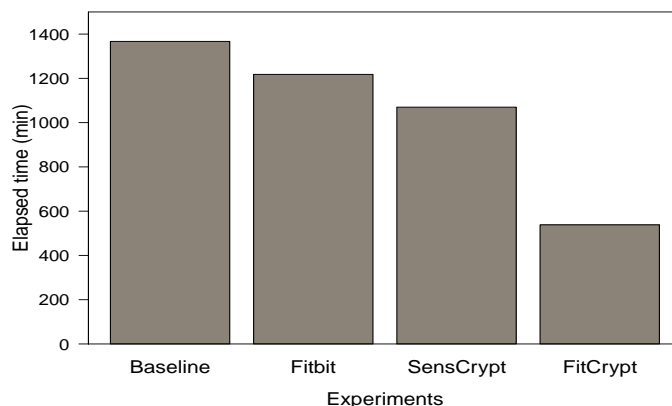


Figure 5.11: Battery lifetime for 9V cell powered Sens.io device in four scenarios: Baseline, Fitbit, SensCrypt and FitCrypt-RSA-256. The last three scenarios record sensor data every 2s. The Baseline scenario measures the battery lifetime of Arduino device with no functionality. SensCrypt reduces 13% of the battery lifetime over Fitbit’s operation. Even a vulnerable FitCrypt-RSA-256 reduces the battery lifetime to half of SensCrypt.

sensor recording operations. We have tested several RSA key sizes (2048 to 256 bit long). An (insecure) RSA key size of 256 bits was the largest value that did not hang on an Arduino board after only a few encryptions. We have also run a baseline experiment, measuring the battery lifetime of an Arduino board that is not recording any sensor data.

Figure 5.11 shows our results. In the Baseline scenario, the battery lasted 56 hrs and 23 mins. When running Fitbit’s sensor data record operation, the battery lasted 50 hrs and 18 mins. When running SensCrypt’s *RecordData* operation, the battery lasted 43 hrs and 38 mins. Thus, Fitbit’s sensor recording operation shortens the battery by 10% over the baseline. SensCrypt’s *RecordData* reduces the battery lifetime by 13% of the Fitbit battery lifetime. Finally, when running FitCrypt-RSA-256, the battery lasted only 22 hrs and 10 mins. Even with a vulnerable key size, FitCrypt reduces the battery lifetime by 49% of the SensCrypt lifetime. This

confirms the unsuitability of public key cryptosystems to secure resource constrained fitness trackers.

5.7 Limitations

While SensCrypts defenses may not be immediately adopted by existing products, this paper provides a foundation upon which to create, implement and test new defensive mechanisms for future tracker designs.

5.8 Summary

In this chapter, We presented SensCrypt, a secure and efficient solution for storing and communicating tracker sensor data. Firstly we described the reverse-engineering process of the Fitbit Ultra and Garmin Forerunner communication protocol. We then identified and exploited several vulnerabilities in the design of Fitbit and Garmin, to launch inspection and injection attacks. We show that SensCrypt protects even against invasive attackers, capable of reading the memory of captured trackers. To demonstrate the usefulness of the proposed solution, several sets of experiments are conducted to demonstrate its effectiveness and efficiency. Finally, we built a prototype tracker, Sens.io based on our proposed solution which is secure and cost-effective.

Algorithm 3 Tracker memory management pseudocode. Instructions preceded by W : are executed at the webserver, those preceded by T : are executed at the tracker. $W \rightarrow T$: I denotes an instruction I issued at W and executed at T . The entire RecordData is executed at T . Figure 5.8 illustrates the pseudocode.

```

1. Object implementation Memory;
2. T: mem : record[];           #tracker memory
3. T: dirty : int; #pointer to used area
4. T: clean : int; #pointer to unused area
5. T: start, end : int; #memory bounds
6. W: Kw : byte[]; #key for T
7. W, T: KT : byte[]; #key shared by T, W
8. W, T: ctr : int; #counter shared by T, W

9. Operation int T: RecordData(D : sensor data)
10.   mem[clean]  $\oplus$  = D  $\oplus$  EKT(ctr, clean);
11.   clean = clean + 1;
12.   if (clean == end) then;
13.     clean = start; fi
14. end
15. Operation void ProcessRecord(ind : int, c : int)
16.   W: D = mem[ind]  $\oplus$  EKw(c, ind)  $\oplus$  EKT(c, ind);
17.   W: process(D);
18.   W  $\rightarrow$  T: mem[ind] = EKw(c + 1, ind);
19. end
20. Operation void Upload()
21.   if (dirty < clean) do
22.     for (i = dirty; i < clean; i++) do
23.       ProcessRecord(i, ctr); od
24.   else if (clean < dirty) do
25.     for (i = dirty; i  $\leq$  end; i++) do
26.       ProcessRecord(i, ctr); od
27.     for (i = start; i < clean; i++) do
28.       ProcessRecord(i, ctr + 1); od
29.     W, T: ctr = ctr + 1; fi
30.   T: dirty = clean;
31. end

```

CHAPTER 6

TOWARD PRESERVING PRIVACY AND FUNCTIONALITY IN GEOSOCIAL NETWORKS

In this chapter, we would mainly focus on preserving privacy and functionality in geosocial networks and exploit our approaches for computing safety snapshots of co-located mobile devices as well as geosocial network users. Part of the content in this section has been published during my Ph.D study, including the problem formulation and the proposed solutions and its evaluation results.

The outline of this chapter is as follows: The motivation and challenges of this topic will be presented in Section 6.1. The system and adversary model will be described and the problem will be defined in Section 6.2. In Section 6.3, the collected dataset will be described. Then in Section 6.4, the deceptive behavior detection mechanisms will be introduced and evaluated. In Section 6.5, experimental results will be presented to evaluate the performance of our proposed methods. Some limitations of this work will be discussed in Section 6.6. Finally, a short chapter summary about this problem and the proposed solutions will be provided in Section 6.7.

6.1 Motivation and Challenges

Online social networks have become a significant source of personal information. Geosocial networks (GSNs) such as Yelp [Yela] and Foursquare [fou] further collect fine grained location information, through *check-ins* performed by users at visited venues. Providing personal information exposes however users to significant risks, as social networks have been shown to leak [KW10] and even sell [SF] user data to third parties. There exists therefore a conflict. Without privacy people may be reluctant to use geosocial networks; without user information the provider and venues cannot support applications and have no incentive to participate. In this

paper, we take first steps toward addressing this conflict. Our approach is based on the concept of *location centric profiles* (LCPs). LCPs are statistics built from the profiles of (i) users that have visited a certain location or (ii) a set of co-located users. We then envision a system where users are seamlessly made aware of their safety in a personalized manner, through quotidian experiences such as navigation, mobile authentication, choosing a restaurant or finding a place to live. We propose to achieve this vision by introducing a framework for defining public safety. Intuitively, public safety aims to answer the question “Will location L present any danger for user A when she visits L at a future time T ”?

An important challenge to achieving this vision is the need to properly understand and define safety. While safety is naturally location dependent, it is also inherently volatile. It not only exhibits temporal patterns (e.g., function of the season, day of week or time of day) but also depends on the current *context* (e.g., people present, their profile and behavior). Furthermore, as suggested by the above question, public safety has a personal dimension: users of different backgrounds are likely to be impacted differently by the same location/time context.

Previous attempts to make people safety-aware include the use of social media to distribute information about unreported crimes [FAdO⁺10], or web based applications for visualizing unsafe areas [Cri, Gua]. The main drawbacks of these solutions stem from the difficulty of modeling safety and of integrating it in quotidian user experiences.

Instead, in this paper we investigate the combination of space and time indexed crime datasets, with mobile technologies and online social networks to provide personalized and context aware safety recommendations for mobile and social network users. To achieve this, we first define location centric, static crime and safety metrics, based on recorded crime events. Given observed crime periodicities, we show

that timeseries forecasting tools are able to predict future crime and safety index values of locations, based on past crime events.

We then introduce Profil_R , a framework for constructing *location centric profiles* (LCPs), aggregates built over the profiles of users that have visited discrete locations (i.e., venues). Profil_R endows users with strong privacy guarantees and providers with correctness assurances. We then introduce iSafe, a distributed algorithm that addresses privacy concerns raised by the use of trajectory traces and associated crime and safety index values. iSafe takes advantage of the wireless capabilities of mobile devices to compute real-time snapshots of the safety profiles of close-by users in a privacy preserving manner.

6.2 Model and Background

We consider a core functionality that is supported by the most influential geosocial network (GSN) providers, Yelp [Yela] and Foursquare [fou]. This functionality is simple and general enough to be applicable to most other GSNs (e.g., Facebook Places, Google Latitude). In this model, a provider S hosts the system, along with information about registered venues, and serving a number of users. To use the provider’s services, a client application, the “client”, needs to be downloaded and installed. Users register and receive initial service credentials, including a unique user id.

The provider supports a set of businesses or venues, with an associated geographic location (e.g., restaurants, yoga classes, towing companies, etc). Users are encouraged to report their location, through *check-ins* at venues where they are present. During a check-in operation, performed upon an explicit user action, the user’s device retrieves its GPS coordinates, reports them to the server, who then

returns a list of nearby venues. The device displays the venues and the user needs to choose one as her current check-in location.

Participating venue owners need to install inexpensive equipment (e.g., a \$25 Raspberry PI [Ras], a BeagleBoard [Col09] or any Android smartphone). This equipment can be installed and used for other purposes as well, including detecting fake user check-ins [CP12] preventing fake badges and incorrect rewards, and validating social network (e.g., Yelp [Yela]) reviews. Venue deployed equipment provides a necessary ingredient: ground truth information from remote locations.

6.2.1 Location Centric Profiles (LCP)

Each user has a profile $P_U = \{p_{U_1}, p_{U_2}, \dots, p_{U_d}\}$, consisting of values on d dimensions (e.g., age, gender, home city, etc). Each dimension has a range, or a set of possible values. Given a set of users \mathcal{U} at location L , the *location centric profile* at L , denoted by $LCP(L)$ is the set $\{LCP_1, LCP_2, \dots, LCP_d\}$, where LCP_i denotes the aggregate statistics over the i -th dimension of profiles of users from \mathcal{U} .

In the following, we focus on a single profile dimension, D . We assume D takes values over a range R that can be discretized into a finite set of sub-intervals (e.g., set of continuous disjoint intervals or discrete values). Then, given an integer b , chosen to be dimension specific, we divide R into b intervals/sets, R_1, \dots, R_b . For instance, gender maps naturally to discrete values ($b = 2$), while age can be divided into disjoint sub-intervals, with a higher b value.

We define the aggregate statistics S for dimension D of $LCP(L)$ to consist of b counters c_1, \dots, c_b ; c_i records the number of users from \mathcal{U} whose profile value on dimension D falls within range R_i , $i = 1..b$.

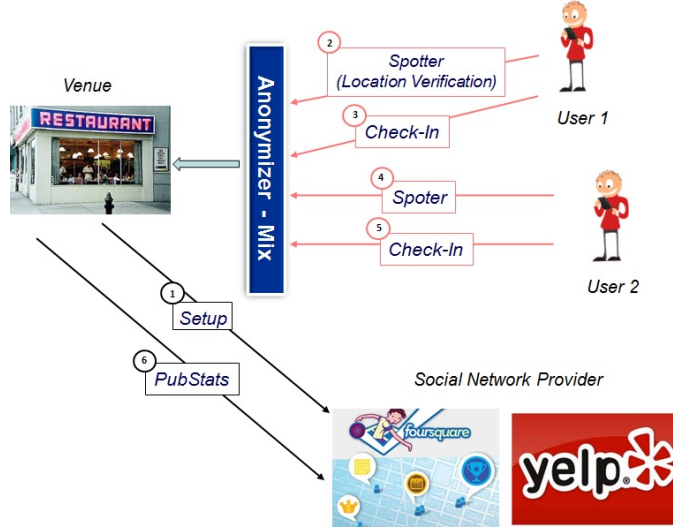


Figure 6.1: Solution architecture ($k=2$). The red arrows denote anonymous communication channels, whereas black arrows indicate authenticated (and secure) communication channels.

6.2.2 Private LCP Requirements

Let k be a security parameter, denoting the level of privacy we need to provide for users at any location. We then define a private LCP solution to be a set of functions, $PP(k) = \{Setup, Spotter, CheckIn, PubStats\}$, see Figure 6.1. *Setup* is run by each venue where user statistics are collected, to generate parameters for user check-ins. To perform a check-in, a user first runs *Spotter*, to prove her physical presence at the venue. *Spotter* returns error if the verification fails, success otherwise. If *Spotter* is successful, *CheckIn* is run between the user and the venue, and allows the collection of profile information from the user. Specifically, if the user's profile value v on dimension D falls within the range R_i , the counter c_i is incremented by 1. Finally, *PubStats* publishes collected LCPs. In the following, we use the notation $Prot(P_1(args_1), \dots, P_n(args_n))$ to denote protocol *Prot* run between participants P_1, \dots, P_n , each with its own arguments.

Let C_V be the set of counters defined at a venue V . We use \bar{C}_V to denote the set of sets derived from C_V as follows. Each set in \bar{C}_V differs from C_V in exactly one counter, whose value increments the value of the corresponding counter in C_V . For instance, if $C_V = \{2, 5, 9\}$, then $\bar{C}_V = \{\{3, 5, 9\}, \{2, 6, 9\}, \{2, 5, 10\}\}$. A private LCP solution needs to satisfy the following properties:

k -Privacy: Let \mathcal{A} denote an adversary that controls any number of venues and let \mathcal{C} denote a challenger controlling k users. \mathcal{C} runs *Spotter* followed by *CheckIn* at a venue V controlled by \mathcal{A} on behalf of $i < k$ users. Let C_i denote the resulting counter set. For each $j = 1..b$, \mathcal{A} outputs c'_j , its guess of the value of the j -th counter of C_i . The advantage of \mathcal{A} , $Adv(\mathcal{A}) = |Pr[C_i[j] = c'_j] - 1/(i + 1)|$, defined for each $j = 1..b$, is negligible.

Location Correctness: Let \mathcal{A} denote an adversary that controls the GSN provider and any number of users. Let \mathcal{C} be a challenger that controls a venue V . \mathcal{A} running as a user U not present at V , has negligible probability to successfully complete *Spotter* at V .

LCP Correctness: Let \mathcal{A} denote an adversary that controls the GSN provider and any number of users. Let \mathcal{C} be a challenger that controls a venue V . Let C_V denote the set of counters at V before \mathcal{A} runs *CheckIn* at V and let C'_V be the set of counters afterward. If $C'_V \notin \bar{C}_V$, the *CheckIn* completes successfully with only negligible probability.

Check-In Indistinguishability (CI-IND): Let a challenger \mathcal{C} control two users U_0 and U_1 and let an adversary \mathcal{A} control any number of venues. \mathcal{A} generates randomly q bits, b_1, \dots, b_q , and sends them to \mathcal{C} . For each bit b_i , $i = 1..q$, \mathcal{C} runs *Spotter* followed by *CheckIn* on behalf of user U_{b_i} . At the end of this step, \mathcal{C} generates a random bit b and runs *Spotter* followed by *CheckIn* on behalf of U_b at

a venue not used before. \mathcal{A} outputs a bit b' , its guess of b . The advantage of \mathcal{A} , $Adv(\mathcal{A}) = |Pr[b' = b] - 1/2|$ is negligible.

6.2.3 Geosocial Network Attacker Model

We assume venue owners are malicious and will attempt to learn private information from their patrons. Clients installed by users can be malicious, attempting to bias LCPs constructed at target venues. We consider a semi-honest, or honest-but-curious service provider. That is, the service provider is assumed to follow the protocol correctly, but attempts to learn personal user information as possible.

6.2.4 Safe City System Model

We consider a framework consisting of three participants, (i) a service provider, (ii) mobile device users and (iii) geosocial networks. The service provider, denoted by S , centralizes crime and census information and provides it upon request. We assume that the mobile devices are equipped with wireless interfaces, enabling the formation of transient, ad hoc connections with neighboring devices. Devices are also equipped with GPS interfaces, allowing them to retrieve their geographic location. Devices have Internet connectivity, which, for the purpose of this work may be intermittent. Users take advantage of Internet connectivity not only to communicate with the geosocial networks but also to retrieve safety information (both described in the following). Each user needs to install an application on her mobile device, which we henceforth denote as the *client*. Geosocial networks (GSNs) such as Yelp and Foursquare extend classic social networks with the notions of (i) venues, or businesses and (ii) *check-ins*.

6.3 Dataset.

6.3.1 Geosocial network data.

We have collected Yelp information from all the venues in the Miami-Dade county, Florida, for a total of 7699 venues. For each venue, we have collected the name, type and address, along with the list of reviews received. For each review, we collected the home city and state of the reviewer. The supplemental material includes plots showing that (i) the number of reviews received by Miami-Dade venues exhibits a long tail distribution and (ii) Yelp reviews are mostly positive as most aggregate ratings are at or above 4 stars.

6.3.2 Crime and Census data.

We use a historical database of more than 2.3 million crime incidents reported in the Miami Dade county area since 2007 [Ter]. Each record is labeled with a crime type (e.g., homicide, larceny, robbery, etc), the time and the geographic location where it has occurred. We mapped crimes into 7 categories: Murder, Forcible Rape, Aggravated Assault, Robbery, Larceny/Theft, Burglary/Arson, Motor Vehicle Theft. We removed minor crime reports that did not fall into these categories. Let c denote the number of crime types. In our case, $c = 7$. Let $\overline{CT} = \{CT_1, \dots, CT_c\}$ denote the set of crime types. We also use Census data sets [Cen10], reporting population counts and demographic information. The data is divided into polygon shaped geographical extents called *census block groups*. Each block contains information about the population within (e.g., population count, various statistics). According to the data, Miami Dade county has a population of 2,496,435. The supplemental material includes more details of the data classification process and a plot showing the Miami-Dade population density, at block granularity.

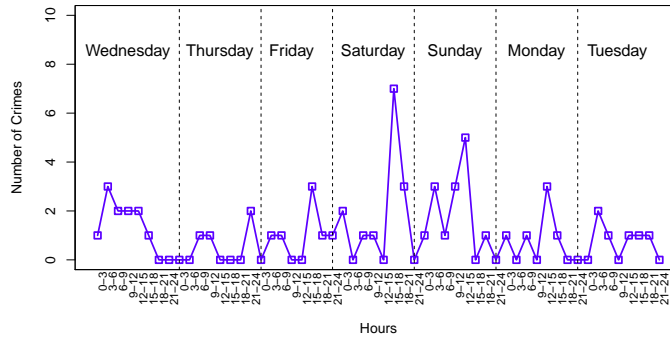


Figure 6.2: 1 week (July 13-19, 2011) evolution of the number of crimes reported within one Miami-Dade block.

6.4 Proposed Methods

6.4.1 Profil_R : A framework for constructing location centric profiles

As mentioned before, SPOTR_V denote the device installed at venue V . For each user profile dimension D , SPOTR_V stores a set of *encrypted counters* – one for each sub-range of R .

Overview. Initially, and following each cycle of k check-ins executed at venue V , SPOTR_V initiates *Setup*, to request the provider S to generate a new Benaloh key pair. Thus, at each venue time is partitioned into *cycles*: a cycle completes once k users have checked-in at the venue. The communication during *Setup* takes place over an authenticated and secure channel (see Figure 6.1).

When a user U checks-in at venue V , it first engages in the *Spotter* protocol with SPOTR_V , allowing the venue to verify U 's physical presence. A successful run of *Spotter* provides U with a share of the secret key employed in the Benaloh cryptosystem of the current cycle. For each venue and user profile dimension, S stores a set Sh of shares of the secret key that have been revealed so far.

Subsequently, U runs *CheckIn* with SPOTR_V , to send its share of the secret key and to receive the encrypted counter sets. As shown in Figure 6.1, the communication takes place over an anonymous channel to preserve U 's privacy. During *CheckIn*, for each dimension D , U increments the counter corresponding to her range, re-encrypts all counters and sends the resulting set to SPOTR_V . U and SPOTR_V engage in a zero knowledge protocol that allows SPOTR_V to verify U 's correct behavior: exactly one counter has been incremented. SPOTR_V stores the latest, proved to be correct encrypted counter set, and inserts the secret key share into the set Sh .

Once k users successfully complete the *CheckIn* procedure, marking the end of a cycle, SPOTR_V runs *PubStats* to reconstruct the private key, decrypt all encrypted counters and publish the tally. The communication during *PubStats* takes place over an authenticated channel (see Figure 6.1).

The Solution

Let C_i denote the set of encrypted counters at V , following the i -th user run of *CheckIn*. $C_i = \{C_i[1], \dots, C_i[b]\}$, where $C_i[j]$ denotes the encrypted counter corresponding to R_j , the j -th sub-range of R . We write $C_i[j] = E(u_j, u'_j, c_j, j) = [E(u_j, c_j), E(u'_j, j)]$, where u_j and u'_j are random obfuscating factors and $E(u, M)$ denotes the Benaloh encryption of a message M using random factor u . That is, an encrypted counter is stored for each sub-range of domain R of dimension D . The encrypted counter consists of two records, encoding the number of users whose values on dimension D fall within a particular sub-range of R .

Let $RE(v_j, v'_j, E(u_j, u'_j, c_j, j))$ denote the re-encryption of the j -th record with two random values v_j and v'_j : $RE(v_j, v'_j, E(u_j, u'_j, c_j, j)) = [RE(v_j, E(u_j, c_j)), RE(v'_j, E(u'_j, j))] = [E(u_j v_j, c_j), E(u'_j v'_j, j)]$. Let $C_i[j] + + =$

$E(u_j, u'_j, c_j + 1, j)$ denote the encryption of the incremented j -th counter. Note that incrementing the counter can be done without decrypting $C_i[j]$ or knowing the current counter's value: $C_i[j] + + = [E(u_j, c_j)y, E(u'_j, j)] = [y^{c_j+1}u_j^r, E(u'_j, j)] = [E(u_j, c_j + 1), E(u'_j, j)]$.

In the following we use the above definitions to introduce Profil_R . Profil_R instantiates $PP(k)$, where k is the privacy parameter. The notation $P(A(\text{params}_A), B(\text{params}_B))$ denotes the fact that protocol P involves participants A and B , each with its own parameters.

Setup($V(), S(k)$): The provider S runs the key generation function $KG(l)$ of the Benaloh cryptosystem (see Section 5.2.4). Let p and q be the private key and n and y the public key. S sends the public key to SPOTR_V . SPOTR_V generates a signature key pair and registers the public key with S . For each user profile dimension D of range R with b sub-ranges, SPOTR_V performs the following steps:

- Initialize counters c_1, \dots, c_b to 0.
- Generate $C_0 = \{E(x_1, x'_1, c_1, 1), \dots, E(x_b, x'_b, c_b, b)\}$, where $x_i, x'_i, i = 1..b$ are randomly chosen values. Store C_0 indexed on dimension D .
- Initialize the share set $S_{key} = \emptyset$.
- Generate system wide parameters k and $m > k$ and initialize the (k, m) TSS.

Spotter($U(L, T), V(), S(k)$): Let L and T denote U 's location and current time. To ensure anonymity, U generates fresh random MAC and IP addresses. These addresses are used for a single execution of the *Spotter* and *CheckIn* protocols. SPOTR_V uses one of the location verification procedures proposed in [CP12] to verify U 's presence at L and T (see Section 5.2.4).

Let U be the i -th user checking-in at V . If the verification succeeds and $i \leq k$, S uses the (k, m) TSS to compute a share of p (Benaloh secret key, factor of the

modulus n). Let p_i be the share of p . S sends the (signed) share p_i to U . If $i > k$, S calls *Setup* to generate new parameters for V .

CheckIn($U(p_i, n, V), V(n, y, C_{i-1}, S_{key})$): Executes only if the previous run of *Spotter* is successful. U uses the same random MAC and IP addresses as in the previous *Spotter* run. Let U be the i -th user checking-in at V . Then, C_{i-1} is the current set of encrypted counters. $SPOTR_V$ sends C_{i-1} to U . Let v , U 's value on dimension D , be within R 's j -th sub-range, i.e., $v \in R_j$. U runs the following steps:

- Generate b pairs of random values $\{(v_1, v'_1), \dots, (v_b, v'_b)\}$. Compute the new encrypted counter set C_i , where the order of the counters in C_i is identical to C_{i-1} : $C_i = \{RE(v_l, v'_l, C_{i-1}[l]) | l = 1..b, l \neq j\} \cup RE(v_j, v'_j, C_{i-1}[j] + +)$.
- Send C_i and the signed (by S) share p_i of p to V .

If $SPOTR_V$ successfully verifies the signature of S on the share p_i , U and $SPOTR_V$ engage in a zero knowledge protocol ZK-CTR (see Section 6.4.1). ZK-CTR allows U to prove that C_i is a correct re-encryption of C_{i-1} : only one counter of C_{i-1} has been incremented. If the proof verifies, $SPOTR_V$ replaces C_{i-1} with C_i and adds the share p_i to the set S_{key} . Otherwise, $SPOTR_V$ drops C_i and rolls back to C_{i-1} .

PubStats($V(C_k, Sh, V), S(p, q)$): $SPOTR_V$ performs the following actions:

- If $|Sh| < k$, abort.
- If $|Sh| = k$, use the k shares to reconstruct p , the private Benaloh key.
- Use p and $q = n/p$ to decrypt each record in C_k , the final set of counters at V . Publish results.

ZK-CTR: Proof of Correctness

We now present the zero knowledge proof of the set C_i being a correct re-encryption of the set C_{i-1} , i.e., a single counter has been incremented. Let ZK-CTR(i) denote the protocol run for sets C_{i-1} and C_i . U and SPOTR_V run the following steps s times:

- U generates random values $(t_1, t'_1), \dots, (t_b, t'_b)$ and random permutation π , then sends to SPOTR_V the proof set $P_{i-1} = \pi\{RE(t_l, t'_l, C_{i-1}[l]), l = 1..b\}$.
- U generates random values $(w_1, w'_1), \dots, (w_b, w'_b)$. It sends to SPOTR_V the proof set $P_i = \pi\{RE(w_l, w'_l, C_i[l]), l = 1..b\}$
- SPOTR_V generates a random bit a and sends it to U .
- If $a = 0$, U reveals random values $(t_1, t'_1), \dots, (t_b, t'_b)$ and $(w_1, w'_1), \dots, (w_b, w'_b)$. SPOTR_V verifies that for each $l = 1..b$, $RE(t_l, t'_l, C_{i-1}[l])$ occurs in P_{i-1} exactly once, and that for each $l = 1..b$, $RE(w_l, w'_l, C_i[l])$ occurs in P_i exactly once.
- If $a = 1$, U reveals $o_l = v_l w_l t_l^{-1}$ and $o'_l = v'_l w'_l t_l'^{-1}$, for all $l = 1..b$ along with j , the position in P_{i-1} and P_i of the incremented counter. SPOTR_V verifies that for all $l = 1..b, l \neq j$, $RE(o_l, o'_l, P_{i-1}[l]) = P_i[l]$ and $RE(o_j, o'_j, P_{i-1}[j]y) = P_i[j]$.
- If any verification fails, SPOTR_V aborts the protocol.

Preventing Venue-User Collusion

For simplicity of presentation, we have avoided the Sybil attack problem: participants that cheat through multiple accounts they control or by exploiting the anonymizer. For instance, a rogue venue owner, controlling $k-1$ Sybil user accounts or simulating $k-1$ check-ins, can use Profil_R to reveal the profile of a real user. Conversely, a rogue user (including the venue) could bias the statistics built by the

venue (and even deny service) by checking-in multiple times in a short interval. Sybil detection techniques (see Section 2.1.2) can be used to control the number of fake, Sybil accounts. However, the use of the anonymizer prevents the provider and the use of the unique IP and MAC addresses prevents the venue from differentiating between interactions with the same or different accounts. In this section we propose a solution, that when used in conjunction with Sybil detection tools, mitigates this problem. The solution introduces a trade-off between privacy and security. Specifically, we divide time into epochs (e.g., one day long). A user can check-in at any venue at most once per epoch. When active, once per epoch e , each user U contacts the provider S over an authenticated channel. U and S run a blind signature [Cha82] protocol: U obtains the signature of S on a random value, $R_{U,e}$. S does not sign more than one value for U for any epoch. In runs of *Spotter* and *CheckIn* during epoch e , U uses $R_{U,e}$ as its pseudonym (i.e., MAC and IP address). Venues can verify the validity of the pseudonym using S 's signature. A venue accepts a single *CheckIn* per epoch from any pseudonym, thus limiting the user's impact on the LCP. The privacy breach mentioned above is due to the fact that now S can correlate *CheckIns* executed using the same $R_{U,e}$. However, S does not know the real user identity behind $R_{U,e}$ – due to the use of blind signatures.

Snapshot LCP

We extend Profil_R to allow not only venues but also users to collect *snapshot* LCPs of other, co-located users. To achieve this, we take advantage of the ability of most modern mobile devices (e.g., smartphones, tablets) to setup ad hoc networks. Devices establish local connections with neighboring devices and privately compute the instantaneous aggregate LCP of their profiles.

Snapshot Profil_R

We assume a user U co-located with k other users U_1, \dots, U_k . U needs to generate the LCP of their profiles, without infrastructure, GSN provider or venue support. An additional difficulty then, is that participating users need assurances that their profiles will not be revealed to U . However, one advantage of this setup is that location verification is not needed: U intrinsically determines co-location with U_1, \dots, U_k . Snapshot Profil_R consists of three protocols, $\{Setup, LCPGen, PubStats\}$:

Setup($U(r), U_1, \dots, U_k()$): U runs the following steps:

- Run the key generation function $KG(l)$ of the Benaloh cryptosystem (see Section 5.2.4). Send the public key n and y to each user U_1, \dots, U_k .
- Engage in a multi-party secure function evaluation protocol [JJ00] with U_1, \dots, U_k to generate shares of a public value $R < n$. At the end of the protocol, each user U_i has a share R_i , such that $R_1 \dots R_k = R \bmod n$ and R_i is only known to U_i .
- Assign each of the k users a unique label between 1 and k . Let U_1, \dots, U_k denote this order.
- Generate $C_0 = \{E(x_1, x'_1, 0, 1), \dots, E(x_b, x'_b, 0, b)\}$, where x_i, x'_i , $i = 1..b$ are randomly chosen. Store C_0 indexed on dimension D .

Each of the k users engages in a 1-on-1 *LCPGen* with U to privately and correctly contribute her profile to U 's LCP.

LCPGen($U(C_{i-1}), U_i()$): Let C_{i-1} be the encrypted counters after U_1, \dots, U_{i-1} have completed the protocol with U . U sends C_{i-1} to U_i . U_i runs the following:

- Generate random values $(v_1, v'_1), \dots, (v_b, v'_b)$. Let j be the index of the range where U_i fits on dimension D .

- Compute the new encrypted counter set C_i as: $C_i = \{RE(v_l, v'_l, C_{i-1}[l])R_i \bmod n | l = 1..b, l \neq j\} \cup RE(v_j, v'_j, C_{i-1}[j] + +)R_i \bmod n\}$ and send it to U .
- Engage in a ZK-CTR protocol to prove that $C_i \in \bar{C}_{i-1}$. The only modification to the ZK-CTR protocol is that all re-encrypted values are also multiplied with $R_i \bmod n$, U_i 's share of the public value R . If the proof verifies, U replaces C_{i-1} with C_i .

After completing *LCPGen* with U_1, \dots, U_k , U 's encrypted counter set is $C_k = \{E_j = E(u_j, u'_j, c_j, j)R_1 \dots R_k | j = 1..d\}$, where u_j and u'_j are the product of the obfuscation factors used by U_1, \dots, U_k in their re-encryptions. The following protocol enables U to retrieve the snapshot LCP.

PubStats($U(C_k)$) : Compute $E_j K$, $\forall j = 1..d$, where $K = R^{-1} \bmod n$ ($R = R_1 \dots R_k$), decrypt the outcome using the private key (p, q) and publish the resulting counter value. U verifies that the j -th decrypted record is of format (c_j, j) and that the sum of all counters equals k . If any verification fails, U drops the statistics - a cheater exists. Otherwise, the resulting counters denote the aggregate stats of U_1, \dots, U_k .

Even though U has the private key allowing it to decrypt any Benaloh ciphertext, the use of the secret R_i values prevents it from learning the profile of U_i , $i = 1..k$.

This protocol is a secure function evaluation - the participants learn their aggregated profiles, without learning the profiles of any participant in the process. We note however that existing SFE solutions cannot be used here: We need to ensure the input user profiles are correct, that is, each user increments a single counter.

6.4.2 Safety for Geosocial Networks

Location based Safety

We exploit the crime dataset to define an initial, location-centric safety metric. We divide space into census blocks. We divide time into fixed-length epochs, e.g., 1 hour long, 24 epochs per day. To understand the need for a time dependent safety metric, we have studied the evolution in time of crimes reported within blocks of the Miami-Dade county. Figure 6.2 shows the evolution over seven consecutive days (Wed.-Tue., July 13-19, 2011) of the number of crimes reported within one such block, with a 3 hour time granularity. Most of the events are larcenies. The plot shows that the number of crimes reported varies abruptly throughout a day. Case in point, on the depicted Saturday, 7 crimes are reported between hours 15-18, 3 crimes between 18-21 and 0 between 21-24. Thus, a time-invariant aggregate of past crime events is unlikely to accurately define the present. The supplemental material includes a similar plot, drawn for the same block, over an interval of 18 consecutive weeks.

Block crime and safety indexes. For a census block B and an epoch e denoted by the time interval ΔT , let $C(B, \Delta T)$ represent a c -dimensional vector, where the i -th entry denotes the number of crimes of type $CT[i]$ recorded in block B during interval ΔT . Let \overline{W} denote a c -dimensional vector of weights; each crime type of \overline{CT} (defined in Section 6.4.2) has a weight proportional to its seriousness (defined shortly). Let $BC(\Delta T)$ denote the population count recorded for block B . We then define the *crime index* of block B during interval ΔT as

$$CI(B, \Delta T) = \min\left\{\frac{C(B, \Delta T)\overline{W}}{BC(\Delta T)}, 1\right\} \quad (6.1)$$

where $C(B, \Delta T)\overline{W}$ denotes the vectorial product between the number of crimes per type and the weights of the crime types. That is, B 's crime index is the per-

Crime Type	Weight
Assault	0.176
Robbery	0.180
Rape	0.307
Homicide	0.336

Table 6.1: Crime weight assignment using the FCPC.

capita weighted average of crimes recorded during interval ΔT . The safety index SI of block B during interval ΔT is then defined as

$$SI(B, \Delta T) = 1 - CI(B, \Delta T) \quad (6.2)$$

Both the CI and SI metrics take values in the $[0, 1]$ interval. In the evaluation section we show that crime index values of blocks in the Miami-Dade county are always smaller than 1. Higher $SI(B, \Delta T)$ values denote safer blocks.

Crime weight assignment. We need to assign meaningful weights to the crime types \overline{CT} . An inappropriate assignment may make a large number of “lighter” offenses overshadow more serious but less frequent crime events, (e.g., consider larcenies vs. homicides). We propose to assign each crime type a weight proportional to its seriousness, defined according to the criminal punishment code, i.e., the Florida Criminal Punishment Code (FCPC) [oC]. The FCPC is divided into *levels* ranging 1-10, and each level L_k contains different types of felonies. The higher the level, the more serious is the felony. Each felony has a *degree*, (i.e., capital, life, first, second and third degree, sorted in decreasing order of seriousness), with an associated punishment (years of imprisonment) [Hor].

Let L_k denote the set of felonies within level k and let P_k denote the set of corresponding punishments. Let $l_k = |L_k|$ denote the number of felonies within

level k . Then, we define the weight of crime type $CT[i]$, \bar{w}_i , as

$$\bar{w}_i = \sum_{k=1}^{10} \rho_k \frac{P_k[i]}{\sum_{j=1}^{l_k} P_k[j]},$$

where $\rho_k = k / \sum_{i=1}^{10} i$ is the weight assigned to level k (normalized to the sum of the number of levels). Thus, the weight of crime type $CT[i]$ is the weighted sum of the per-level punishment value ($P_k[i]$) associated with the occurrence of $CT[i]$ within the felonies of level k , normalized to the total punishment of level k . Table 6.1 shows the resulting weights.

Example. We study the impact of level L_8 on the weight of the “Robbery” crime. Out of the felonies represented on level 8, two are related to “Robbery”: “Robbery with a weapon” and “Home-invasion robbery”. Both are first degree felonies, therefore punishable with up to 30 years of imprisonment. The other represented felonies are “Homicide”, with 6 different counts, for a total of 135 years penalty and “Rape”, with 1 count of up to 15 years penalty. Thus, the contribution of level 8 to the weight of “Robbery” is $\frac{8}{55} \times \frac{60}{60+135+15} = 0.0415$.

Illustration. We use the Miami-Dade crime set to illustrate the geographic distribution of block-level safety index information, where the epoch, denoted by the interval ΔT , is the year 2010. We use the census dataset to extract the population count $BC(\Delta T)$. Figure 6.3 shows the color-coded safety index for each block group in the Miami-Dade county (FL) where crimes have been reported during 2010. The safety index considers only crimes against persons. Grey blocks have a very low reported crime level. Green blocks denote safer locations while darker yellow and red blocks denote areas with more reported crimes.

Predicting Safety

The crime index computation of Equation 6.1 can only be performed for past epochs, when all crime events have been reported. Safety information however is most useful

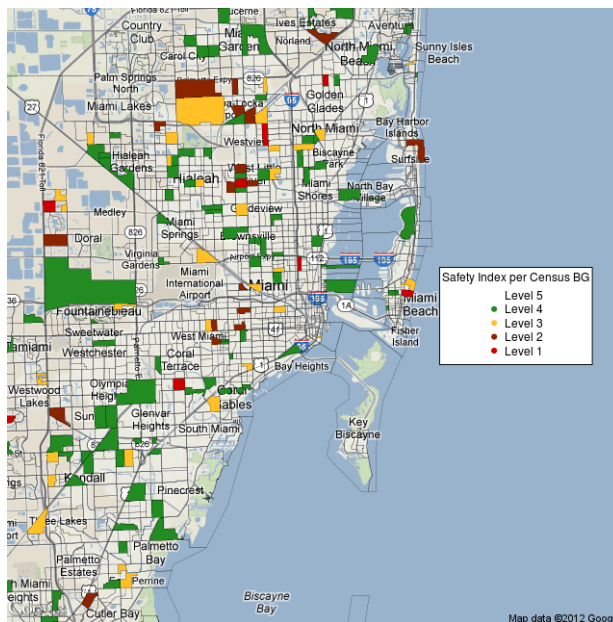


Figure 6.3: Safety index illustration for the Miami-Dade county: $SI(B, \Delta T)$ values are mapped into color-coded “safety levels”: the higher the level, the safer the block.

when provided for the present or near future. One way to predict the crime index of a block B for the next epoch (denoted by the interval ΔT), $PCI(B, \Delta T)$, is the average crime index of the block during the same epoch in the day for the past d days, where d is a system parameter (e.g., $d=7$ for 1 week of recorded per-block history). This solution however is unable to detect and factor in all crime periodicities, including seasonal, weekly and daily fluctuations. As such, it may include unnecessary errors – e.g., higher number of crimes in a past August may introduce inaccuracies in the crime index considered in the current month of April.

We propose to address this issue through the use of the time series forecasting techniques discussed in Section 5.2.4. Specifically, we use time series forecasting tools to compute long and short term predictions of the number of crimes to be committed within an area (e.g., census block, zipcode, city, etc), based on the area’s recorded history. Section 6.5.2 evaluates the ability of the time series forecasting tools to accurately predict near-future crime counts.

Predicting crime and safety indexes. At the beginning of each epoch (denoted by the time interval ΔT), compute predictions for the number of crimes of each crime type to be reported at each census block B during the epoch. Let $PC(B, \Delta T)[i]$ denote the predicted number of crimes of type $CT[i]$. Using a formula similar to Equation 6.1 compute the predicted crime index for B during interval ΔT as $PCI(B, \Delta T) = \min\{PC(B, \Delta T)\bar{W}/BC(\Delta T), 1\}$. The predicted safety index is then $PSI(B, \Delta T) = 1 - PCI(B, \Delta T)$.

Personalized, Context-Aware Safety

The ultimate goal of defining crime and safety indexes is to provide users with safety advisory information. People are however not equally exposed and vulnerable to all crime types. Age, gender and an array of personal features, preferences and choices play a central role on the perception of an individual’s safety. Since such information may not be readily accessible, we use instead the localization capabilities of a user’s mobile device to periodically record and locally store her trajectory trace. This enables us to define the crime index level with which a user is comfortable: the average crime index of the locations in her trajectory. We then introduce personalized safety recommendations both when enough crime information exists to enable the prediction of the near-future crime index of a location and when insufficient such information exists.

We propose to exploit the context of a location, through the people located there. We use the trajectory trace of the user to define the chance of a crime to occur around the user and generalize this approach to compute the chance of a crime to occur around groups of users. This enables us to introduce the concept of *context aware safety*: a user is safe if the chance of a crime to occur around her equals or exceeds the chance of a crime to occur around her co-located users.

Personalized User Safety.

We extend the crime and safety index definitions from locations to users. We assume the device can capture the location of the user with block level precision. Let $TJ_U = \{[B_i, T_i, CI(B_i, \Delta T_i)] | i = 1..h\}$ denote the trajectory trace of user U , consisting of recorded [block, epoch, crime index] tuples. ΔT_i denotes the epoch containing time T_i , when U was present at block B_i , $T_i \in \Delta T_i$. For privacy reasons, we require each user to store her trajectory trace on her device.

We define the *vicinity crime* metric for a user U , V_U to be the percentage of the user's trajectory places where crimes have been reported around the time of her visit:

$$V_U = \frac{\sum_{i=1}^h \text{sgn}(CI(B_i, \Delta T_i))}{h} \quad (6.3)$$

$\text{sgn}(x)$ denotes the sign function, that is 0 when x is 0, and 1 when x is larger than 0. For instance, if a user has 100 locations in her trajectory and crimes have been reported at 60 of those locations during the epoch of the user's presence, the user's vicinity crime metric is 60%. We then define the crime index of a user U to be the average crime index of locations in her trajectory:

$$CI_U = \frac{\sum_{i=1}^h CI(B_i, \Delta T_i)}{h} \quad (6.4)$$

Safety Decision With Accurate Crime Data.

We assume first that user U is located at time T_c in a block B , where accurate past crime data exists. This allows the proper prediction of the crime index, thus the computation of the predicted crime index $PCI(B, \Delta T)$, as specified in Section 6.4.2. ΔT denotes the current epoch, $T_c \in \Delta T$. We then introduce the notion of personalized safety recommendation:

Definition 6.4.1 (*Personalized safety*). A user U is safe at a block B within time interval ΔT , if $CI_U \geq PCI(B, \Delta T)$.

Intuition. A user is safe if the user’s crime index equals or exceeds the block’s crime index predicted for the duration of the user’s presence. If the crime index of the user’s current block, predicted for the epoch of the user’s presence, does not exceed the user’s level of comfort, it means the user has spent at least half of her time in locations with more crime than the current location. Thus, the user is likely to be comfortable with the crime level of her current location.

Safety Decision Without Accurate Crime Data.

Certain locations may have insufficient crime data to ensure an accurate prediction of the location’s crime index. For instance, as shown in Figure 6.2, the number of recorded events can quickly spike or drop to 0 in short time intervals. Accurately predicting event counts within a short time interval is difficult, as the difference between 0 and 1 crimes is significant. This is the case also during unexpected events (natural and man made disasters) when the future does not reflect the past. To address this issue, we propose to use existing context information, collected from co-located users.

Our approach is the following. We define the safety index of a user U to be the chance of no event being reported in her vicinity: $SI_U = 1 - V_U$. Let U_1, \dots, U_k be the users co-located with user U . We define a *super user* $SUP_{1..k}$, as a fictitious user whose trajectory trace encompasses the trajectories of users U_1, \dots, U_k . That is, $TJ_{U_{1..k}} = TJ_{U_1} \cup \dots \cup TJ_{U_k}$. We note that both users and super users can be located in multiple blocks during the same epoch. We then use Equation 6.3 to compute the vicinity crime metric of $SUP_{1..k}$, $V_{SUP_{1..k}}$. We define the safety index, $SI_{SUP_{1..k}} = 1 - V_{SUP_{1..k}}$. These definitions enable us to introduce the notion of personalized safety recommendation:

Definition 6.4.2 (*Context-aware safety*). A user U is safe in a context consisting of neighboring users U_1, \dots, U_k , if $SI_U \leq SI_{SUP_{1..k}}$, i.e., $V_U \geq V_{SUP_{1..k}}$.

Thus, a user is safe if surrounded by users whose aggregate safety index is higher or equal to the user’s safety index.

Intuition. The safety index of a user encodes the probability that no event occurs around the user. The safety index of a group of users (e.g., $SUP_{1..k}$) is defined as the chance that no event occurs around the group. Definition 6.4.2 states that a user is safe if it is surrounded by a group of users whose aggregated chance of no event occurring is higher or equal to the user’s chance of no event occurring. A low safety index value does not imply the user is unsafe, but merely the fact that the user spends time in places where events do occur. If the location sampling process is done periodically, the formula naturally ensures that blocks where the user spends more time have more impact on the user’s safety index. Being around a group of users whose aggregated safety index is low suggests that the place is likely to have a low safety level.

Factoring in duration of stay. The duration of a user’s presence within a block needs to be considered when determining the user’s safety. For instance, walking through an unsafe block should be avoided. However, when driving on a highway, an unsafe block raises lower safety concerns. One way to address this issue is by using smaller epochs. Another approach is, given a user’s trajectory trace, predict the time the user will spend within the current block. The block should raise safety concerns only if the predicted interval exceeds a certain threshold.

iSafe.

User trajectories contain sensitive information, including blocks of interest and behavior patterns. We introduce iSafe, a distributed algorithm that allows the aggregation of trajectory traces of co-located users while preserving the privacy of involved

participants. iSafe achieves this by taking advantage of the wireless communication capabilities of user mobile devices to form short lived, ad hoc communities.

Overview. iSafe contacts the neighboring devices, reachable over local wireless interfaces, that run iSafe. If their number exceeds a (system wide) parameter value, iSafe initiates a multiparty computation. The procedure enables iSafe to privately and distributively compute the total number of blocks visited by the owners of those devices as well as the total number of blocks visited that had crimes committed during their presence. This enables iSafe to compute their aggregated vicinity crime index, and rely on Definition 6.4.2 to decide the user’s safety.

Details. Algorithm 4 contains the pseudocode of iSafe. Its main procedure is *safetyDecision*(ΔT), executed periodically by a client C , at C ’s current block, B . In the first step, C contacts the service provider S , storing the crime and Census datasets. C retrieves the predicted crime index of the block B where the user is located. This operation is performed privately, by using a private information retrieval technique [Gas04]. This prevents S from learning the current location of C .

If the crime index of the block can be accurately predicted, the operation returns the decision according to Definition 6.4.1. Otherwise, it invokes the *cas* operation. *cas* first discovers all the ad hoc neighbors of the user. If the number of neighbors is below a system-wide threshold value, $NThr$, it returns -1: not enough information exists to provide an accurate recommendation, and not enough privacy is provided. Otherwise, it invokes the *multiPartySum* operation twice, with different input arguments. When invoked with argument 0, *multiPartySum* calculates BWC_{SUP} , the sum of the blocks with crimes visited by all the user’s neighbors. When invoked with argument 1, *multiPartySum* calculates $TBlk_{SUP}$, the sum of the total blocks visited by all the user’s neighbors.

Algorithm 4 iSafe pseudocode.

```
1. Object implementation iSafe;
2. neighbor[] N;           #set of neighbors
3. double CI, SI;         #crime, safety indexes
4. double V;              #vicinity crime prob
5. BigInteger R;          #random value
6. BigInteger[] shares;    #set of shares
7. BigInteger[] NShares;  #shares of neighbors

8. int BWC;                #blocks with crime
9. int TBlk;               #total blocks visited

10. Operation int safetyDecision(Epoch  $\Delta T$ )
11.   B := getCurrentBlock();
12.   PCIB := S.getPCI(B,  $\Delta T$ );
13.   if (PCIB! = -1) then return (CI  $\geq$  PCIB);
14.   else return cas(); fi end

15. Operation int cas()
16.   N := discoverNeighbors();
17.   if (N.size < NThr) then return - 1;
18.   BWCSUP := multiPartySum(0) - BWC;
19.   TBlkSUP := multiPartySum(1) - TBlk;
20.   return(V  $\geq$  BWCSUP/TBlkSUP); end

21. Operation BigInteger multiPartySum(int type)
22.   R := getRandom();
23.   shares := split(R, N.size);
24.   for i := 1 to N.size do
25.     send(N[i], shares[i]);
26.     NShares[i] := recv(N[i]); od
27.   int order := electLeaderOrder();
28.   BigDecimal S := 0; int count := 0;
29.   while (count < N.size) do
30.     count := count + 1;
31.     if (count = order) then
32.       if (type = 0) then S := S + BWC + R;
33.       else S := S + TBlk + R; fi
34.       for i := 1 to |N| do S := S - NShares[i]; od
35.       mcast(S);
36.     else S := recv(); fi od
37.   return S; end
```

The *multiPartySum* operation is a secure multi-party sum evaluation. It achieves privacy through the use of (i) frequently changing, random MAC addresses for user devices and (ii) secret splitting. Each client generates a random value and splits it into shares – one for each neighbor. That is, if the random value is R , the shares sh_1, \dots, sh_k are generated randomly such that $\sum_{i=1}^k sh_i = R$. The client sends each share to one neighbor and receives a share from each neighbor. The clients engage in a leader election and order selection distributed algorithm, where each client is assigned a unique identifier, between 1 and k .

When a client’s turn comes, according to the order established, it adds either the user’s BWC value (number of census blocks with events visited by the user) or the user’s TBlk value (total number of blocks visited), according to the input variable *type*, and adds its random value R to the overall sum (S). It then subtracts all the shares of secrets of its neighbors and sends a multicast of the result, reaching all its neighbors. If it is not the user’s turn to transmit, the client waits to receive the multicast values of its neighbors.

The ratio of the computed BWC_{SUP} and $TBlk_{SUP}$ values is the vicinity crime metric of the super user representing the neighbors of C . *cas* returns the safety decision of Definition 6.4.2.

Analysis.

We first define the notion of location privacy in terms of the inability of an adversary \mathcal{A} to guess the location of a user with probability non-negligibly higher than $1/n$, where n is the number of blocks supported by the system.

Definition 6.4.3 (*Location Privacy*). *Let \mathcal{A} control the provider S and any number of clients, such that the number of clients controlled by \mathcal{A} at any location is at most $NThr - h$, where $NThr$ and $h > 1$ are integer parameters. The challenger \mathcal{C} controls one client, *Client*. \mathcal{A} contacts \mathcal{C} at any time T . \mathcal{C} invokes *safetyDecision*(ΔT) on behalf of *Client*, where B denotes the current block of *Client* and $T \in \Delta T$. \mathcal{A} outputs B' , its guess of the block B where *Client* is located. We say a solution provides location privacy if the advantage of \mathcal{A} in this game, $Adv_{\mathcal{A}} = |Pr[B' = B] - 1/n|$ is negligible.*

We introduce several results whose proofs are included in the supplemental material, along with techniques for preventing an adversary from tampering with safety information.

Theorem 2 *An adversary \mathcal{A} controlling $k - h$ out of k participants in the iSafe algorithm, can only find the sum of the input values of the remaining h honest participants.*

Theorem 3 *iSafe provides location privacy.*

An adversary can attempt to use iSafe to identify and target areas considered to be safe. However, safety is personalized: areas denoted “safe” for the adversary may not necessarily be safe for other users, who may in effect avoid them. iSafe is also adaptive: newly reported incidents as well as the lack of incidents are used to continuously adjust block safety values.

iSafe Implementation

We implemented iSafe as a (i) web server, (ii) a browser plugin running in the user’s browser and (iii) a mobile app.

Browser Plugin.

We implemented a plugin for the Chrome browser using HTML, CSS and Javascript. The plugin interacts with Yelp pages and the web server, using content scripts (Chrome specific components that let us access the browser’s native API) and cross-origin XMLHttpRequests. The plugin becomes active when the user navigates to a Yelp page. For user and venue pages, it parses their HTML files and retrieves their reviews. We employ a stateful approach, where the server’s SQLite DB stores all reviews of pages previously accessed by users. This enables significant time savings, as the plugin needs to send to the web server only reviews written after the date of the last user’s access to the page.

Given the venue’s set of reviews, the server determines the corresponding reviewers. The crime index of blocks of venues reviewed by each user generate the crime



Figure 6.4: Snapshots of iSafe on Android.

index of the user. Crime indexes of reviewers are used to compute the crime index of the venue. The server sends back this information, which the plugin displays in the browser using color codes, ranging from green (safe) to red (unsafe). The supplemental material shows a snapshot of the browser plugin.

Mobile iSafe.

We have implemented the location centric static safety labeling component of mobile iSafe using Android. We used the Android Maps API to facilitate the location based service employed by our approach. iSafe periodically retrieves the user's current GPS location, derives the current census block and also the corresponding crime index. It stores the user's trajectory as one record $[block, time, crime_index]$ in a local SQLite database. The initial threshold value for creating a new record is 60 seconds.

iSafe uses Bluetooth to compute the vicinity crime metrics of the user's neighbors. We implemented a client-server Bluetooth communication protocol where each device acts as a server and other connected devices act as clients per P2P

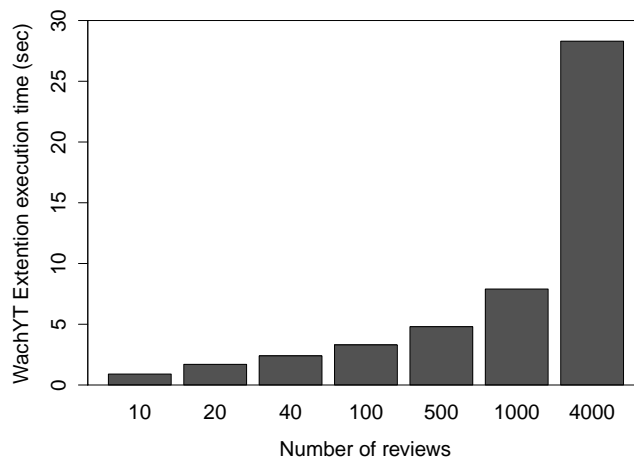


Figure 6.5: iSafe browser plugin overhead: Collecting reviews from venues, as a function of the number of reviews.

communication. When compared to Wi-Fi, Bluetooth has drawbacks concerning the transmission range, complexity of the pairing process and the number of communicating peers. However, it also has an important advantage: energy efficiency. Bluetooth consumes less energy than Wi-Fi interfaces, particularly when idle, thus motivating users to leave it always on. iSafe has a separate background service that displays in the status bar of the Android device, the safety color label of the user’s current location. Figures 6.4(a) and 6.4(b) show snapshots of the functionality of the mobile iSafe application.

6.5 Empirical Evaluation

6.5.1 Evaluation of Profil_R .

For testing purposes we have used Samsung Admire smartphones running Android OS Gingerbread 2.3 with a 800MHz CPU and a Dell laptop equipped with a 2.4GHz Intel Core i5 processor and 4GB of RAM for the server. For local connectivity the

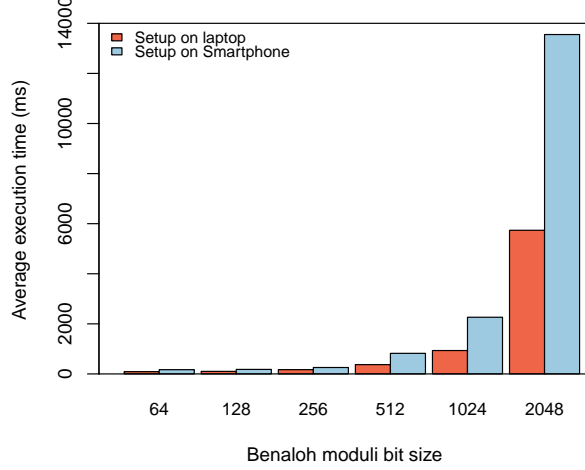


Figure 6.6: *Setup* dependence on Benaloh modulus size. Note the significant increase to 13.5s for a 2048 bit modulus. This cost is however amortized over multiple check-in executions.

devices used their 802.11b/g Wi-Fi interfaces. All reported values are averages taken over at least 10 independent protocol runs.

We have first measured the overhead of the *Setup* operation. If d is the number of profile dimensions, N is the Benaloh modulus size and b the sub-range count of domain D , the computation overhead of Setup is $T_{Setup} = T_{keysig} + dbT_E + T_{TSS}$. T_{keysig} is the time to generate the signature key, T_E is the average time of Benaloh encryption and T_{TSS} is the time to initialize the TSS (i.e., random polynomial generation). The storage overhead of *Setup* is $Store_{Setup} = dbN$.

We set the b to be 10, Shamir’s TSS group size to 1024 bits and RSA’s modulus size to 1024 bits. Figure 6.6 shows the *Setup* overhead on the smartphone and laptop platforms, when the Benaloh modulus size ranges from 64 to 2048 bits. Note that even a resource constrained smartphone takes only 2.2s for 1024 bit sizes (0.9s on a laptop). A marked increase can be noticed for the smartphone when the Benaloh bit size is 2048 bit long - 13.5s. We note however that this cost is amortized over multiple check-in runs.

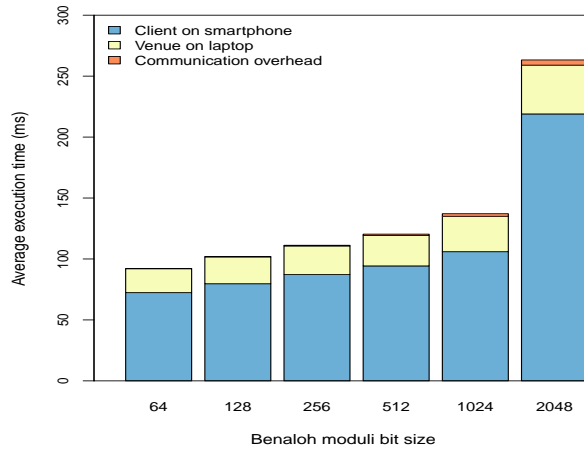


Figure 6.7: The overhead imposed by ZK-CTR as a function of the Benaloh modulus size. Note the significant overhead increase for a 2048-bit modulus, of approximately 260ms per ZK-CTR round.

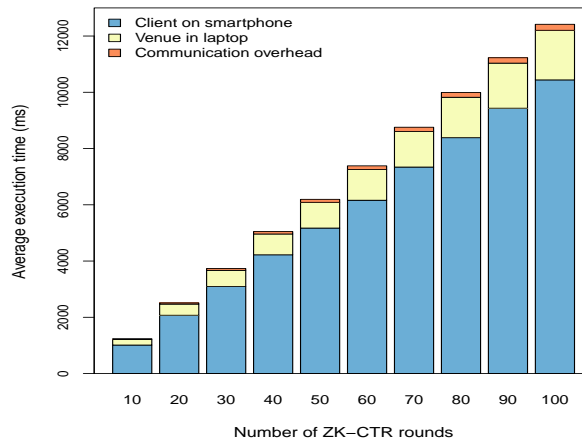


Figure 6.8: The overhead of the ZK-CTR protocol as a function of the number of proof rounds. The linear increase in the number of rounds leads to a 12s overhead for 100 rounds. 100 rounds reduce however the probability of client cheating to an insignificant value, 2^{-100} .

The computation overhead of *CheckIn* is $T_{CI} = bT_{RE} + T_{ZK}$, where T_{RE} is the Benaloh re-encryption cost and T_{ZK} is the overhead of the ZK-CTR protocol. The formula does not consider the cost of modular multiplication, random number generation and random permutation operations, that are negligible compared to the other costs. Given s , the number of rounds of ZK-CTR, $T_{ZK} = 2sbT_{RE} + sbT_{RE} + \frac{s}{2}bT_{RE} = \frac{7}{2}sbT_{RE}$. The communication overhead is $T_{com_CI} = bN + T_{com_ZK}$. The communication cost of ZK-CTR, T_{com_ZK} is $s(2bN + \frac{1}{2}4bN + \frac{1}{2}2bN) = 5sbN$.

We now focus on the most resource consuming component, the ZK-CTR protocol. While the above formulas assume similar capabilities for the client and venue components, we now measure the client side running on the smartphone and the venue component executing on the laptop. Figure 6.7 shows the dependence of the three costs for a single round of ZK-CTR on the Benaloh modulus size. Given the more efficient venue component and the superior computation capabilities of the laptop, the venue component has a much smaller overhead. We have set $b = 10$. The communication overhead is the smallest, exhibiting a linear increase with bit size. For a Benaloh key size of 1024 bits, the average end-to-end overhead of a single ZK-CTR round is 135ms. The venue component is 29ms and the client component is 106ms. Furthermore, Figure 6.8 shows the overheads of these components as a function of the number of ZK-CTR rounds, when the Benaloh key size is 1024 bit and $b = 10$. For 30 rounds, when a cheating client's probability of success is 2^{-30} (1 in a billion), the total overhead is 3.6s.

We further examine the communication overhead in terms of bits transferred during ZK-CTR between a client and a venue. The communication overhead in a single ZK-CTR round is $4bN + 3bN = 7bN$. The second component of the sum is due to the average outcome of the challenge bit. Figure 6.9 shows the dependency of the communication overhead (in KB) on b , when $N = 1024$. Even when $b = 20$,

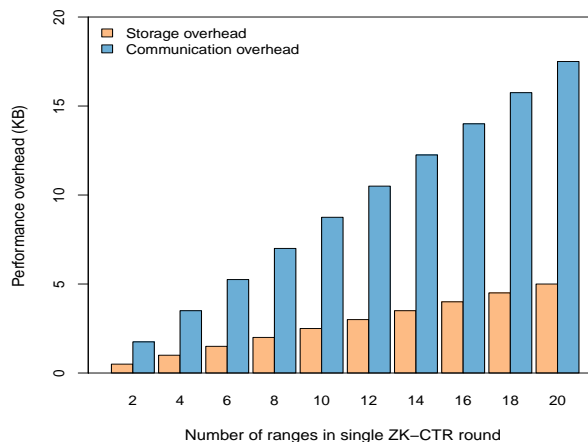


Figure 6.9: Storage and communication overhead (in KB) as a function of b , the number of sub-intervals considered in the statistics computation. Even for $b=20$, the storage overhead is only 5KB and the communication is 17KB.

the communication overhead is around 17KB. Figure 6.9 shows also the storage overhead (at a venue). The storage overhead is only a fraction of the (single round) communication overhead, $2bN$. For a single dimension, with 20 sub-ranges, the overhead is 5KB.

6.5.2 Evaluation Results for iSafe Browser Plugin Performance

Figure 6.5 shows the overhead of the iSafe plugin when collecting the reviews of a venue browsed by the user, as a function of the number of reviews the venue has. It includes the cost to request each review page, parse and process the data for transfer. It exhibits a sub-linear dependence on the number of reviews of the venue (under 1s for 10 reviews but under 30s for 4000 reviews), showing that Yelp’s delay for successive requests decreases. While even for 500 reviews the overhead is less than 5s, we note that this cost is incurred only once per venue. Subsequent accesses to the same venue, by any other user, no longer incur this overhead.

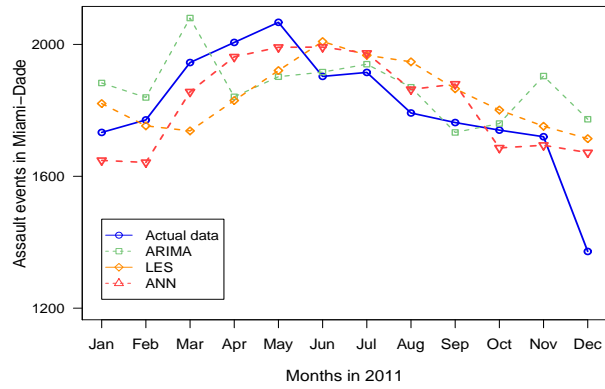
Model	Figure 5.a		Figure 5.b		Figure 5.c	
	RMSE	MAPE	RMSE	MAPE	RMSE	MAPE
ARIMA	158.80	6.42	38.77	7.08	1.27	43
LES	151.03	6.79	53.57	11.89	1.41	42.08
ANN	116.48	5.32	40.44	8.23	1.3	35.72

Table 6.2: Error measurement data for ARIMA, LES and ANN. Figures reference to the main document.

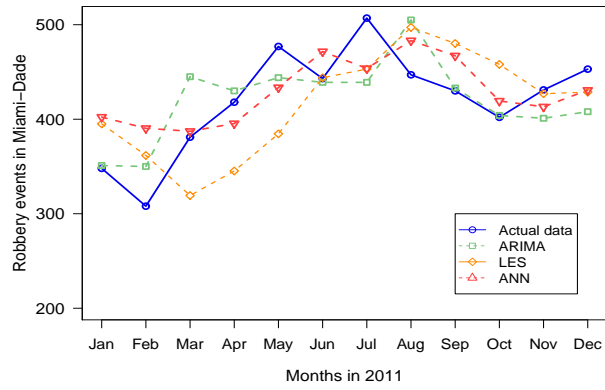
Forecasting Accuracy

We investigate here the accuracy of the time series forecasting techniques discussed in Section 5.2.4 in predicting the number of crimes to occur at a location during the near future. We used the R statistical software package [R D11] to generate the ARIMA model and MATLAB toolboxes [MAT10] for the LES and ANN models. In the following, we analyze separately three crime types: aggravated assault, robbery and larceny/theft that make up for more than 75% of the total amount of crimes. For ANN, we set the maximum lag to 12 (to cover the last 12 months/weeks in the lag structure), and the learning rate to 0.1. While a learning rate of 0.4 worked well, we set it to 0.1 to ensure convergence. The higher the learning rate, the faster the network is trained.

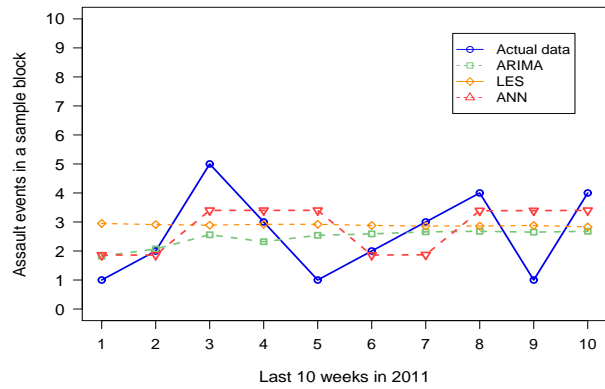
We used crime data recorded between 2007 and 2010 to predict per-month categorized event counts for the year 2011, for the Miami-Dade county. Figure 6.10(a) compares the predictions for the number of assaults made by ARIMA, LES and ANN against the recorded values. For ARIMA, we set $p=1$, $q=1$, $d=1$. Details for choosing the ARIMA parameters are provided in supplemental material. All three models correctly predict the downward trend from May until December, with ANN achieving a slightly better accuracy than LES and ARIMA. Figure 6.10(b) compares the predictions for the number of robberies. For ARIMA, we set $p=3$, $q=0$, $d=1$. All models accurately predict the initial increase followed by a slight decrease in the



(a)



(b)



(c)

Figure 6.10: Crime Forecasting experiments in Miami-Dade: (a) Prediction of assaults. (b) Prediction of robberies. (c) Prediction of assaults in a given block.

number of robberies. ARIMA and ANN outperform the LES model as confirmed by the RSME and MAPE values (see Table 6.2). ARIMA slightly outperforms ANN.

We further focus on finer grained spatial and temporal predictions: per-block, weekly events. For ANN, we partition the input data into 95 training vectors and 10 test vectors. Figure 6.10(c) compares the recorded data against the ARIMA, LES and ANN predictions of assault events in the last ten weeks of 2011, for one block in the Miami-Dade county. The ARIMA parameters are $p=1$, $q=1$, $d=0$.

Yelp Safety Profiles

We have collected public information from the accounts of 2025 Yelp users, all residents of the Miami-Dade county. The information collected for each user includes the number of reviews, the venues reviewed, existing check-ins at any venues, and the date when each review and check-in was recorded. We build the crime index, CI , value for each Census block from the Miami-Dade county in 2010. Figure 6.11(a) shows the cumulative distribution function of the CI values (Figure 6.3 shows their spatial distribution). It shows that for the Miami-Dade county, most blocks experience relatively low levels of crime per-capita: 50% of blocks have a CI value smaller than 0.0015 and only 5% of blocks have CI values exceeding 0.01.

Given the CI values of the blocks containing the venues visited (reviewed or subject of a check-in) by a yelper (Yelp user), we compute the user's crime index value, as defined by Equation 6.4, then the user's safety index: SI_U . Out of the 2025 collected yelpers, 1194 had written reviews in 2010. Figure 6.12 shows the distribution of the safety index values of these 1194 yelpers. It shows that most Miami-Dade county yelpers are safe: all have a safety index value larger than 0.96 (1 is the maximum value), with 90% of them exceeding 0.99.

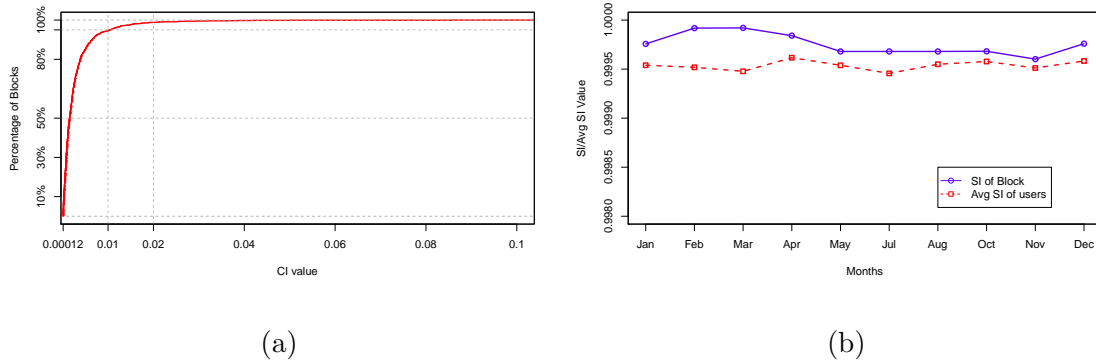


Figure 6.11: (a) Distribution of block crime index values in the Miami-Dade county. (b) Evolution in time of the SI value of a Miami-Dade block and the average SI values of Yelp users that visited the block.

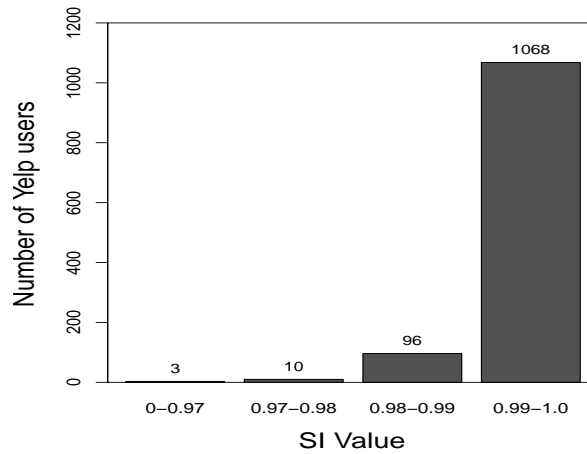


Figure 6.12: Distribution of safety index values of Yelp users.

We further compare the evolution in time of the safety index SI_B of a block B with the average safety index values over the Yelp users that visited B (and left feedback). To this end, based on the crime database, for each month we calculate the SI value of each block in the Miami-Dade county. We then compute the monthly average of safety index values of yelpers that reviewed venues within B (during the month). Figure 6.11(b) shows the monthly evolution of the SI_B value of a Miami-Dade block and the average safety index value of the Yelp users that visited the

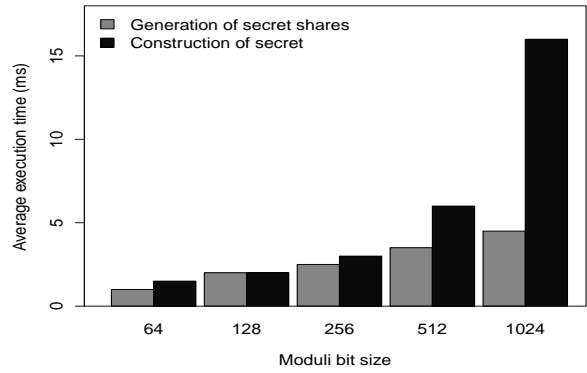
block during 2010. For this block, the two metrics have similar values. This shows that an average of the safety indexes of the block's visitors can be used to replace a crime-based safety index for the block.

Android iSafe Evaluation

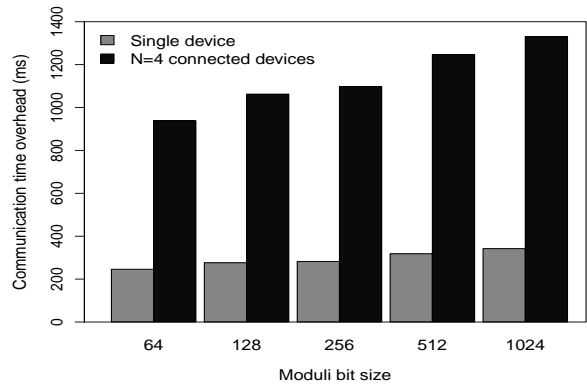
We have created a testbed consisting of 4 Android smartphones: Samsung Admire (OS: Gingerbread 2.3.4), HTC Aria (OS: Eclair 2.1), Sony E10i (OS: Eclair 2.1) and Samsung GALAXY S II (OS: Gingerbread 2.3.4). For single device testing, we used the Samsung Admire smartphone with a 800MHz CPU. Thus, we set the $NThr$ value to 3 and the number of secret shares to 4. In the following, all reported values are averages over at least 10 independent protocol runs.

We have first measured the overhead of the secret share generation and reconstruction operation. Figure 6.13(a) shows the overhead on the smartphone, when the modulus size ranges from 64 to 1024 bits. Note that even a resource constrained smartphone takes only 4.5 ms and 16 ms for secret splitting and reconstruction even for 1024 bit long moduli.

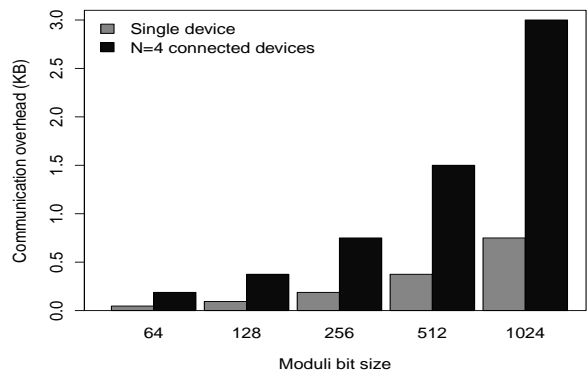
Furthermore, we focus on the time and space communication overhead for a single device as well as for the 4 connected devices in our testbed. Figure 6.13(b) shows the dependence of the communication time on the modulus bit size. Even for a modulus size of 1024 bits, the average end-to-end communication overhead of a single device is 342ms and 1.3s of our whole system. Figure 6.13(c) shows the dependency of the communication overhead (in KB) on the modulus size ranging from 64 to 1024 bits, for a single device and for the whole system of 4 connected devices. Even for 1024 bit moduli, the total communication overhead is around 3KB.



(a)



(b)



(c)

Figure 6.13: Android iSafe overhead. (a) Secret share generation and secret reconstruction time overhead. (b) iSafe communication overhead. (c) iSafe total communication size.

6.6 Limitations

Our holistic approach toward evaluating the safety of a user was evaluated based on crime and census data from the Miami-Dade (FL) county. We proposed to assign each crime type a weight proportional to its seriousness, defined according to the criminal punishment code, i.e., the Florida Criminal Punishment Code (FCPC) [oC] which may be different state-wise. To make our approach more generalized, we need to overcome this limitation.

6.7 Summary

At first, we proposed Profil_R , a framework and mechanisms for privately and correctly building location-centric profiles. We have proved the ability of our solutions to satisfy the privacy and correctness requirements and shown that Profil_R is efficient, even when executed on resource constrained mobile devices. We then introduced one major application for Profil_R : smart and safe cities by proposing several techniques for evaluating the safety of users based on their spatial and temporal dimensions and crime data. We have shown that data collected by geosocial networks bears relations with crimes. Our Android and browser plugin implementations show that our approach is efficient both in terms of the computation and the communication overheads.

CHAPTER 7

CONCLUSION AND FUTURE WORK

Online social networks are central to numerous aspects of people’s daily online and physical activities. People rely on online reviews to make decisions on purchases, services and opinions, among others. Unfortunately, online social networks have also become an attractive target for malicious behaviors, mainly due to the popularity and influence of different forms of media data in people’s lives. We have demonstrated that our proposed solutions can be used to significantly enhance the data authentication capability of some important products in online social networks. In this dissertation, the problem of authenticating different forms of data (e.g. media, reviews, sensor data) present in online social networks has been discussed. Specifically, four related but orthogonal concrete problems have been studied: 1) the fraudulent behavior detection problem in review centric social networks and social app markets; 2) the video liveness verification problem to verify the authenticity of a video captured in mobile devices; 3) Secure data storage and communication problem to protect low power wearable fitness trackers from security vulnerabilities; and 4) the conflict between profit and privacy in geosocial networks. We have demonstrated that the proposed solutions can be used to address these security challenges in online social networks.

In chapter 2, we propose, develop and evaluate a system, Marco that exploits the unique combination of social, spatial and temporal signals gleaned from Yelp, to detect venues whose ratings are impacted by fraudulent reviews. Our approach increases the cost and complexity of attacks, by imposing a tradeoff on fraudsters, between their ability to impact venue ratings and their ability to remain undetected. We demonstrate that Marco is effective and fast; its classification accuracy is up to 94% for reviews, and 95.8% for venues while outperforming the state-of-the-

art approaches by a large margin (around 20%). It flags 244 of the 7,435 venues analyzed as deceptive; manual inspection revealed that they were indeed suspicious. Furthermore, we use Marco to evaluate the impact of Yelp events, organized for elite reviewers, on the hosting venues and we show that twice as many hosting venues experience a significant rating boost rather than a negative impact. We have also investigated the fraudulent behaviors in Google's Android app market. We propose and develop a system, FairPlay that combines relational, behavioral and linguistic indicators as well as longitudinal app data to identify both malware and apps involved in search rank fraud. FairPlay's accuracy in classifying gold standard datasets of malware, fraudulent and legitimate apps we collected from Google Play, exceeds 95%. Our results show that 75% of the identified malware apps have also engaged in search rank fraud. In addition, we discovered tens of apps involved in a novel fraud technique, that coerce their users to participate in search rank fraud. In future work we intend to explore other domains of online social networks like bidding sites, freelance job sites etc. to identify any sorts of deceptive behaviors that try to manipulate the system. In google Play, we discovered a few (tens of apps) coercive apps using a keyword search and manually investigating the search results. In future, we plan to investigate more in this direction with the help of machine learning tools. We have devised solutions to detect fraudulent reviews, deceptive apps and malwares. Our future work will target to identify deceptive developers who may use fraud on many of their developed apps, and may even reuse fraudsters for the jobs on their various apps.

In chapter 3, we deal with the fundamental question of whether the visual stream uploaded by a user has been captured live on a mobile device, and has not been tampered with by a malicious user attempting to game the system. We propose, develop and evaluate a system, Movee that relies on the accelerometer sensors ubiquitously

deployed on most recent mobile devices to verify the liveness of a simultaneously captured video stream. We develop strong attacks both by utilizing fully automated attackers and by employing trained human experts for creating fraudulent videos to thwart mobile video verification systems. We have implemented Movee on both Android and Google Glass devices, and through extensive experiments, we have shown that (i) it is efficient in differentiating fraudulent and genuine videos (Movee’s accuracy ranges between 68-93% on a smartphone, and between 76-91% on a Google Glass device.) and (ii) imposes reasonable overheads on the server. We also introduce the concept of video motion categories to annotate the camera and user motion characteristics of arbitrary videos. We share motion annotations of YouTube citizen journalism videos and of free-form video samples that we collected through a user study. We observe that the performance of our approach differs across video motion categories which is a very important finding. In future work we intend to integrate more sensors (e.g., gyroscope), as well as the use of MonoSLAM [DRMS07] as an alternative VMA implementation to improve accuracy. We also intend to integrate more user motion types, for example, jumping, driving etc. and more video motion types in our video classification. We have introduced and evaluated Vamos against manual, automatic and mixed attacks. We leave the exhaustive exploration of the attack space for future work.

In chapter 4, we deal with another challenge of securing low power wearable sensor devices during communication and data storage in social sensor networks. We devise SensCrypt, a lightweight protocol for secure data storage and communication, for use by makers of affordable and lightweight personal trackers. To prove the existence of security vulnerabilities present in popular wearable devices from Fitbit and Garmin, we build attack tools that exploit vulnerabilities and demonstrated several inspection and injection attacks. SensCrypt protects not only against inspect

and inject attacks, but also against attackers that physically capture and read the memory of trackers. SensCrypt’s hardware and computation requirements are minimal, just enough to perform low-cost symmetric key encryption and cryptographic hashes. SensCrypt does not impose storage overhead on trackers and ensures an even wear of the tracker storage, extending the life of flash memories with limited program/erase cycles. We have also implemented Sens.io, a tracker platform, of similar capabilities with existing popular solutions but at a fraction of the cost and we show that SensCrypt running on Sens.io is very efficient. SensCrypt is applicable to a range of sensor based platforms, that includes a large number of popular fitness and home monitoring solutions. While SensCrypt’s defenses may not be immediately adopted by existing products, our work provides a foundation upon which to create, implement and test new defensive mechanisms for future tracker designs.

In chapter 5, we take first steps toward addressing the conflict of profit and privacy in geosocial networks. We propose a framework, Profil_R that allows the construction of LCPs based on the profiles of present users, while ensuring the privacy and correctness of participants. We devise both a venue centric and a decentralized solution and prove that Profil_R satisfies the proposed privacy and correctness properties. We have shown that Profil_R is efficient: the end-to-end overhead is small when executed even on resource constrained mobile devices even under strong privacy and correctness assurances. We have also proposed a holistic approach toward evaluating the safety of a user, that combines the predicted safety of the user’s location with the aggregated safety of the people co-located with the user. Our Android and browser plugin implementations show that our approach is efficient both in terms of the computation and the communication overheads. In future work we will develop solutions for detecting and eliminating fraudulent information from data sources, including reviews and check-ins. Furthermore, we will integrate safety information

in other user experiences, including navigation directions and mobile authentication solutions.

BIBLIOGRAPHY

- [AG97] Yali Amit and Donald Geman. Shape quantization and recognition with randomized trees. *Neural Comput.*, 9(7):1545–1588, October 1997.
- [AM12] Michael Anderson and Jeremy Magruder. Learning from the crowd: Regression discontinuity estimates of the effects of an online review database. *Economic Journal*, 122(563):957–989, 2012.
- [Ama] Amazon. www.amazon.com.
- [ANCT09] Ghazi Al-Naymat, Sanjay Chawla, and Javid Taheri. Sparsedtw: a novel approach to speed up dynamic time warping. In *Proceedings of the Eighth Australasian Data Mining Conference-Volume 101*, pages 117–127. Australian Computer Society, Inc., 2009.
- [ANT] ANT-FS and FIT. <http://www.thisisant.com/developer/ant/ant-fs-and-fit>.
- [App] iOS Security. https://www.apple.com/ipad/business/docs/iOS_Security_Feb14.pdf.
- [Arda] Arduino Guide. <http://arduino.cc/en/Guide/Introduction>.
- [ardb] Arduino Uno. <http://arduino.cc/en/Main/arduinoBoardUno>.
- [Arl07] Sylvain Arlot. Model selection by resampling penalization, 2007.
- [Arx] Mobile Application Protection. <https://www.arxan.com/products/application-protection/mobile/>.
- [AT06] Ankur Agarwal and Bill Triggs. Recovering 3d human pose from monocular images. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 28(1):44–58, 2006.
- [ATPD10] Golnaz Abdollahian, Cüneyt M. Taskiran, Zygmunt Pizlo, and Edward J. Delp. Camera motion-based analysis of user generated video. *IEEE Transactions on Multimedia*, 12(1):28–41, 2010.
- [Att] A.G. Schneiderman Announces Agreement With 19 Companies To Stop Writing Fake Online Reviews And Pay More Than \$350,000

- In Fines. www.ag.ny.gov/press-release/ag-schneiderman-announces-agreement-19-companies-stop-writing-fake-online-reviews-and.
- [Ban] Global Specialties R680 C-Programmable Banshi Robotic Arm. <http://www.testequipmentdepot.com/products.htm?item=R680&ref=gbase&gclid=CNn0rqi2zsICFe7m7AodMS8A8A>.
- [Bas] Basis B1. <http://www.mybasis.com/>.
- [BBC] British Broadcasting Corporation. <http://www.bbc.com/>.
- [BBFN10] Petra Berenbrink, André Brinkmann, Tom Friedetzky, and Lars Nagel. Balls into bins with related random choices. In *Proceedings of the Symposium on Parallelism in Algorithms and Architectures (SPAA)*, 2010.
- [BCK96] Mihir Bellare, Ran Canetti, and Hugo Krawczyk. Keying hash functions for message authentication. In *Proceedings of the 16th Annual International Cryptology Conference on Advances in Cryptology, CRYPTO '96*, pages 1–15, 1996.
- [BDH⁺11] Ingo Bente, Gabi Dreo, Bastian Hellmann, Stephan Heuser, Joerg Vieweg, Josef von Helden, and Johannes Westhuis. Towards permission-based attestation for the android platform. In *Trust and Trustworthy Computing*, pages 108–115. Springer, 2011.
- [Bes15] Best App Promotion. <http://www.bestreviewapp.com/>, Last accessed on April 2015.
- [Bis95] Christopher M. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, Inc., New York, NY, USA, 1995.
- [BKL09] Steven Bird, Ewan Klein, and Edward Loper. *Natural Language Processing with Python*. O’Reilly, 2009.
- [BKM10] Johannes Barnickel, Hakan Karahan, and Ulrike Meyer. Security and privacy for mobile electronic health monitoring and recording systems. In *Proceedings of the IEEE International Symposium on A World of Wireless, Mobile and Multimedia Networks (WoWMoM)*, pages 1–6, 2010.

- [BMD] Jawbone takes a big bite out of health tech: acquires BodyMedia, launches Up app platform. <http://venturebeat.com/2013/04/30/jawbone-takes-a-big-bite-out-of-health-tech-acquires-bodymedia-launches-up-app-platform>.
- [BoA14] Deposit checks easily and securely with Mobile Check Deposit. <http://promo.bankofamerica.com/mobile-check-deposit/>, Last retrieved on July 5, 2014.
- [Bod] Body Media. <http://www.bodymedia.com/>.
- [Bre96] Leo Breiman. Bagging predictors. *Mach. Learn.*, 24(2):123–140, August 1996.
- [Bre01] Leo Breiman. Random forests. *Machine Learning*, 45:5–32, 2001.
- [Bre06] Ing Breeuwsma. Forensic imaging of embedded systems using JTAG (boundary-scan). *Digital Investigation*, 3, 2006.
- [BSV98] N. Barberis, A. Shleifer, and R Vishny. A model of investor sentiment. *Journal of Financial Economics*, 49:307–243, 1998.
- [BYS07] Alessandro Bissacco, Ming-Hsuan Yang, and Stefano Soatto. Fast human pose estimation using appearance and motion via multi-dimensional boosting regression. In *Computer Vision and Pattern Recognition, 2007. CVPR'07. IEEE Conference on*, pages 1–8. IEEE, 2007.
- [BZNT11] Iker Burguera, Urko Zurutuza, and Simin Nadjm-Tehrani. Crow-droid: behavior-based malware detection system for android. In *Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices*, pages 15–26. ACM, 2011.
- [cam] Android Camera API. <http://developer.android.com/reference/android/hardware/Camera.html>.
- [CDBN09] A. Caragliu, C. Del Bo, and P. Nijkamp. Smart cities in europe. Serie Research Memoranda 0048, VU University Amsterdam, Faculty of Economics, Business Administration and Econometrics, 2009.

- [Cen10] United States Census. 2010 census. <http://2010.census.gov/2010census/>, 2010.
- [Cha82] David Chaum. Blind signatures for untraceable payments. In *Advances in Cryptology: Proceedings of CRYPTO '82*, pages 199–203, 1982.
- [Chely] G. Chetty. Biometric liveness detection based on cross modal fusion. In *Information Fusion, 2009. FUSION '09. 12th International Conference on*, pages 2255–2262, July.
- [Cit] Citizen Evidence Lab. <http://citizenevidence.org/>.
- [CJHP10] Shu-Chuan Chu, Lakhmi C Jain, Hsiang-Cheh Huang, and Jeng-Shyang Pan. Error-resilient triple-watermarking with multiple description coding. *Journal of Networks*, 5(3):267–274, 2010.
- [CMS09] Antonio Cutillo, Refik Molva, and Thorsten Strufe. Safebook: Feasibility of transitive cooperation for privacy on a decentralized social network. In *IEEE WOWMOM*, pages 1–6, 2009.
- [CNN] Cable News Network. www.cnn.com.
- [CNW⁺11] D. H. Chau, C. Nachenberg, J. Wilhelm, A. Wright, and C. Faloutsos. Polonium: Tera-scale graph mining and inference for malware detection. In *Proceedings of the SIAM SDM*, 2011.
- [Col09] Gerald Coley. Beagleboard system reference manual. *BeagleBoard.org*, December, 2009.
- [Coo93] I.D. Coope. Circle fitting by linear and nonlinear least squares. *Journal of Optimization Theory and Applications*, 76:381–388, 1993.
- [CP12] Bogdan Carbunar and Rahul Potharaju. You unlocked the Mt. Everest Badge on Foursquare! Countering Location Fraud in GeoSocial Networks. In *Proceedings of the 9th IEEE International Conference on Mobile Ad hoc and Sensor Systems (MASS)*, 2012.
- [CR05] S. Chainey and J. Ratcliffe. *GIS and Crime Mapping*. Wiley, 2005.

- [Cri] James Cridland. Mapping the riots. <http://james.cridland.net/blog/mapping-the-riots/>.
- [CRKH11] D. Christin, A. Reinhardt, S. Kanhere, and M. Hollick. A survey on privacy in mobile participatory sensing applications. *Journal of Systems and Software*, 84(11):1928 – 1946, 2011.
- [CRS02] S Chainey, S Reid, and N Stuart. *When is a Hotspot a Hotspot? A Procedure for Creating Statistically Robust Hotspot Maps of Crime*. Kidner, D and Higgs, G and White, S, 2002.
- [CS12] David Clifford and Glenn Stone. Variable penalty dynamic time warping code for aligning mass spectrometry chromatograms in r. *Journal of Statistical Software*, 47(8):1–17, April 2012.
- [CTU08] Spencer Chaineya, Lisa Tompson, and Sebastian Uhlig. The utility of hotspot mapping for predicting spatial patterns of crime. *Security Journal*, 21:4 – 28, 2008.
- [CW06] Girija Chetty and Michael Wagner. Multi-level liveness verification for face-voice biometric authentication. In *Biometric Symposium*, 2006.
- [DBC] Death By Captcha. www.deathbycaptcha.com/.
- [DCM87] E. De Castro and C. Morandi. Registration of translated and rotated images using finite fourier transforms. *IEEE Trans. Pattern Anal. Mach. Intell.*, 9(5):700–703, May 1987.
- [Dep] Miami-Dade Police Department. CrimeView Community. <http://crimemaps.miamidade.gov>.
- [DLP03] Kushal Dave, Steve Lawrence, and David M. Pennock. Mining the peanut gallery: Opinion extraction and semantic classification of product reviews. In *Proceedings of the 12th International Conference on World Wide Web, WWW '03*, pages 519–528, New York, NY, USA, 2003. ACM.
- [DM09] George Danezis and Prateek Mittal. Sybilinfer: Detecting sybil nodes using social networks. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, 2009.

- [DMS04] Roger Dingledine, Nick Mathewson, and Paul F. Syverson. Tor: The second-generation onion router. In *USENIX Security Symposium*, pages 303–320, 2004.
- [DRMS07] Andrew J. Davison, Ian D. Reid, Nicholas D. Molton, and Olivier Stasse. Monoslam: Real-time single camera slam. *IEEE Trans. Pattern Anal. Mach. Intell.*, 29(6):1052–1067, 2007.
- [dur] Duracell Product Data Sheets. ww2.duracell.com/media/en-US/pdf/gtcl/Product_Data_Sheet/NA_DATASHEETS/PC1604_US_PC.pdf.
- [Ear] Earndit: We reward you for exercising. <http://earndit.com/>.
- [eBa] eBay. <http://www.eBay.com>.
- [ECC⁺05] John E. Eck, Spencer Chainey, James G. Cameron, Michael Leitner, and Ronald E. Wilson. Mapping crime: Understanding hot spots. Special, U.S. Department of Justice, Office of Justice Program, National Institute of Justice, August 2005.
- [Est10] Deborah L. Estrin. Participatory sensing: applications and architecture. In *Proceedings of the 8th international conference on Mobile systems, applications, and services*, 2010.
- [FAo⁺10] Vasco Furtado, Leonardo Ayres, Marcos de Oliveira, Eurico Vasconcelos, Carlos Caminha, Johnatas DORleans, and Mairon Belchior. Collective intelligence in law enforcement the wikicrimes system. *Information Sciences*, 180(1):4 – 17, 2010.
- [FD00] R.W. Frischholz and U. Dieckmann. Bioid: A multimodal biometric identification system. *IEEE Computer*, 33(2):64–68, 2000.
- [FF09] Jean Baptiste Joseph Fourier and Alexander Freeman. *The Analytical Theory of Heat*. Cambridge University Press, 2009.
- [ffd] RATC free form dataset. <http://users.cis.fiu.edu/~mrahm004/RATC/>.
- [Fir05] Vo2 estimation method based on heart rate measurement. Technical report, Firstbeat Technologies Ltd, 2005.

- [Fit] Fitbit. <http://www.fitbit.com/>.
- [Fit13] Fitbit Specs. <http://www.fitbit.com/one/specs>, Last retrieved on October 1st, 2013.
- [Fiv] Fiverr. <https://www.fiverr.com/>.
- [FLE] Flelp. www.yelp.com/topic/miami-flelp-we-rock.
- [FM06] Zakia Ferdousi and Akira Maeda. Unsupervised outlier detection in time series data. In *Data Engineering Workshops, 2006. Proceedings. 22nd International Conference on*, pages x121–x121. IEEE, 2006.
- [FML⁺13] Geli Fei, Arjun Mukherjee, Bing Liu, Meichun Hsu, Malu Castellanos, and Riddhiman Ghosh. Exploiting Burstiness in Reviews for Review Spammer Detection. In *ICWSM*, 2013.
- [For] Garmin Forerunner. <http://sites.garmin.com/forerunner610/>.
- [fou] Foursquare. <https://foursquare.com/>.
- [Fow10] Geoffrey Fowler. App Watch: PayPal Lets You Cash Checks On Your Phone. <http://blogs.wsj.com/digits/2010/09/30/app-watch-paypal-lets-you-cash-checks-on-your-phone/>, September 2010.
- [Fre] Freelancer. <http://www.freelancer.com>.
- [FRVM⁺10] Dario Freni, Carmen Ruiz Vicente, Sergio Mascetti, Claudio Bettini, and Christian S. Jensen. Preserving location and absence privacy in geo-social networks. In *Proceedings of the 19th ACM CIKM '10*, pages 309–318, NY, USA, 2010. ACM.
- [FXGC12] Song Feng, Longfei Xing, Anupam Gogar, and Yejin Choi. Distributional footprints of deceptive product reviews. In *Proceedings of the 6th International Conference on Weblogs and Social Media (ICWSM)*, 2012.
- [Gal90] S. I. Gallant. Perceptron-based learning algorithms. *Trans. Neur. Netw.*, 1(2):179–191, June 1990.

- [Gas04] William I. Gasarch. A survey on private information retrieval (column: Computational complexity). *Bulletin of the EATCS*, 82:72–107, 2004.
- [GG03] Marco Gruteser and Dirk Grunwald. Anonymous usage of location-based services through spatial and temporal cloaking. In *Proceedings of MobiSys*, 2003.
- [GHW⁺10] Hongyu Gao, Jun Hu, Christo Wilson, Zhichun Li, Yan Chen, and Ben Y. Zhao. Detecting and characterizing social spam campaigns. In *Proceedings of the 10th annual conference on Internet measurement*, IMC '10, pages 35–47, 2010.
- [GKBM10] Minas Gjoka, Maciej Kurant, Carter T. Butts, and Athina Markopoulou. Walking in Facebook: A Case Study of Unbiased Sampling of OSNs. In *Proceedings of IEEE INFOCOM '10*, San Diego, CA, March 2010.
- [GO01] Gorr and A. Olligschlaeger. Crime hot spot forecasting: Modeling and comparative evaluation. Draft final report, U.S. Department of Justice, Office of Justice Program, National Institute of Justice, 2001.
- [Goo] Google Play. <https://play.google.com/>.
- [GPW⁺04] Nils Gura, Arun Patel, Arvinderpal Wander, Hans Eberle, and Sheueling Chang Shantz. Comparing Elliptic Curve Cryptography and RSA on 8-bit CPUs. In *Proceedings of Cryptographic Hardware and Embedded Systems (CHES)*, pages 119–132, 2004.
- [Gre14] Andy Greenberg. Malware Apps Spoof Android Market To Infect Phones. Forbes Security, <http://www.forbes.com/sites/andygreenberg/2011/06/21/malware-apps-spoof-android-market-to-infect-phones/>, 2014.
- [Gua] The Guardian. Uk riots: every verified incident. <http://www.guardian.co.uk/news/datablog/2011/aug/09/uk-riots-incident-listed-mapped>.
- [Gua14] Us intelligence officials working to establish authenticity of video of sotloff being killed, reportedly by the same fighter who murdered

- james foley. <http://www.theguardian.com/world/2014/sep/02/isis-video-steven-sotloff-beheading>, September 2014.
- [GZZ⁺12] Michael Grace, Yajin Zhou, Qiang Zhang, Shihong Zou, and Xuxian Jiang. Riskranker: scalable and accurate zero-day android malware detection. In *Proceedings of the 10th international conference on Mobile systems, applications, and services*, pages 281–294. ACM, 2012.
- [HA04] Victoria J Hodge and Jim Austin. A survey of outlier detection methodologies. *Artificial Intelligence Review*, 22(2):85–126, 2004.
- [HE04] A. J. Hulbert and P. L. Else. Basal Metabolic Rate: History, Composition, Regulation, and Usefulness. *Physiological and Biochemical Zoology*, 77(6):869–876, 2004.
- [HGH⁺08] Baik Hoh, Marco Gruteser, Ryan Herring, Jeff Ban, Dan Work, Juan-Carlos Herrera, Re Bayen, Murali Annavaram, and Quinn Jacobson. Virtual Trip Lines for Distributed Privacy-Preserving Traffic Monitoring. In *Proceedings of ACM MobiSys*, 2008.
- [HHBR⁺08] D. Halperin, T. Heydt-Benjamin, B. Ransford, S. Clark, B. Defend, W. Morgan, K. Fu, T. Kohno, and W. Maisel. Pacemakers and implantable cardiac defibrillators: Software radio attacks and zero-power defenses. In *Proceedings of IEEE Symposium on Security and Privacy*, pages 129–142, 2008.
- [Hid] Hide My Ass! Free Proxy and Privacy Tools. <http://www.hidemyass.com/>.
- [Hil87] Begnaud Francis Hildebrand. *Introduction to numerical analysis: 2nd edition*. Dover Publications, Inc., 1987.
- [HL04] Mingqing Hu and Bing Liu. Mining and summarizing customer reviews. In *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '04, pages 168–177, New York, NY, USA, 2004. ACM.
- [Hor] Richard Hornsby. Florida Criminal Penalty Chart. <http://www.richardhornsby.com/criminal/penalties/>.

- [HRWZ08] Guangming Hong, Ahmad Rahmati, Ye Wang, and Lin Zhong. Sensecoding: accelerometer-assisted motion estimation for efficient video encoding. *MM '08*, pages 749–752. ACM, 2008.
- [HT04] Kohsia S Huang and Mohan M Trivedi. Robust real-time detection, tracking, and pose estimation of faces in video streams. In *Pattern Recognition, 2004. ICPR 2004. Proceedings of the 17th International Conference on*, volume 3, pages 965–968. IEEE, 2004.
- [IBM] IBM. Ibm smarter cities. http://www.ibm.com/smarterplanet/us/en/smarter_cities/overview/index.html.
- [IIS99] P. Indyk, G. Iyengar, , and N. Shivakumar. Finding pirated video sequences on the internet. Technical report, Stanford University, 1999.
- [Inf] InformaCam: Verified Mobile Media. <https://guardianproject.info/apps/informacam/>.
- [Jaw] Jawbone UP24. <https://jawbone.com/up>.
- [Jef99] E. Jefferis. A multi-method exploration of crime hot spot: A summary of findings. Technical report, U.S. Department of Justice, Office of Justice Program, National Institute of Justice, 1999.
- [JJ00] Markus Jakobsson and Ari Juels. Mix and match: Secure function evaluation via ciphertexts. In *Advances in Cryptology - ASIACRYPT 2000, 6th International Conference on the Theory and Application of Cryptology and Information Security*, pages 162–177, 2000.
- [JL08] Nitin Jindal and Bing Liu. Opinion spam and analysis. In *Proceedings of the international conference on Web search and web data mining, WSDM '08*, pages 219–230, New York, NY, USA, 2008. ACM.
- [JLL10] Nitin Jindal, Bing Liu, and Ee-Peng Lim. Finding unusual review patterns using unexpected rules. In *Proceedings of the 19th ACM international conference on Information and knowledge management, CIKM '10*, pages 1549–1552, 2010.
- [KD03] C. Karlof and D. Wagner, editors. *Secure Routing in Sensor Networks: Attacks and Countermeasures*, 2003.

- [KFB09] Klaus Kollreider, Hartwig Fronthaler, and Josef Bigün. Non-intrusive liveness detection by face images. *Image Vision Comput.*, 27(3):233–244, 2009.
- [Kic] Kickstarter. <http://www.kickstarter.com/>.
- [KKSM13] Arash Molavi Kakhki, Chloe Kliman-Silver, and Alan Mislove. Iolaus: Securing online content rating systems. In *Proceedings of the Twenty-Second International World Wide Web Conference (WWW'13)*, Rio de Janeiro, Brazil, May 2013.
- [Koh95a] Ron Kohavi. A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection. In *Proceedings of IJCAI*, 1995.
- [Koh95b] Ron Kohavi. A study of cross-validation and bootstrap for accuracy estimation and model selection. pages 1137–1143, 1995.
- [KW10] Balachander Krishnamurthy and Craig E. Wills. On the leakage of personally identifiable information via online social networks. *Computer Communication Review*, 40(1):112–117, 2010.
- [Lab] MIT Media Lab. Smart cities. <http://cities.media.mit.edu/>.
- [LHY⁺11] Fangtao Li, Minlie Huang, Yi Yang, Xiaoyan Zhu, and Xiaoyan Zhu. Learning to identify review spam. In *IJCAI*, pages 2488–2493, 2011.
- [LHYZ11] Fangtao Li, Minlie Huang, Yi Yang, and Xiaoyan Zhu. Learning to identify review spam. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence, IJCAI '11*, pages 2488–2493, 2011.
- [lib] Libfitbit: Library for accessing and transferring data from the fitbit health device. <https://github.com/qdot/libfitbit>.
- [LLLS14] Yulong Liu, Huaping Liu, Yunhui Liu, and Fuchun Sun. User-generated-video summarization using sparse modelling. In *Neural Networks (IJCNN), 2014 International Joint Conference on*, pages 3909–3915, July 2014.

- [LNJ⁺10] Ee-Peng Lim, Viet-An Nguyen, Nitin Jindal, Bing Liu, and Hady Wirawan Lauw. Detecting product review spammers using rating behaviors. In *Proceedings of the International Conference on Information and Knowledge Management (CIKM)*, pages 939–948, 2010.
- [LOCL10] S. Lim, T.H. Oh, Y. Choi, and T. Lakshman. Security issues on wireless body area network for remote healthcare monitoring. In *Proceedings of the IEEE International Conference on Sensor Networks, Ubiquitous, and Trustworthy Computing (SUTC)*, pages 327–332, 2010.
- [LRJ11] Chunxiao Li, A. Raghunathan, and N.K. Jha. Hijacking an insulin pump: Security attacks and defenses for a diabetes therapy system. In *Proceedings of the IEEE International Conference on e-Health Networking Applications and Services (Healthcom)*, 2011.
- [Luc] Michael Luca. Reviews, Reputation, and Revenue: The Case of Yelp.com. Available at hbswk.hbs.edu/item/6833.html.
- [MAT10] MATLAB. *version 7.10.0 (R2010a)*. The MathWorks Inc., Natick, Massachusetts, 2010.
- [MCT09] Erik Murphy-Chutorian and Mohan M Trivedi. Head pose estimation in computer vision: A survey. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 31(4):607–626, 2009.
- [MD07] Ramon Marti and Jaime Delgado. Security in a wireless mobile health care system, 2007.
- [MFB⁺11] Sergio Mascetti, Dario Freni, Claudio Bettini, X. Sean Wang, and Sushil Jajodia. Privacy in geo-social networks: proximity notification with untrusted service providers and curious buddies. *The VLDB Journal*, 20(4):541–566, August 2011.
- [MGKV06] Ashwin Machanavajjhala, Johannes Gehrke, Daniel Kifer, and Muthuramakrishnan Venkitasubramaniam. l-diversity: Privacy beyond k-anonymity. In *Proceedings of the 22nd International Conference on Data Engineering (ICDE)*, 2006.
- [Min14] Zach Miners. Report: Malware-infected Android apps spike in the Google Play store. PC-

- World, <http://www.pcworld.com/article/2099421/report-malwareinfected-android-apps-spike-in-the-google-play-store.html>, 2014.
- [MKL⁺13] Arjun Mukherjee, Abhinav Kumar, Bing Liu, Junhui Wang, Meichun Hsu, Malu Castellanos, and Riddhiman Ghosh. Spotting opinion spammers using behavioral footprints. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '13, pages 632–640, 2013.
- [Ml07] Meinard Mller. Dynamic time warping. In *Information Retrieval for Music and Motion*, pages 69–84. Springer Berlin Heidelberg, 2007.
- [MLG12] Arjun Mukherjee, Bing Liu, and Natalie Glance. Spotting fake reviewer groups in consumer reviews. In *Proceedings of the Int'l Conference on World Wide Web*, 2012.
- [Mlo14] Stephanie Mlot. Top Android App a Scam, Pulled From Google Play. PCMag, <http://www.pcmag.com/article2/0,2817,2456165,00.asp>, 2014.
- [MLSL] Fahad Moiz, Daniel Leon-Salas, and Yugyung Lee. A wearable motion tracker. *BodyNets '10*, pages 214–219.
- [MLV⁺ch] J. Mantyjarvi, M. Lindholm, E. Vildjiounaite, S.-M. Makela, and H.A. Ailisto. Identifying users of portable devices from gait pattern with accelerometers. In *ICASSP '05*, volume 2, pages ii/973–ii/976 Vol. 2, March.
- [MM06] Greg Mori and Jitendra Malik. Recovering 3d human body configurations using shape contexts. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 28(7):1052–1062, 2006.
- [MO08] Rajani Muraleedharan and Lisa Ann Osadciw. Secure health monitoring network against denial-of-service attacks using cognitive intelligence. In *Proceedings of the Communication Networks and Services Research Conference*, pages 165–170, 2008.
- [Mota] Mototola MotoActv. <http://www.motorola.com/us/MOTOACTV-16GB-Golf-Edition/121481.html>.

- [Motb] Sense: The meaning of life. <https://sen.se/store/mother/>.
- [MVL⁺13] Arjun Mukherjee, Vivek Venkataraman, Bing Liu, , and Natalie Glance. What yelp fake review filter might be doing. In *Proceedings of the International Conference on Weblogs and Social Media*, 2013.
- [Nes] Nest Thermostat. <https://nest.com/thermostat/life-with-nest-thermostat/>.
- [Nik] Nike+. nikeplus.com.
- [NKZS10] Mohammad Nauman, Sohail Khan, Xinwen Zhang, and Jean-Pierre Seifert. Beyond kernel-level integrity measurement: enabling remote attestation for the android platform. In *Trust and Trustworthy Computing*, pages 1–15. Springer, 2010.
- [NNMF06a] Alexandros Ntoulas, Marc Najork, Mark Manasse, and Dennis Fetterly. Detecting spam web pages through content analysis. In *Proceedings of the 15th international conference on World Wide Web, WWW '06*, pages 83–92, 2006.
- [NNMF06b] Alexandros Ntoulas, Marc Najork, Mark Manasse, and Dennis Fetterly. Detecting spam web pages through content analysis. In *Proceedings of the 15th International Conference on World Wide Web, WWW '06*, pages 83–92, New York, NY, USA, 2006. ACM.
- [NSSA04] J. Newsome, E. Shi, D. Song, and A. Perrig. The sybil attack in sensor networks: Analysis and defenses. In *Third International Symposium on Information Processing in Sensor Networks (IPSN)*, 2004.
- [NZD⁺14] Muhammad Naveed, Xiaoyong Zhou, Soteris Demetriou, XiaoFeng Wang, and Carl A Gunter. Inside job: Understanding and mitigating the threat of external device mis-bonding on android. In *Proceedings of ISOC Network and Distributed Computing Security (NDSS)*, 2014.
- [oC] Florida Department of Corrections. Florida Criminal Punishment Code. http://www.dc.state.fl.us/pub/sen_cpcm/cpc_manual.pdf.
- [OCCH11] Myle Ott, Yejin Choi, Claire Cardie, and Jeffrey T. Hancock. Finding deceptive opinion spam by any stretch of the imagination. In *Proceedings of the 49th Human Language Technologies (HLT)*, 2011.

- [OM12] Jon Oberheide and Charlie Miller. Dissecting the android bouncer. *SummerCon2012, New York*, 2012.
- [Ope] Open Source Computer Vision. <http://opencv.org/>.
- [opt] Optical mouse. https://en.wikipedia.org/wiki/Optical_mouse.
- [OTGH10] Femi G. Olumofin, Piotr K. Tysowski, Ian Goldberg, and Urs Hengartner. Achieving Efficient Query Privacy for Location Based Services. In *Privacy Enhancing Technologies*, pages 93–110, 2010.
- [Pat10] Z. Patton. Sensors make cities smarter. <http://www.governing.com/topics/public-justice-safety/Sensors-Make-Cities-Smarter.html>, April 2010.
- [Pay13] Pay Per Post. <https://payperpost.com/>, Last accessed October 12, 2013.
- [PCWF07] Shashank Pandit, Duen Horng Chau, Samuel Wang, and Christos Faloutsos. Netprobe: a fast and scalable system for fraud detection in online auction networks. In *Proceedings of the International Conference on World Wide Web*, pages 201–210. ACM, 2007.
- [PGS⁺12] Hao Peng, Chris Gates, Bhaskar Sarma, Ninghui Li, Yuan Qi, Rahul Potharaju, Cristina Nita-Rotaru, and Ian Molloy. Using probabilistic generative models for ranking risks of android apps. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 241–252. ACM, 2012.
- [Ple] Please Rob Me. <http://www.http://pleaserobme.com/>.
- [PLV02] Bo Pang, Lillian Lee, and Shivakumar Vaithyanathan. Thumbs Up? Sentiment Classification Using Machine Learning Techniques. In *Proceedings of EMNLP*, 2002.
- [PMX09] Xiao Pan, Xiaofeng Meng, and Jianliang Xu. Distortion-based anonymity for continuous queries in location-based mobile services. In *GIS*, pages 256–265, 2009.
- [Pos13] Posting Positive Reviews. postingpositivereviews.blogspot.com/, Last accessed October 12, 2013.

- [PWM07] Gyu-tae Park, Haitao Wang, and Young-su Moon. Liveness detection method and apparatus of video image, August 2007.
- [Py105] Timo Pylvänäinen. Accelerometer based gesture recognition using continuous hmms. *IbPRIA'05*, pages 639–646. Springer-Verlag, 2005.
- [PZ10] Krishna P. N. Puttaswamy and Ben Y. Zhao. Preserving privacy in location-based mobile social applications. In *Proceedings of the Eleventh Workshop on Mobile Computing Systems & Applications*, HotMobile '10, pages 1–6, New York, NY, USA, 2010. ACM.
- [R D11] R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2011. ISBN 3-900051-07-0.
- [RAPK06] Jim Rodgers, Dragomir Anguelov, Hoi-Cheung Pang, and Daphne Koller. Object pose detection in range scan data. In *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*, volume 2, pages 2445–2452. IEEE, 2006.
- [Ras] Raspberry Pi. An ARM GNU/Linux box for \$25. Take a byte! <http://www.raspberrypi.org/>.
- [RB09] Bobby Bhattacharjee Randy Baden, Neil Spring. Identifying close friends on the internet. In *Hotnets*, 2009.
- [RCHBC09] K. B. Rasmussen, C. Castelluccia, T. S. Heydt-Benjamin, and S. Capkun. Proximity-based access control for implantable medical devices. In *ACM Conference on Computer and Communications Security*, 2009.
- [Reva] 17 U.S. Code ?? 1201 - Circumvention of copyright protection systems. <https://www.law.cornell.edu/uscode/text/17/1201>.
- [Revb] 3 Tips for Spotting Fake Product Reviews - From Someone Who Wrote Them. MoneyTaksNews, www.moneytalksnews.com/2011/07/25/3-tips-for-spotting-fake-product-reviews—from-someone-who-wrote-them.
- [RG03] Henrik Rehbinder and Bijoy K Ghosh. Pose estimation using line-based dynamic vision and inertial sensors. *Automatic Control, IEEE Transactions on*, 48(2):186–199, 2003.

- [RMMR11] Kiran K. Rachuri, Cecilia Mascolo, Mirco Musolesi, and Peter J. Rentfrow. Sociablesense: Exploring the trade-offs of adaptive sampling and computation offloading for social sensing. In *Proceedings of the 17th Annual International Conference on Mobile Computing and Networking, MobiCom '11*, pages 73–84, New York, NY, USA, 2011. ACM.
- [Rob] RobotGeek Snapper Robotic Arm. <http://www.trossenrobotics.com/robotgeek-snapper-robotic-arm?feed=Froogle&gclid=CN00ps22zsICFcxQ7Aod0GUArw>.
- [Rob15] Daniel Roberts. How to spot fake apps on the Google Play store. Fortune, <http://fortune.com/2015/07/08/google-play-fake-app/>, 2015.
- [ROC] Receiver Operating Characteristic (ROC). Wikipedia, http://en.wikipedia.org/wiki/Receiver_operating_characteristic.
- [Roo] Root and Me. <https://play.google.com/store/apps/details?id=com.iamjake.root&hl=en>.
- [ROUMdR08] Grégory Rogez, Carlos Orrite-Uruñuela, and Jesús Martínez-del Rincón. A spatio-temporal 2d-models framework for human pose recovery in monocular sequences. *Pattern Recognition*, 41(9):2926–2944, 2008.
- [RRR⁺08] Grégory Rogez, Jonathan Rihan, Srikumar Ramalingam, Carlos Orrite, and Philip HS Torr. Randomized trees for human pose detection. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1–8. IEEE, 2008.
- [RS98] Martin Raab and Angelika Steger. Balls into Bins” - A Simple and Tight Analysis. In *Proceedings of Randomization and Approximation Techniques in Computer Science (RANDOM)*, pages 159–170, 1998.
- [RTC13] Mahmudur Rahman, Umut Topkara, and Bogdan Carbunar. Seeing is not believing: Visual verifications through liveness analysis using mobile devices. In *Proceedings of the 29th Annual Computer Security Applications Conference*, 2013.

- [SC07] Stan Salvador and Philip Chan. Toward accurate dynamic time warping in linear time and space. *Intelligent Data Analysis*, 11(5):561–580, 2007.
- [Seg11] David Segal. A Rave, a Pan, or Just a Fake? the New York Times, www.nytimes.com/2011/05/22/your-money/22haggler.html, 2011.
- [Sen] Sensor Delay. <http://developer.android.com/reference/android/hardware/SensorManager.html>.
- [SF] Emily Steel and Geoffrey Fowler. Facebook in privacy breach. <http://online.wsj.com/article/SB10001424052702304772804575558484075236968.html>.
- [SFE10] A. Shabtai, Y. Fledel, and Y. Elovici. Securing android-powered mobile devices using selinux. *Security Privacy, IEEE*, 8(3):36–44, 2010.
- [She12] Tracey Shelton. The most disturbing fake videos making the rounds in Syria. <http://www.globalpost.com/dispatch/news/regions/middle-east/syria/121109/fake-syria-videos-images>, November 2012.
- [Sie14] Ezra Siegel. Fake Reviews in Google Play and Apple App Store. Apptentive, <http://www.apptentive.com/blog/fake-reviews-google-play-apple-app-store/>, 2014.
- [SIG01] Bluetooth SIG. Specification of the bluetooth system, 2001.
- [Six11] Jeff Six. *Application Security for the Android Platform: Processes, Permissions, and Other Safeguards*. O’Reilly, 2011.
- [SK12] Justin Sahs and Latifur Khan. A machine learning approach to android malware detection. In *Intelligence and Security Informatics Conference (EISIC), 2012 European*, pages 141–147. IEEE, 2012.
- [SKE⁺12] Asaf Shabtai, Uri Kanonov, Yuval Elovici, Chanan Glezer, and Yael Weiss. andromaly: a behavioral malware detection framework for android devices. *Journal of Intelligent Information Systems*, 38(1):161–190, 2012.

- [SLG⁺12] Bhaskar Pratim Sarma, Ninghui Li, Chris Gates, Rahul Potharaju, Cristina Nita-Rotaru, and Ian Molloy. Android permissions: a perspective combining risks and benefits. In *Proceedings of the 17th ACM symposium on Access Control Models and Technologies*, pages 13–22. ACM, 2012.
- [Smi11] Julius O. Smith. *Spectral Audio Signal Processing*. W3K Publishing, 2011. online book.
- [SPE] Spelp. www.yelp.com/topic/boston-spelp-9.
- [Spo13] Sponsored Reviews. www.sponsoredreviews.com/, Last accessed October 12, 2013.
- [SSL⁺13] Borja Sanz, Igor Santos, Carlos Laorden, Xabier Ugarte-Pedrero, Pablo Garcia Bringas, and Gonzalo Álvarez. Puma: Permission usage to detect malware in android. In *International Joint Conference CISIS12-ICEUTE' 12-SOCO' 12 Special Sessions*, pages 289–298. Springer, 2013.
- [Sta10] StatSoft. Finding the right number of clusters in k-means and em clustering: v-fold cross-validation. In *Electronic Statistics Textbook*. Tulsa, OK, 2010.
- [SVD03] Gregory Shakhnarovich, Paul Viola, and Trevor Darrell. Fast pose estimation with parameter-sensitive hashing. In *Computer Vision, 2003. Proceedings. Ninth IEEE International Conference on*, pages 750–757. IEEE, 2003.
- [TB04] Norman Poh Hoon Thian and Samy Bengio. Evidences of equal error rate reduction in biometric authentication fusion, 2004.
- [TBGE10] A. Thiagarajan, J. Biagioni, T. Gerlich, and J. Eriksson. Cooperative transit tracking using smart-phones. In *8th ACM Conference on Embedded Networked Sensor Systems*, pages 85–98, 2010.
- [TCX] Training Center XML (TCX). <http://developer.garmin.com/schemas/tcx/v2/>.
- [TD00] A. C. Tamhane and D. D Dunlop. *Statistics and data analysis: From elementary to intermediate*. Upper Saddle River, NJ: Prentice Hall, 2000.

- [Ter] Terraflly Project. Crimes and Incidents Reported by Miami-Dade County and Municipal Police Departments. http://vn4.cs.fiu.edu/cgi-bin/arquery.cgi?lat=25.81&long=-80.12&category=crime_dade.
- [THH⁺09] Brian Thompson, Stuart Haber, William G. Horne, Tomas Sander, and Danfeng Yao. Privacy-preserving computation and verification of aggregate queries on outsourced databases. In *Proceedings of the 9th International Symposium on Privacy Enhancing Technologies*, PETS '09, pages 185–201, Berlin, Heidelberg, 2009. Springer-Verlag.
- [TKS13] Kota Tsubouchi, Ryoma Kawajiri, and Masamichi Shimosaka. Working-relationship detection from fitbit sensor data. In *Proceedings of the 2013 ACM conference on Pervasive and ubiquitous computing adjunct publication*, UbiComp '13 Adjunct, pages 115–118, 2013.
- [TML09] Nguyen Tran, Bonan Min, Jinyang Li, and Lakshminarayanan Subramanian. Sybil-resilient online content voting. In *NSDI'09: Proceedings of the 6th USENIX symposium on Networked systems design and implementation*, pages 15–28, Berkeley, CA, USA, 2009. USENIX Association.
- [TNB⁺10] Vincent Toubiana, Arvind Narayanan, Dan Boneh, Helen Nissenbaum, and Solon Barocas. Adnostic: Privacy preserving targeted advertising. In *Proceedings of the Network and Distributed Systems Security (NDSS) Symposium*, 2010.
- [TSGW09] A. Tootoonchian, S. Saroiu, Y. Ganjali, and A. Wolman. Lockr: Better Privacy for Social Networks. In *Proc. of ACM CoNEXT*, 2009.
- [UCL] Urban Sensing CENS UCLA. Walkability project. <http://urban.cens.ucla.edu/projects/walkability/>.
- [ULGW12] Osman Ugus, Martin Landsmann, Dennis Gessner, and Dirk Westhoff. A smartphone security architecture for app verification and process authentication. In *Computer Communications and Networks (ICCCN), 2012 21st International Conference on*, pages 1–9. IEEE, 2012.
- [Unl] Unlock Root. <http://www.unlockroot.com/>.

- [Uno07] Takeaki Uno. An efficient algorithm for enumerating pseudo cliques. In *Proceedings of ISAAC, ISAAC'07*, 2007.
- [Vin] Vine. <http://vine.co/>.
- [Vir15] VirusTotal - Free Online Virus, Malware and URL Scanner. <https://www.virustotal.com/>, Last accessed on May 2015.
- [WDR12] Marius Wernke, Frank Drr, and Kurt Rothermel. Pshare: Position sharing for location privacy based on multi-secret sharing. In *Per-Com*, pages 153–161, 2012.
- [Wek] Weka. <http://www.cs.waikato.ac.nz/ml/weka/>.
- [WeM] WeMo Switch. <http://www.belkin.com/us/p/P-F7C027/>.
- [Wik] Wikipedia current events. http://en.wikipedia.org/wiki/Category:Current_events.
- [Wit] Witness.org. <http://witness.org/>.
- [WKW⁺13] Gang Wang, Tristan Konolige, Christo Wilson, Xiao Wang, Haitao Zheng, and Ben Y. Zhao. You are how you click: Clickstream analysis for sybil detection. In *Proceedings of USENIX Security*, 2013.
- [WLTX06] Yan Wang, Yanghua Liu, Linmi Tao, and Guangyou Xu. Real-time multi-view face detection and pose estimation in video stream. In *Pattern Recognition, 2006. ICPR 2006. 18th International Conference on*, volume 4, pages 354–357. IEEE, 2006.
- [WXLY11] Guan Wang, Sihong Xie, Bing Liu, and Philip S. Yu. Review graph based online store review spammer detection. In *ICDM '11*, pages 1242–1247, 2011.
- [XPr] XPrivacy 1.9.5: The ultimate privacy manager. <http://forum.xda-developers.com/showthread.php?t=2320783>.
- [YBL⁺08] Anna Yu, Athanasios Bamis, Dimitrios Lymberopoulos, Thiago Teixeira, and Andreas Savvides. Personalized Awareness and safety with mobile phones as sources and sinks. In *International Workshop on*

Urban, Community, and Social Applications of Networked Sensing Systems (UrbanSense08), 2008.

- [Yela] Yelp. <http://www.yelp.com>.
- [Yelb] Yelp admits a quarter of submitted reviews could be fake. BBC, www.bbc.co.uk/news/technology-24299742.
- [Youa] YouTube. <http://www.youtube.com>.
- [youb] Youtube videos. <http://seclab.cs.fiu.edu/resources>.
- [YSM14] S.Y. Yerima, S. Sezer, and I. Muttik. Android malware detection using parallel machine learning classifiers. In *Next Generation Mobile Apps, Services and Technologies (NGMAST), 2014 Eighth International Conference on*, pages 37–42, Sept 2014.
- [YTWM00] Kenji Yamanishi, Jun-Ichi Takeuchi, Graham Williams, and Peter Milne. On-line unsupervised outlier detection using finite mixtures with discounting learning algorithms. In *Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '00*, pages 320–324, New York, NY, USA, 2000. ACM.
- [ZJ12] Yajin Zhou and Xuxian Jiang. Dissecting android malware: Characterization and evolution. In *Security and Privacy (SP), 2012 IEEE Symposium on*, pages 95–109. IEEE, 2012.
- [ZSGL07] Kejia Zhang, Shengfei Shi, Hong Gao, and Jianzhong Li. Unsupervised outlier detection in sensor networks using aggregation tree. In *Advanced Data Mining and Applications*, pages 158–169. Springer, 2007.
- [ZSK12] Arthur Zimek, Erich Schubert, and Hans-Peter Kriegel. A survey on unsupervised outlier detection in high-dimensional numerical data. *Statistical Analysis and Data Mining*, 5(5):363–387, 2012.
- [ZZL⁺12] Weiwei Zhang, Ru Zhang, Xianyi Liu, Chunhua Wu, and Xinxin Niu. A video watermarking algorithm of h. 264/avc for content authentication. *Journal of Networks*, 7(8):1150–1154, 2012.

VITA

MAHMUDUR RAHMAN

- 2010 – Now Ph.D., Computer Science
Florida International University
Miami, Florida, U.S.A.
- 2010 – 2012 M.S., Computer Science
Florida International University
Miami, Florida, U.S.A.
- 2003 – 2007 B.Sc., Computer Science and Engineering
Bangladesh University of Engineering and Technology
Dhaka, Bangladesh.

PUBLICATIONS

1. Mahmudur Rahman, Umut Topkara, Bogdan Carbunar. Movee: Video Liveness Verification for Mobile Devices using Built-in Motion Sensors. Accepted in the IEEE Transactions on Mobile Computing (IEEE TMC), 2015.
2. Mahmudur Rahman, Umut Topkara, Bogdan Carbunar. Secure Management of Low Power Fitness Trackers. Accepted in the IEEE Transactions on Mobile Computing (IEEE TMC), 2015.
3. Mahmudur Rahman, Bogdan Carbunar, Jaime Ballesteros, Duen Horng (Polo) Chau. To Catch a Fake: Curbing Deceptive Yelp Ratings and Venues. Statistical Analysis and Data Mining, Volume 8, Issue 3, Wiley, Pang-Ning Tan and Arindam Banerjee, editors (invited), 2015.
4. Bogdan Carbunar, Mahmudur Rahman, Jaime Ballesteros, Naphtali Rishe, Thanos Vasilakos. IEEE Transactions on Information Forensics and Security (TIFS), Volume 9, Issue 4, 2014.
5. Jaime Ballesteros, Bogdan Carbunar, Mahmudur Rahman, Naphtali Rishe, S.S. Iyengar. Towards Safe Cities: A Mobile and Social Networking Approach. IEEE Transactions on Parallel and Distributed Systems (TPDS), Volume 25, Issue 9, 2014.
6. Bogdan Carbunar, Mahmudur Rahman, Nikki Pissinou, A. V. Vasilakos. A Survey of Privacy and Security Attacks in GeoSocial Networks. In the IEEE Communications Magazine, Volume 15, Issue 11, 2013.
7. Mahmudur Rahman. Search Engines going beyond Keyword Search: A Survey. International Journal of Computer Applications (IJCA), Volume 25, No 17, 2013.

8. Mahmudur Rahman, Mozhgan Azimpourkivi, Umut Topkara, Bogdan Carbunar. Liveness Verifications for Citizen Journalism Videos. In proceedings of 8th ACM Conference on Security and Privacy in Wireless and Mobile Networks (ACM WiSec), NY City, June, 2015.
9. Mahmudur Rahman, Umut Topkara, Bogdan Carbunar. SensCrypt: A Secure Protocol for Managing Low Power Fitness Trackers. In proceedings of the 22nd IEEE International Conference on Network Protocols (ICNP'14), North Carolina, Oct 2014.
10. Mahmudur Rahman, Bogdan Carbunar, Jaime Ballesteros, George Burri, Duen Horng (Polo) Chau. Turning the Tide: Curbing Deceptive Yelp Behaviors. In proceedings of the SIAM International Conference on Data Mining (SDM), Philadelphia, April 2014.
11. Mahmudur Rahman, Umut Topkara, Bogdan Carbunar. Seeing is Not Believing: Visual Verifications through Liveness Analysis using Mobile Devices. In proceedings of the 29th Annual Computer Security Applications Conference (ACSAC'13), New Orleans, Dec 2013.
12. Bogdan Carbunar, Mahmudur Rahman, Jaime Ballesteros, Naphtali Rische. Private Location Centric Profiles for GeoSocial Networks. In proceedings of the 20th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, California, Nov 2013.
13. Mahmudur Rahman, Bogdan Carbunar, Madhusudan Banik. Fit and Vulnerable: Attacks and Defenses for a Health Monitoring Device. The 6th Workshop on Hot Topics in Privacy Enhancing Technologies (HotPETs), held in conjunction with PETS, Bloomington, July 2013.
14. Jaime Ballesteros, Mahmudur Rahman, Bogdan Carbunar, Naphtali Rische. Toward Safe Cities through Participatory Sensing. In proceedings of the 37th IEEE Conference on Local Computer Networks (LCN'12), Florida, Oct 2012.
15. Jaime Ballesteros, Mahmudur Rahman, Bogdan Carbunar, Naphtali Rische. Yelp Events: Making Bricks Without Clay?. In Proceedings of the International Workshop on Hot Topics in Peer-to-peer Computing and Social Networking (HotPOST'13), Philadelphia, July 2013.