Privacy in Computational Social Science

Riccardo Pietri

Kongens Lyngby 2013
IMM-M.Sc.-2013-68

Technical University of Denmark
Informatics and Mathematical Modelling
Building 321, DK-2800 Kongens Lyngby, Denmark
Phone +45 45253351, Fax +45 45882673
reception@imm.dtu.dk
www.imm.dtu.dk IMM-M.Sc.-2013-68

Summary

The goal of the thesis is to give an overview of privacy management in Computational Social Science (CSS), to show what is the current situation and to understand areas that can be improved. Computational Social Science is an interdisciplinary research process that gathers and mines wealth of *sensitive* data to study human behaviour and social interactions. It relies on the mixture of social studies and nowadays technologies such as smartphones and Online Social Networks. CSS's studies are aimed in understanding causes and effects in human behaviour, giving insights in their interactions, and trying to explain the inner nature of their relationship.

In the first part, it is presented an overview of existing CSS studies and their approach to participants' privacy. *Section 2* introduces CSS's capabilities and *Section 3* categorizes the works studied for this overview. The current situation regarding privacy regulations and informed consent practises for social experiments is discussed in *Section 4*. *Section 5* shows methods employed for securing users' data and relative threats. Anonymization techniques are discussed in *Section 6*. *Section 7* presents information sharing and disclosure techniques. Findings are summarized in *Privacy Actionable Items*.

Part II briefly illustrates *sensible-data*, a new service for data collection and analysis developed by the collaboration of DTU and MIT universities. *sensible-data* implements best practises and outlined improvements identified in *Part I*, de-facto setting new standards for privacy management in *Big Data*. In the CSS context, *sensible-data*'s contributions are two-fold: researchers have a unique tool to create, conduct, and share their studies in a secure way, while participants can closely monitor and control their personal data, empowering their privacy.

Part III shows the engineering process to create one of *sensible-data* framework's components. *sensible-auditor* is a tamper-evident auditing system that records in a secure way all the interactions within *sensible-data* system, such as users' enrolments, participants' data flows, etc. Design, implementation, and evaluation of *sensible-auditor*'s realization are presented after a general introduction that explains the role of *auditing* in system security.

Preface

This thesis was prepared at the department of *Informatics and Mathematical Modelling* at the *Technical University of Denmark* in fulfilment of the requirements for acquiring an Master of Science in *Computer Science and Engineering*.

Both my advisor, *Prof. Sune Lehmann*, and my supervisor, *Ph.D. Arek Stopczynski*, actively collaborated with me in the realization of this thesis. Their substantial contribution in the authorship of *Part I* can be summarized in: formulation of the research topic, structure planning, critical content revision, and discussion of the obtained results. It is our opinion that throughout the process, we gained valuable insights in the problematic of privacy in Computational Social Science, therefore it is our intention to publish an article based on such work.

Finally, the contents of *Part I* have been exposed and presented in visual form at the conferences *NetSci 2013* and *NetMob 2013* (see *Appendix* A).

This manuscript contains work done from January to July 2013.

Lyngby, 05-July-2013

[illegible]

Riccardo Pietri

Acknowledgements

First and foremost, I would like to thank my family. My mother, for her endless love; my father, for being a role model to me; my sister, because she is always there when I need her. I cannot say thank you enough for giving me this opportunity and your total support.

I would like to express my greatest appreciation to my advisor *Prof. Sune Lehmann* whose guidance and foresight helped me to a great extend with the realization of my master thesis. I want to thank him for the sincere engagement, the time and attention dedicated to me. A special gratitude I give to my supervisor *Ph.D. Arek Stopczynski* who offered invaluable assistance in stimulating and coordinating my project. I could not have imagined having a better team to work with.

A special thanks goes to *Tonia* for her humour; to *Bryan*, for all the insightful conversations; to *Francesco*, for his contagious enthusiasm about technology; to *Elbys* for reminding me that I am only a "stupid stupid Italian"! I also want to thank my amazing friends *Man*, *Bru*, *Ru*, and *Tusbi* for making it feel like home even if we had hundreds of kilometres between us. I am lucky to have such great friends.

Finally, an honourable mention goes to DTU, Google, and Wikipedia. To my music playlists, coffee, CBS library, Danish beer, English subtitles, Zefside, and Distortion. *"The path to becoming an engineer is littered with distractions. I would like to thank those distractions for making me the person I am".*

To my grandmother, who always knew I was going to be an engineer.
Thank you all

Contents

Part I

Privacy in Computational Social Science: A Survey

Chapter 1

Abstract

In recent years, the amount of information collected about human beings has increased dramatically. This development has been driven by individuals collecting their data in Online Social Networks (such as Facebook or Twitter) or collecting their data for self-tracking purposes (Quantified-Self movement). In addition, data about human behaviour is collected through the environment (embedded RFIDs, cameras, traffic monitoring, etc.), and in explicit Computational Social Science (CSS) experiments. CSS experiments and studies are often based on data collected with a high resolution and scale. Using computational power combined with mathematical models, such rich datasets can be mined to infer underlying patterns, providing insights into human nature. Much of the data collected is sensitive, private in the sense that most individuals would feel uncomfortable sharing their collected personal data publicly. For that reason, the need for solutions to ensure the privacy of the individuals generating data, has grown alongside the data collection efforts. Here, we focus on the case of studies designed to measure human behaviour, and note that — typically — the privacy of participants is not sufficiently addressed: study purposes are often not explicit, informed consent is ill-defined, and security and sharing protocols are only partially disclosed. In this paper we provide a survey of the work in CSS related to addressing privacy issues. We include reflections on the key problems and provide some recommendations for the future work. We hope that the overview of the privacy-related practices in Computational Social Science studies can be used as a frame of reference for future practitioners in the field.

Chapter 2

Introduction

Over the past few years the amount of information collected about human behaviour has increased dramatically. The datasets come from diverse sources such as user generated content in Online Social Networks (e.g. Twitter, Facebook) and on other online services (e.g. Flickr, Blogger); human communication patterns recorded by telecom operators and email providers; customer information collected by traditional companies and online wholesalers (e.g. Amazon, Target); data from pervasive environments such as sensor-embedded infrastructures (e.g. smart houses); Social Science experiments; the list continues.

As technology advances, the technical limitations related to storing and sharing these collections of information are gradually overcome, providing the opportunity to collect and analyse an unprecedented amount of digital data. This ever-increasing volume of *User Generated Content* intrinsically carries immense economic and social value and is thus of great interest for business organizations, governmental institutions, and social science researchers.

For the research community this data revolution has an impact that can hardly be underestimated. Data persistence and searchability combined with enhanced computational power has given rise to "Computational Social Science" (CSS), the interdisciplinary research process that gathers and mines this wealth of data to study human behaviour and social interactions [LPA+09, EPL09, CMP09]. Many CSS studies employ smartphones as *sociometers* [Pen08, ROE09, OMCP11] to sense, collect, and transmit large quantities of multi-purpose data. Data collection includes WiFi and Bluetooth devices IDs, GPS traces, SMS and call logs, data from social applications running on the devices (Facebook, Twitter), and more. While such a longitudinal approach allows scientists to maintain a broad scope, the scale and accuracy of the collected data often results in large amounts of sensitive information about the users, resulting in privacy concerns.

To a large degree, the public is unaware of the potential problems associated with sharing of the sensitive data. This lack of awareness is revealed in a number of contexts, for example via a documented tendency to "trade privacy for services" [Var12], or displaying carelessness regarding possible risks [Kru09, KBN11]. It has been shown that while many users are comfortable with sharing daily habits and movements through multiple applications, only a minority of them are aware of which parties are the receivers of this information. Concurrently pinpointing sensitive information about others is becoming easier using powerful search engines such as Google, Facebook Graph Search, or smartphone mashup apps (e.g *Girls Around Me*). Further aggravating this scenario, scientists have shown that many of the techniques employed so far to protect users' anonymity are flawed. Scandals such as re-identification of user in the *NetFlix Prize* data set [NS08] and other similar breaches [Swe00, BZH06] show that simple de-identification methods can be reversed to reveal the identity of individuals in those data sets [dMHVB13]. Attackers can also use these de-anonymization techniques to perpetrate so-called "reality theft" attacks [AAE$^+$11].

For the Computational Social Science field, ensuring the privacy of participants is crucial. Should scientists fail to defend participant's rights and their data, even in a single case, the consequences for CSS as a field could be catastrophic. A breach of security or abuse of sensitive data, could result in a loss of public confidence and — as a consequence — a decreased ability to carry out studies.

In order to avoid such a negative scenario and to maintain and increase the trust relation between research community and participants, the scientific community has to reconcile the benefits of their research with the respect for users' privacy and rights. The current situation in the field is a heterogeneous set of approaches that raise significant concerns: study purposes are often not made explicit, 'informed consent' is problematic in many cases, security and sharing protocols are only partially disclosed.

As the bitrate of the collected information, number of participants, and the duration of the studies increase, the pre-existing relation between researchers and participants will be growing weaker. Today the participants in the largest deployments of CSS studies are still students from particular university, members of a community, or friends and family of the researchers. Studies growing more open, allow for participants with no prior relation to the researchers. As a consequence, Ensuring good practices in informing those participants about their rights, the consent they express, the incentives etc. becomes even more important.

Our contributions in this paper are two-fold. First, we provide an overview of the privacy-related practices in existing CSS studies; we have selected representative works in the field and analysed the fundamental privacy features of each one. The result is a longitudinal survey that we intend as a frame of reference for current and future practitioners in the field. Second, we lay the groundwork for a *privacy management change process*. Using the review as a starting point, we have constructed a list the most important challenges to overcome for CSS studies: we call these *Privacy Actionable Items*. For each items, we delineate realistic implementations and reasonable life-spans. Our goal is to inspire introspection and discussion, as well as to provide a list of concrete items that can be implemented today and overcome some of the problems related to the current privacy situation.

2.1 Privacy

Probably the best known definition of *privacy* is *"the right to be left alone"*[1]. People should be able to determine how much of personal information can be disclosed, to whom, and how it should be maintained and disseminated[2].

Privacy can be understood as a conflict between liberty and control[3] where privacy hinges on *people* and *"enables their freedom"*. *Data confidentiality* is one of the instruments to guarantee privacy *"ensuring that information is accessible only to those authorized"*[4], [KZTO05]. Privacy is about *disclosure*, the degree to which a *private* information can be exposed. It is also related to *anonymity*, the property to remain unidentified in the public realm.

Disclosure boundaries, what is considered to be private and what is not, change among cultures, individuals, time, place, audience, and circumstances. The notion of privacy is thus dynamic, people protecting or disclosing their private information in order to gain some value in return [Cat97]. This process may lead to the paradoxical conclusion of *"trading off long-term privacy for short term benefits"* [AG05].

Privacy is a very complex topic that must be addressed from business, legal, social, and technical perspectives.

[1]*S. D. Warren* and *L. D. Brandeis* in *The Right to Privacy* (1890) [WB90].
[2]A. F. Westin *Privacy and Freedom* [WBC70].
[3]*B. Schneier*, `http://www.schneier.com/essay-114.html`
[4]http://research.uci.edu/ora/hrpp/privacyAndConfidentiality.htm

Chapter 3

Literature Review

We broadly categorize the projects selected for our survey into two families: generic frameworks and specialized applications. The former category contains platforms that collect a variety of different data streams deployed for the purposes of studying human behaviour in longitudinal studies. The second category consists of particular applications that collect data for a specific purpose.

3.1 Generic frameworks

Human Dynamics Group *Friends and Family* [API$^+$11] is a data collection study deployed by the Massachusetts Institute of Technology (MIT) in 2011 to perform controlled social experiments in a graduate family community. For the purpose of this study, researchers collected 25 types of different data signals (e.g. wireless network names, proximity to Bluetooth devices, statistics on applications, call and SMS logs) using Android smartphones as sensors. *Funf*[1]—the mobile sensing framework developed for this study—is the result of almost a decade of studies in *MediaLab Human Dynamics Group* on data collection using mobile devices. In 2008 a Windows Mobile OS [MCM$^+$11] was used to collect data from students and study connections between behavior and health conditions (e.g. obesity), and to measure

[1]Released as an open source framework available at `http://funf.org/`.

the spread of opinions. Four years prior, a team from MediaLab studied social patterns and relationships in users' daily activities, using Nokia phones [EP06]. And, in 2003 a Media Lab team pioneered the field by developing the first sensing platform [EP03] in order to establish how face-to-face interactions in working environments influence efficiency of organizations. While purposes of the studies and mobile sensing technologies have evolved, the general setup with a single server to collect and store the data coming from the devices, remained unchanged.

OtaSizzle A recent study conducted by the Aalto University in 2011 [KN11] in which researchers analyzed social relations combining multiple data sources. The results showed that in order to better describe social structure, different communication channels should be considered. Twenty students at the university were recruited by email invitation and participated in the experiment for at least three months. The research platform involved three different data sources: text messages, phone calls (both gathered with Nokia N97 smartphones), and data from a OSN experimental project called *OtaSizzle*, hosting several social media applications for Aalto University. All the gathered information were temporarily stored on the smartphone, before uploading to a central server.

Lausanne Another longitudinal mobile data collection, the *Lausanne Data Collection Campaign* (*LDCC*) [AN10, KBD+10, LGPA+12] (2009-2011) was conducted by Nokia Research Center with the collaboration of the EPFL institute of technology. The purpose was to study users' socio-geographical behavior in a region close to the Geneva Lake. The LDCC platform involved a proxy server that collected raw information from the phones and anonymized the data before moving them to a second server for research purposes.

3.2 Specialized frameworks

Here we present an overview of three groups of specialized platforms and smartphone applications developed by research groups for different purposes. In *Table 3.1* we present seven distributed architecture frameworks. Shifting the focus to privacy policies creation and management, we list three tools in *Table 3.2*. In *Table 3.3* we present the privacy related applications that generate, collect, and share information about users using smartphones as sensing devices. Other frameworks are also cited to provide useful examples. We remark that it is not our interest to discuss the primary goals of the mentioned studies (incentives, data mining algorithms, or results), but to present an overview on architectures, procedures and techniques employed for data collection and treatment – with a specific focus on privacy.

Name	Purpose	Privacy measures
Vis-a'-Vis [SLC+11] *2011 - Duke University, AT&T Labs*	A personal virtual host running in a cloud computing infrastructure and containing users' (location) information.	Allows users to manage their information directly from their virtual host with full control; exposes *unencrypted* data to the storage providers.
Confab [HL04] *2004 - University of California at Berkeley, University of Washington*	A distributed framework facilitating development of other privacy-aware applications for ubiquitous computing.	Personal data is stored in computers owned by the users, providing greater control over information disclosure.
MyLifeBits [GBL06, GBL+02] 2001 - Microsoft Research	Early example of digital database for individual's everyday life, recording and managing a massive amount of information such as digital media, phone calls, meetings, contacts, health data etc.	Information kept in SQL databases. Privacy concerns mentioned but *not* addressed in the project.
VPriv [PBB09] 2009 - MIT, Stanford University	Privacy-aware location framework for car drivers, producing an anonymized location database. Can be used to create applications such as usage-based tolls, automated speeding tickets, and pay-as-you-go insurance policies.	*Homomorphic encryption* [RAD78], ensures that drivers' identities are never disclosed in the application.
HICCUPS [MSF09] 2009 - University of Massachusetts Amherst	A distributed medical system where a) physicians and caregivers access patient's medical data; b) researchers can access medical aggregate statistics	Implements homomorphic encryption techniques to safeguard patients' privacy.
Darwin [MCR+10] 2010 - Dartmouth College, Nokia	A collaborative framework for developing a variety of sensing applications, such as place discovery or tagging applications.	Provides distributed machine learning algorithms running directly on the smartphones. Raw data is not stored on do not leave the smartphone.
AnonySense [CKK+08, KTC+08] 2008 - Dartmouth College	An opportunistic framework for applications using multiple smartphones to accomplish a single sensing task.	Provides anonymity to the users deploying k-anonymity [Swe02].

Table 3.1: *Distributed frameworks.* The first three frameworks are personal information collectors that play the roles of users' virtual aliases. Two implementations of homomorphic encryption for drivers and healthcare follow. *Darwin* and *AnonySense* are collaborative frameworks.

Name	Purpose	Privacy measures
PViz [MLA12] 2012 - University of Michigan	A graphical interface that helps social networks' users with policy comprehension and privacy settings.	Nodes represent individuals and groups, different colors indicate the respective visibility.
Virtual Walls [KHFK07] 2007 - Dartmouth College, University of St Andrews	A policy language that leverages the abstraction of physical walls for building privacy settings.	Three levels of granularity ("wall transparencies") allow users to control quality and quantity of information disclosure towards other digital entities (users, software, services).
A policy based approach to security for the semantic web [KFJ03] 2003 - University of Maryland Baltimore Country	A distributed alternative to traditional authentication and access control schemes.	Entities (users or web services) can specify their own privacy policies with rules associating credentials with granted rights (access, read, write, etc.).

Table 3.2: *Policy frameworks*. An overview of tools that help users to understand and control their policy settings.

Name	Purpose	Privacy measures
CenceMe [MLF+08] 2008 - Dartmouth College	Uses smartphones to sense peoples' activities (such as dancing, running, ...) and results are automatically shared on Facebook.	As soon the classification is performed on the devices the data is erased.
GreenGPS [GPA+10] 2010 - University of Illinois	A GPS navigation service which discovers *greener* (fuel-efficient) paths through drivers participatory collaboration (based on the previous framework *Poolview* [GPTA08]).	No fine-grained data control: if users feel the need for privacy, they need to switch off the GPS device to stop data collection.
Speechome Recorder [VGWR12, RPD+06] 2012 - MIT, Northeastern University	An audio/video recording device for studying children's daily behaviour in their family house.	Ultra-dense recordings temporary kept locally and uploaded to central server, but only scarce information about data encryption and transport security protocols.
Cityware [KO08] 2008 - University of Bath	Application for comparing Facebook social graph against real-world mobility traces detected using Bluetooth technology.	Switching Bluetooth to invisible as a way to protect users' privacy.
FriendSensing [QC09] 2009 - MIT, University College London	Bluetooth used to suggest new friendships evaluating device proximities.	Same as *Cityware*.
FollowMe [YL10] 2010 - Massachusetts Institute of Technology	Service that uses HTTP and Bluetooth to automatically share indoor position (malls, hospitals, airports, campuses).	Implements a decentralized architecture to improve users' location privacy.
Locaccino [TCD+10] 2010 - Carnegie Mellon University	A mobile sharing system created to study peoples' location privacy preferences.	Relevant privacy considerations will be reported later in the article.
Bluemusic [MKHS08] 2008 - RWTH Aachen, University of Duisburg-Essen	Application developed for studying personalization of public environments. It uses Bluetooth *public* usernames as pointers to web resources that store users preferences.	Same as *Locaccino*.

Table 3.3: *Specific applications.* Although providing a great functionality for the users, the privacy-oriented settings for the user are often not sufficiently implemented.

CHAPTER 4

Informed Consent

Here we examine the current situation of the participant's understanding and control over their personal data in CSS studies.

4.1 Privacy Regulations

The new ways of communication that have developed in the last decade, make every user, knowingly or not, a producer of large quantities of data that travel around the world in instant. Data can be collected in one country, stored in another, modified and accessed from yet elsewhere in a seamless way. The more global the data flow becomes, the more difficult it is to track how data is treated from a technical and legal point of view. For example, different legal jurisdictions may have different data protection standards and different privacy regulations [HNB11]. The result is that modern technology's pace is faster than regulations, leaving the users exposed to potential misuse of their personal data.

This situation lead the European Union to reform the past data protection regulations [1] into a comprehensive legal framework to strengthen online privacy rights

[1] "Recommendations of the Council Concerning Guidelines Governing the Protection of Privacy and Trans-Border Flows of Personal Data" (1980), "Convention for the Protection of Individu-

and foster citizens' trust in digital services. The *General Data Protection Regulation* (GDPR)[2] updates all the previously outlined principles for information (consent, disclosure, access, accountability, transparency, purpose, proportionality, etc.) to meet the new challenges of individual rights for *personal information* protection.

Fragmentation of the E.U. legal environment generates incoherences in different interpretations. This situation is the consequence of divergent implementations in the enforcement process of the state members, which try to follow the directions set by the E.U. directives.

Examples of how different states handle same topics under different legislations are the recent privacy case of *Google Street View* and the investigation on the smartphone application *Whatsapp*. In the former case, the German authority for data protection requested the data collected by the Google cars, intended to photograph public spaces[3]. They discovered a piece of code[4] that captured unencrypted Wi-Fi traffic (user names, passwords, bank details, etc.). Immediately after this disclosure, the respective authorities of U.K. and France inquired the company accordingly to their respective (different) legislations.

In the latter case, the Dutch Data Protection Authority published[5] the findings of an investigation into the processing of personal data for the well-known smartphone application *Whatsapp*. The results revealed a series of security malpractices and privacy abuses: messages were sent unencrypted, algorithms for generating passwords used identifiable device information making relatively easy to be compromised; message history was stored unencrypted on the SD memory card in Android phones. In addition, to facilitate the contact between the users, *WhatsApp* required the access to the whole address book, leaking phone numbers of non-users of the service. This violation is now subject of the Italian Data Protection Authority's inquiry.

These cases show the need for a common regulator that can guarantee to E.U. citizens privacy rights and allow the states members to join their forces and oppose abuses. This fragmentation also affects CSS studies in the privacy policy formulation. As we discuss in the next section, in the cases where privacy policies were created, developers and scientists needed to use their own best judgment, since no common frameworks to use as reference were available, causing large divergences among universities and studies. As an example, the LDCC study performed by EPFL

---

als with regard to Automatic Processing of Personal Data" (1981), "Data Protection Directive" 1995/46/EC, "Privacy Directive" 2002/58/EC , "Data Retention Directive" 2006/24/EC.

[2]Drafted in 2011 and at the time of writing awaiting for European Parliament's first reading.

[3]The interest in the *Google Street View* project raised after people's concerns about being showed in "uncomfortable situations or locations" (e.g. closeness to such as strip clubs, drug clinics, etc.).

[4]`http://www.dailymail.co.uk/sciencetech/article-2179875/Google-admits-STILL-data-Street-View-cars-stole.html`

[5]`http://www.dutchdpa.nl/Pages/en_pb_20130128-whatsapp.aspx`

and Nokia Research Center followed Nokia's *generic* data treatment for processing the participants information.

Another point stressed by the regulation is the *Right to be forgotten*, which states that every user can request at any time the total deletion of personal data from any service or study he has been involved with. A recent European campaign promoted by an Austrian law student interested in Facebook's use of his personal information[6]. Hidden in the 1224 page long report that the social network sent to him when requested, he found that the social network retained the data that he had never consented to disclose as well as data he had previously deleted. The right to be forgotten should also be granted to CSS study participants allowing the user to remove their personal data from the dataset at any time.

The GDPR facilitates the process of transferring personal information from one service provider to another. As already stated, privacy regulations may vary across country boundaries: it might happen that data of E.U. residents will be process by foreign entities; therefore it is GDPR's main concern to extend the whole new policy framework for data protection to all the foreign countries (*data portability right*), assuring users that data will be processed according to the E.U. legislation. For the studies conducted in the United States, Institutional Review Boards (IRBs) are the authorities for privacy regulation for behavioral researches involving humans participants. These academic committees need to "approve, monitor, and review" all the CSS experiments "to assure, both in advance and by periodic review, that appropriate steps are taken to protect the rights and welfare of humans participating as subjects in a research study". One of these step is to obtain trial protocol(s)/amendment(s) and written *informed consent* form(s).

To summarize, CSS scientists should move in the direction of deploying tools for allowing participants to view, cancel, copy, and also transmit collected data from one study to another in the respect of the new regulation. In addition, given the massive amount of data collected in CSS studies – which intrinsically contain large quantities of sensitive personal information – we recommend that the GDPR will include also common guidelines for the CSS field.

In the CSS studies, *informed consent* consists of an agreement between researchers and the data producer (user, participant) by which the latter confirms she understands and agrees to the procedures applied to her data (collection, transmission, storing, sharing, and analysis). The intention of the informed consent is that the users comprehend which information will be collected, who will have access to that information, what the incentive is, and for which purposes the data will be used [FLM05]. In CSS studies the research ethic is paramount for protecting volunteers' privacy, therefore scientists might need to work under *Non-Disclosure Agreements*

[6]`http://www.nytimes.com/2012/02/06/technology/06iht-rawdata06.html`

to be able to perform analyses on the collected data [LGPA+12, KN11].

Here we note the scarcity of available informed consent examples in the published studies; the majority of the studies we reviewed have *not* published their consent procedures [MCM+11, EPL09, YL10, MMLP10, CMP09, MLF+08, API+11, MFGPP11, OWK+09]. Due to this fact, it is difficult to produce comparisons and create useful models applicable for future studies. Where the procedure for achieving informed consent are reported, the agreement was carried out using forms containing users' rights (similar to `http://green-way.cs.illinois.edu/GreenGPS_files/ConsentForm.pdf`, e.g. [KBD+10, GPA+10, KAB09, EP06, MCM+11, KN11]) or by accepting the *Terms of Use* during the installation of an application. It is common among the studied frameworks and applications to allow the users to *opt-out* from the experiment at any moment, as required by the research boards or ethics in general.

4.2 Understanding

Presenting all the information to the user does not guarantee that informed consent is implemented sufficiently: years of EULAs and other lengthy legal agreements show that most individuals tend to blindly accept forms that appear before them and to unconditionally trust the validity of the default settings which are perceived as authoritative [BK10]. One improvement would be to allow users to gradually grant permission over time, but the efficacy of this approach is not clear yet: some studies have shown that users understand the issues about security and privacy more when the requests are presented gradually [EFW12]; others argue that too many warnings distract users [KHFK07, FGW11, Cra06]. So far there has been little interest whether informed consent actually informs the audience. Evaluating how people understand their privacy conditions can be done by conducting feedback sessions throughout the duration of the experiment [KBD+10, MKHS08].

Nevertheless, a simple yes/no informed consent option does not live up to the complex privacy implications related to studies of human behaviour. For that reason, users should play a more active role in shaping their involvement in such studies. This view gains support from studies showing that people do not in general realize smartphone sensing capabilities nor the consequences of privacy decisions [KCC+09, FHE+12, Kru09]. Additionally, special cases where the participants may not have the competence or the authority to fully understand the privacy aspects [SO13, URM+12, VGWR12, SGA13] should be carefully considered. Finally, it is fundamental to clearly state study purposes when performing data collection *for later use* and to inform the participants about what happens to the data once the study is concluded [FLM05].

4.3 Control

In most cases, the current informed consent techniques represent an *all-or-nothing* approach, that does not allow the user to select subsets of the permissions, making it only possible to either participate in the study fully or not at all [FGW11]. In addition, once the consent is granted by the user, all his data contribution to the dataset becomes owned by the researchers, in that they can analyze, modify, or redistribute it as they see fit, depending on the terms of the consent, but typically simply provided that basic de-identification is performed. As we suggest in *Section 7*, it is a good practice for the researchers to clarify to the participants the sharing schemes and expiration of the collected information: if users cannot follow the flow of their data, it is difficult to claim that a real informed consent is expressed. Since so little is understood about the precise nature of conclusions that may be drawn from high resolution data, it is important to continuosly work to improve and manage the informed consent as new conclusions from the data can be drawn. We recommend that the paradigm should move from a one-time static agreement to dynamic consent management [Sha06]. Furthermore, the concerns related to privacy are context-specific [TCD$^+$10, LCW$^+$11] and vary across different cultures [ABK09, MKHS08]. In the literature, the need for a way to let the users easily understand and specify which kinds of data they would like to share and under what conditions was foreseen in 2002 by the W3C group, with the aim to define a *Platform for Privacy Preferences (P3P)* (suspended in 2006), in 2003 by *Kagal et al.* [KFJ03], and also in 2005 by *Friedman et al.* [FLM05], all shaping dynamic models for informed consent. Recent studies such as [TSH10] have worked to design machine learning algorithms that automatically infer policies based on user similarities. Such frameworks can be seen as a mixture of recommendation systems and collaborative policy tools where default privacy settings are suggested to the user and then modified over time.

4.4 Living Informed Consent.

We propose the term *Living Informed Consent*, for the aligned business, legal, and technical solutions where the participant in the study is empowered to understand the type and quality of the data that is being collected about her, not only during the enrolment, but also when the data is being collected, analysed, and shared with 3rd parties. Rather than pen & clipboard approach for the user enrolment, the users should expect to have a virtual place (a website or application) where they can change their authorizations, drop-out from the study, request data deletion, as well as audit who, and how much is analysing their data. As the quantity and quality of the data collected increases, it becomes difficult to claim that single sentence description of *we will collect your location* truly allows the participant to realize the complexity of the signal collected and possible knowledge that can be extracted from it. Such engaging approach to the users' consent will also be beneficial for the research community: as the relation with the user in terms of their consent expression extends beyond initial enrolment, the system proposed here makes it possible for the user to sign up for the new studies and donate their data from the other studies.

CHAPTER 5

Data Security

The security of the collected data, although necessary for ensuring privacy goals, is something that is not often discussed [GPA$^+$10, MCR$^+$10, MLF$^+$08, KAB09, VGWR12, KN11, SLC$^+$11]. In the next sections we illustrate how security has been addressed in the *centralized frameworks* and how it can be integrated in (future) *distributed solutions*. This is not an exhaustive list, but a compendium of techniques that can be applied for CSS frameworks, as well as attacks that one needs to consider.

5.1 Centralized architecture

The centralized architecture, where the data is collected in a single dataset, has been the preferred solution in the majority of the surveyed projects [VGWR12, URM$^+$12, KHFK07, PBB09, MFGPP11, MMLP10, KBD$^+$10, MLF$^+$08, GPA$^+$10, VGWR12, RPD$^+$06, API$^+$11, OWK$^+$09, EP06, MCM$^+$11, KO08, QC09, TCD$^+$10, MKHS08, KN11]. The centralized architecture suffers from several problem. First, if the server is subject to *denial-of-service* attacks, it can not guarantee the availability of the service. This might result in the smartphones having to retain more information locally with consequential privacy risks. More importantly, if compromised, a single server can reveal all user data.

The number of malware and viruses for mobile phones is growing. Given the amount of sensitive information present on these devices, social scientists should consider using and developing robust portable applications in order to avoid privacy thefts[AAE+11]. To tackle this problem, some of the studied frameworks reduce the time that the raw information collected by the sensors is kept on the phone. For example, in *Darwin* platform the data records are discarded once the classification task has been performed. Since most of the sensing applications use opportunist uproach to the data uploading, they might store a large amount of data temporarily on external memory [MFGPP11]. This introduces a security threat if the device does not procure an encrypted file-system by default. A possible way to tackle this problem is employing frameworks like *Funf* the open-source sensing platform developed for [API+11] and also used in the *SensibleDTU* study. *Funf* provides a reliable storing system that encrypts the files before moving them to special archives on the phone memory. An automatic process uploads the archives, keeping a temporary (encrypted) backup. This mitigates the risk of disclosure of information if the smartphone is lost or stolen. In such case, the last resort would be to provide a remote access to delete the data off the phone. Generally, to reduce the risks, good practice is to minimize the amount of information exchanged and avoid transmitting the raw data [MLF+08].

Some frameworks use default HTTP protocol to transmit data [HL04, MLF+08, GPA+10, YL10, MKHS08], other use *HTTP over SSL* to secure data transmission [SLC+11, CKK+08, KTC+08, KAB09], but pushing data through WiFi connection remains the most common scenario [API+11, MCM+11, EP06, EP03, AN10, KBD+10, LGPA+12, TCD+10]. Event encrypted content can disclose information to malicious users, for example by observing the traffic flow: the opportunistic architecture of transmission and the battery constrains do not allow smartphones to mask communication channels with dummy traffic to avoid such analysis [HL04, CKK+08].

When data reach the central server, is is usually stored in SQL databases (e.g. [API+11, GBL06, GBL+02, MMLP10, MFGPP11]) which aggregate them for later analysis. We remark that in all of the surveyed frameworks, the mechanisms for access control, user authentication, and data integrity checks (where present), had been implemented for the purpose of given study. For example, in *OtaSizzle* "the data are stored in local servers within the university to ensure adequate security" [KN11]. Ensuring security of the data is a very complex task. We believe that common solutions and practices are important so that the researchers do not need to worry about creating new security platforms for every new study. Finally, given the importance of the stored data, the security of the entire CSS platform (network and server) may be enhanced accordingly to defence in depth paradigm, as illustrated in the guidelines on firewalls [SH09] and intrusion detection systems [SM07] by the National Institute of Standards and Technology (NIST).

5.2 Distributed architecture

In the recent years, the trend has been to store the data in highly distributed architectures, or even off-site, in the "cloud". We define the cloud as any *remote* system which provides a service to users relying on the use of *shared* resources (see [HNB11, FLR12] for different cloud typologies). An example can be a storage system which allows the users to backup their files and ubiquitously access them via the Internet (e.g. Dropbox).

Apart from facilitating the processes of data storage and manipulation, employing cloud solutions can improve the overall security of CSS studies. For every surveyed study [SLC$^+$11, HL04, GBL06, GBL$^+$02, MSF09, MCR$^+$10, CKK$^+$08, KTC$^+$08, KAB09, KO08, YL10], the platforms have been designed and implemented from scratch, in the environment where the thorough testing with respect to security may not be a priority. On the other hand, if platforms like Amazon EC2 are integrated in CSS frameworks, security mechanisms such as access control, encryption schemes, and authorization lists can be enforced in standard and well tested ways. Buying Infrastructure-as-a-Service or Platform-as-a-Service may also be seen to certain extent as buying Security-as-a-Service. In addition, using the cloud solutions can make it possible to create CSS frameworks that allow users to *own* their personal information. Having the constant option to monitor the status of personal data, to control who has access to those data and to be certain of deletion, can make users more willing to participate. One possible way to achieve this, is to upload the data from the mobile devices not to a single server, but to personal datasets (e.g.: personal home computers, cloud-based virtual machines) as shown in *Vis-a'-Vis*, *Confab*, *MyLifeBits* platforms. On one hand, with these electronic aliases users will feel — and possibly be — more in control of their personal data, diminishing their concerns about systems that centralize data. On the other hand, part of the security of users' own data will inevitably rely on the themselves - and on the Service Providers (SPs) who manage the data.

Many of the security mechanisms for centralized solutions can be deployed for distributed approaches too, therefore making a smooth transition towards the cloud feasible. We illustrate the similarities following CSS study steps. Data is usually collected using smartphones (e.g. via smartphone sensing platforms like *Funf*), then it is transmitted over HTTPS connections and stored onto personal datasets (instead of a single server). Then, these information can be analysed using distributed algorithms capable of running with inputs coming from different nodes (the personal datasets), as illustrated by the *Darwin* framework. Prior to this, a discriminating choice determines whether data has to be encrypted or not before being uploaded in the cloud. For example, the distributed solution *Vis-a'-Vis* exposes *unencrypted* data to the storage providers since this facilitates queries to be executed on the remote storage servers by other web services. The opposite approach is to encrypt data *before* storing it in the cloud. Unfortunately, while this approach enhances the confidentiality of users' data (preventing the SPs from reading personal encrypted files), it also hinders CSS scientists from running algorithms on the collected data. We examine in *chapter 7.1* how can computations on encrypted data can be performed with the help of two example frameworks: *VPriv* and *HICCUPS*.

Given the sensitive nature of the data, vulnerabilities in cloud architectures can pose serious risks for CSS studies and, while cloud solutions might provide an increased level of security, they are definitely not immune to attacks. See [CRI10] for a attack taxonomy, [HRFMF13] for a general analysis on the cloud security issues. Sharing resources is a blessing and a curse of cloud computing: it helps to maximize the utility/profit of resources (CPU, memory, bandwidth, physical hardware, cables, operative systems, etc.), but at the same time it makes it more difficult to assure security since both physical and virtual boundaries must be reinforced. The security of the *Virtual Machines* (VM) becomes as important as the physical security because "any flaw in either one may affect the other" [HRFMF13]. Since multiple virtual machines are hosted on the same physical server, attackers might try to steal information from one VM to another; *cross-VM attacks* [RTSS09]). One way to violate data confidentiality is compromising the software responsible for coordinating and monitoring different virtual machines (hypervisor) by replacing its functionalities with others aimed at breaching the isolation of any given pair of virtual machines, a so-called Virtual Machine Based Rootkit [KC06]. Another subtle method to violate security is via *side-channels attacks* [AHFG10] which exploit unintended information leakage due to the sharing of physical resources (such as CPU's duty cycles, power consumption, memory allocation). For example, a malicious software in one VM can try to understand patterns in memory allocation of another co-hosted VM without the need of compromising the hypervisor. One of the first real examples of such attacks is shown in [ZJRR12] where the researchers demonstrated how to extract private keys from an adjacent VM. Finally, deleted data in one VM can be resurrected from another VM sharing the same storage device (Data scavenging [HRFMF13]) or the whole cloud infrastructure can be mapped to locate a particular target VM to be attacked later [RTSS09]. In addition the volatile nature of cloud resources makes difficult to detect and investigate attacks: when VMs are turned off, their resources (CPU, RAM, storage, etc.) become available to other VMs in the the cloud [HNB11] making it difficult to track processes.

Therefore, while we believe that the cloud is becoming more important in CSS studies, the current situation still presents some technical difficulties that need to be addressed. We will focus on methods to control data treatment (*information flow* and *data expiration*) for remote storage systems in section 7.2 to ensure the users about compliance to privacy agreements.

Chapter 6

Privacy and Datasets

The datasets created for CSS studies often contain extremely sensitive information about the participants. *NIST Special Publication 800-122* defines PII as *"any information about an individual maintained by an agency, including any information that can be used to distinguish or trace an individual's identity, such as name, social security number, date and place of birth, mother's maiden name, or biometric records; and any other information that is linked or linkable to an individual, such as medical, educational, financial, and employment information"*[1]. It is researchers' responsibility to protect users' PIIs and consequently their privacy when disclosing the data to public scrutiny [NS08, BZH06, Swe00] or to guarantee that the provided services will not be abused for malicious uses [YL10, PBB09, HBZ$^+$06]. PII can be removed, hidden in group statistics or modified to become less obvious and recognizable to others, but the definition of PII is context dependent, making it very difficult to select which information needs to be purged. In addition, modern algorithms can re-identify individuals even if no apparent PII are published [RGKS11, AAE$^+$11, LXMZ12, dMQRP13, dMHVB13]

We remark that making data anonymous (or de-identified) decreases the data utility by reducing resolution or introducing noise ("privacy-utility tradeoff" [LL09]). To conclude we report attacks that deprive users' privacy, by reverting anonymization techniques.

[1]NIST Special Publication 800-122 `http://csrc.nist.gov/publications/nistpubs/800-122/sp800-122.pdf`

6.1 Privacy Implementations

When sensitive information are outsourced to untrusted parties, various technical mechanisms can be employed to enhance the privacy of participants, by transforming the original data into a different form. In the next sections we present two common ways to augment users' privacy: noise and anonymization, as well as recent developments in applied homomorphic encryption For a classification of different privacy implementation scenarios – such as multiple, sequential, continuous or collaborative data publishing – see [FWCY10].

Noise A difficult trade-off for CSS researchers is how to provide third parties with accurate statistics on the collected data while at the same time protecting the privacy of the individuals in the records. In other words, how one may address the problem of *statistical disclosure control*. Although there is a large literature on this topic, the variety of techniques can be coarsely divided in two families: approaches that introduce noise directly in the database (which are called *data perturbation models* or *offline methods*) and a second group that interactively modifies the database queries (*online methods*). The first method aims to create safe views of the data, for example releasing aggregate information like summaries and histograms. The second actively reacts to the incoming queries and modify the query itself or affects the response to ensure privacy.

Early examples of these privacy-aware data mining aggregations can be found in [AS00]. Here the authors consider building decision-tree classifier from training data with perturbed values of the individual records, and show that it is possible to estimate the distribution of the original data values. This implies that it is possible to build classifiers whose accuracy is comparable to the accuracy of classifiers trained on the original data. In [AA01] the authors show an Expectation Maximization (EM) algorithm for distribution reconstruction, providing robust estimates of the original distribution given that large amount of data is available. A different approach is taken in [EGS03] where the authors present a new formulation of privacy breaches and propose a methodology for limiting them. The method, dubbed *amplification*, makes it possible to guarantee limits on privacy breaches without any knowledge of the distribution of the original data. An interesting work on the tradeoff between privacy and usability of the perturbed (noisy) statistical databases has been redacted in [DN03].

In [DN04] the results from [DN03] are revisited, investigating the possibility of sub-linear number of queries on the database which would guarantee privacy, extending the framework. A second work consolidates discoveries from [DN03], demonstrating, the possibility to create a statistical database in which a trusted administrator introduces noise to the query responses with the goal of maintaining privacy of individual database entries. In [BDMN05] the authors show that this can be achieved using a surprisingly small amount of noise – much less than the sampling error – provided the total number of queries is sublinear in the number of database rows. A different approach is evaluated by *Dwork et al.* in [DKM+06], where an efficient distributed protocol for generating shares of random noise and secure against malicious participants is described. The innovation of this method is the distributed implementation of the privacy-preserving statistical database with noise generation. In these databases, privacy is obtained by perturbing the true answer to a database query by the addition of a small amount of Gaussian or exponentially distributed random noise. The distributed approach eliminates the need for a trusted database administrator. Finally, in [CDM+05] *Chawla and Dwork* proposed a definition of privacy (and privacy compromise) for statistical databases, together with a method for describing and comparing the privacy offered by specific sanitization techniques. They obtained several privacy results using two different sanitization techniques, and then show how to combine them via cross training. They also obtained two utility results involving clustering. This work is advanced in a more recent study [CDMT12], where the scope of the techniques is extended to a broad class of distributions and randomization the histogram constructions to preserve spatial characteristics of the data, allowing to approximate various quantities of interest, e. g., cost of the minimum spanning tree on the data, in a privacy-preserving fashion. We discuss problems with those strategies below.

Anonymization The most common practice in the data anonymization field is to one-way hash all the PII such as MAC addresses, network identifiers, logs, names, etc. This breaks the direct link between a user in given dataset to other, possibly public datasets (e.g. *Facebook* profile). There are two main methods to achieve this. The first - used in the *LDCC* study - is to upload raw data from the smartphone to an intermediate proxy server where algorithms hash the collected information. Once anonymized, the data can be transferred to a second server which researcher have access to. A less vulnerable option is to hash the data directly on the smartphones and then upload the result the final server for analysis. This alternative has been selected for many MIT studies [API+11, MFGPP11, MMLP10, MCM+11] and for the *SensibleDTU* project (`http://www.sensible.dtu.dk/`). In principle, hashing does not reduce the quality of the data (provided that it is consistent within the dataset), but it makes it easier to control which data are collected about the user and where it comes from. However, it does not guarantee that users cannot be identified in the dataset [BZH06, Swe00, NS08].

Finally, some types of raw data - like audio samples - can be *obfuscated* directly on the phone without losing the usability before being uploaded [KBD+10, OWK+09].

Another frequent method employed for anonymization is ensuring k-anonymity [Swe02] for a published database. This technique ensures that is not possible to distinguish a particular user from at least $k-1$ people in the same dataset. *AnonySense* and the platform developed for the *LDCC* both create k-anonymous different-sized tiles to preserve users' location privacy, outputting a geographic region containing at least $k-1$ people instead of single user's location. Nevertheless, later studies have shown how this property is not well suited for a privacy metric [STLBH11]. First, *Machanavajjhala et al.* tried to solve k-anonymity weaknesses with a different privacy notion called l-diversity [MKGV07]; then, *Li et al.* proposed a third metric, t-closeness, arguing against the necessity and the efficacy of l-diversity [LLV07]. Although these two techniques seem to overcome most of the previous limitations, they have not been deployed in any practical framework to date. Finally, while today's anonymization techniques might be considered to be robust enough in providing privacy to the users [CEE11], our survey contains methods that manage to re-identify participants in anonymized datasets (see section 6.2).

Homomorphic encryption Homomorphic encryption is a cryptographic technique [RAD78, Gen09] that enables computation with encrypted data: operations in the encrypted domain correspond to meaningful operations in the plaintext domain. This way, users can allow other parties to perform operations on their encrypted data without exposing the original plaintext, limiting the sensitive data leaked.

Such mechanism can find application in health-related studies, where patients' data should remain anonymous before, during, and after the studies while only authorized personnel has access to clinical data. *Data holders* (hospitals) send encrypted information on behave of *data producers* (patient) to untrusted entities (e.g. researchers and insurance companies) which process them without revealing the data content, as formalized by *mHealth*, an early conceptual framework. *HICCUPS* is a concrete prototype that permits researchers to submit medical requests to a query aggregator that routes them to the respective caregivers. The caregivers compute the requested operations using sensitive patients' data and send the reply to the aggregator in encrypted form. The aggregator combines all the answers and delivers the aggregate statistics to the researchers. A different use of homomorphic encryption to preserve users' privacy is demonstrated by *VPriv*. In this framework the central server first collects anonymous tickets produced when cars exit the highways, then by homomorphic transformations it computes the total amount that each driver has to pay at the end of the month.

Secure two-party computation can be achieved with homomorphic encryption when both parties want to protect their secrets during the computations: none of the involved entities needs to disclose its own data to the other, at the same they achieve the desired result. In [FDH^{+}12] the researchers applied this technique to private genome analysis. A health care provider holds patient's secret genomic data, while a bioengineering company has a secret software that can identify possible mutations. Both want to achieve a common goal (analyze the genes and identify the correct treatment) without revealing their respective secrets: the health care provider is not allowed to disclose patient's genomic data; the company wants to keep formulas secret for business related reasons.

Lately, much effort has been made in building more efficient homomorphic cryptosystems (e.g. [TEHEG12, NLV11]), but we can not foreseen whether or when the results will be practical for CSS frameworks.

6.2 Attacks against Privacy

Every day more and more information about individuals become publicly available [TP12, LXMZ12]. *Paul Ohm* in [Ohm10] defines this trend as the "*database of ruin*" which is inexorably eroding people's privacy. While the researchers mine the data for scientific reasons, malicious users can misuse it in order to perpetrate a new kind of attack: *reality theft*, the "illegitimate acquisition and analysis of people's information"[AAE+11].

Thus, like scientists, reality thieves aim decode human behaviour such as everyday life patterns [STLBH11], friendship relations [QC09, EPL09], political opinions [MFGPP11], purchasing profiles[2], etc. There are companies that invest in mining algorithms for making high quality predictions while others are interested in analyzing competitors' customer profiles [Var12]. Attackers are also developing new types of malware to steal hidden information about social networks directly from smartphones [AAE+11]. Scandals such as *NetFlix Prize*, *AOL searcher* [BZH06] and the *Governor's of Massachusetts health records* [Swe00] show that the anonymization of the data is often insufficient, as it may be reversed revealing the original individuals' identities.

A common approach is to compare "anonymized" datasets against the publicly available ones. These take the name of *side channel information* or *auxiliary data*. For this, social networks are excellent candidates [LXMZ12]. In recent studies [SH12, MVGD10], researchers have shown that users in anonymized datasets may be re-identified studying their interpersonal connections on their public websites like *Facebook*, *LinkedIn*, *Twitter*, *Foursquare*, and others. The researchers identified similar patterns connecting pseudonym in the anonymized dataset to the users' ("real") identity in a public dataset. Frameworks have great difficulty thwarting

[2] http://adage.com/article/digital/facebook-partner-acxiom-epsilon-match-store\-purchases-user-profiles/239967/

these side channel attacks. For example, users' anonymity might be compromised in *VPriv* and *CarTel* every time only a single car is driving on a highway, because it is possible to link the anonymous packets reaching the server to that (unique) car. Same type of consideration is valid for *AnonySense* and *VirtualWalls* (if only one user is performing the sensing task for the first platform or if only one user is located inside a room at a certain time for the second platform).

Internal linking [LXMZ12] is another attack that aims to bond together different interactions of one user within the same system. For example, in [KTC$^+$08] two reports uploaded by a single user might be linked based on their timing.

More generally, it is not the released data that is the source of privacy issues, but the unexpected inferences that can be drawn from it, that worry the researchers, as pointed out by *Tene and Polonetsky* in [TP12]. Some of the surveyed frameworks like *CenceMe*, *FollowMe*, and *Locaccino* allow the users to update their daily habits. Studying symmetries (e.g. in commuting) and frequencies (e.g. meals or weekly workouts) of these behaviors, it is possible to discover underlying patterns and perpetrate attacks against users' privacy [LXMZ12].

Finally, data collected by seemingly innocuous sensors can be exploited to infer physiological and psychological states, addictions (illicit drugs, smoking, drinking), and other private behaviors. Time and temperature can be correlated to decrease location privacy [AA11], while accelerometers and gyroscopes can be used to track geographic position [RGKS11] (e.g. the platform created for the *LDCC* did not anonymize accelerometer data nor other smartphone data, such as battery level and running applications) or even to infer peoples' mood [LXMZ12]. A recent study conducted by the University of Cambridge showed that accurate estimates of personal attributes (such as IQ levels, political views, substance use, etc.) can be inferred from the *Facebook Likes*, which are publicly available by default[3]. These threats are even more dangerous as people seem to not be aware of what can be inferred from seemingly harmless data [MKHS08] nor about smartphone sensing capabilities [KCC$^+$09]. For example, participants of the *Bluemusic* experiment did not show any concerns in "recording all day, everyday" and "store indefinitely on their mobile phone" data collected by accelerometers because perceived as "not particularly sensitive" [MKHS08]. The consequences of *reality theft* can be long lasting and can be much worse than any other kind of attack: it is almost impossible to change personal relationships or life patterns[4] to avoid stalking or other types of criminal activity that might occur because of misuses of behavioral datasets[AAE$^+$11]. Peo-

[3]`http://www.cam.ac.uk/research/news/digital-records-could-expose-intimate-\details-and-personality-traits-of-millions`

[4]Although for slightly different reasons, in 2010 Google's Executive Chairman Eric Schmidt suggested automatic changes of virtual identities to reduce the limitless power of the *database of ruin*: `http://online.wsj.com/article/SB10001424052748704901104575423294099527212.html` .

ple have now little control over their data and the release of their information is an irreversible action. We suggest that the potential for such misuses or attacks during should be mentioned in the informed consent processes in the CSS field. Finally, as seen in the previous chapters, many of the studied platforms build privacy for the users on the hypothesis that techniques like k-anonymity and protections against traffic analysis or side channels will be possibly added in future. At the time of writing, such techniques are not yet integrated, leaving the presented solutions only marginally secure. As practitioners of CSS, we feel the necessity for frameworks that provide holistic, reusable solutions to privacy concerns.

Chapter 7

Information Ownership and Disclosure

CSS frameworks should ideally guarantee the users ways to control who is in possession of their data at any time. Past attempts to create types of *Digital Rights Management* systems for privacy purposes did not show the expected results and without *Trusted Computing Bases* there is no practical way of being sure that data has not been retained/copied or forwarded to third parties [HL04]. Trusted Computing Base of a computer system is a collection of all and only those components (hardware and software) critical to its security. These must be tamperproof and testable for integrity/authenticity and their vulnerabilities might jeopardize the entire system. Whether the users let the researchers physically own the data stored on the universities' servers (see section 5.1) or simply let the scientists borrow the information from the personal datasets (section 5.2 and 7.1), one problem remains: users need to trust that researchers and Service Providers properly manage their personal information as agreed, and do not expose any sensitive data to unauthorized parties. In the following sections we provide more details about two main information disclosure scenarios: the explicit and conscious process of distributing information to other parties, i.e. *information sharing*; and techniques to *control data disclosure*: *data expiration* systems and protections against covert *information flows*.

7.1 Sharing

Individuals' notion of information sensitivity and therefore sharing patterns vary [YL10, Sha06, LCW$^+$11, TCD$^+$10]; some information however should always be perceived as sensitive and requires special attention, examples are health-related, financial, or location data. It is often the case that the sensitivity of the information is proportional to its research value, making users reluctant about disclosure [Kot11, RGKS11, KCC$^+$09]. For example, recent studies have demonstrated that social networks can be mined in order to discover psychological states [LXMZ12, dMQRP13], which can be later used to detect unusual patterns and prevent or treat psychological disorders (social anxiety, solitude, etc.).

Sharing with other users. Social networks and smartphone applications such as *CenceMe*, *Bluemusic*, *FollowMe*, and *Cityware* show that people are comfortable with the idea of sharing personal information with friends [KBN11]. Unfortunately, im most cases the data sharing options lack granularity. For example, users of *CenceMe* and *FollowMe* can unfriend other participants resizing the sharing set while users of *Cityware* or *GreenGPS* need to switch off the Bluetooth/GPS device to avoid being tracked. More fine-grained systems exist, where the users can visualize and edit boundaries and resolution of the collected information before sharing it with others [HL04, KHFK07, KBD$^+$10, TCD$^+$10].

Location sharing is a multifaceted topic. Users today can instantly share their location through an increasing number of services: using native applications (Foursquare, Google Latitude), by means of social network websites (Facebook, Twitter) or within social experiments [YL10, MLF$^+$08, KO08, TCD$^+$10, LCW$^+$11]. The attitude of users towards location sharing varies, it has been established that their understanding of policies and risks is quite limited and often self-contradictory [SLC$^+$11, LCW$^+$11, TCD$^+$10, Duc10]. Although the majority of users seem to be at ease in sharing their check-ins, they assert to be worried about the *Big-Brother* effect when asked directly[RGKS11, FHE$^+$12, EFW12]. These contradictions and the natural complexity of different preferences in location sharing policies raise challenges for the researchers. First of all, we find that better ways to inform the users about possible attacks are needed (see section 4 and 6). Secondly, we believe that new dynamic platforms should be created to enable users to visualize and control their information flows. For example, how to discern which areas can report user's location and which can not [TCD$^+$10]. To reduce the risk of being misused as a stalking tool, *Confab* for example trades usability for privacy: whenever a user requests other user's location, the latter receives a notification (time constraint) and queries are allowed only if both users are located in the same building (space constraint).

Sharing with researchers. As we have already showed, the most common approach for CSS studies is to collect users data in centralized servers and then perform analysis on it. This way, when the information leaves the smartphone, the user effectively loses control over it and can not be *technically* sure that his privacy will be safeguarded. The opposite approach is to let the users *own* the data so they can control, access, and remove them from databases at any moment, as noted in section 5. A possible way to achieve this is to upload the data from the mobile devices not to a single server, but to personal data stores (e.g.: personal home computers, cloud-based virtual machines) as shown in the architectures in [HL04, GBL06, SLC^{+}11]; it is then possible to deploy distributed algorithms capable of running with inputs coming from these nodes, as illustrated by the *Darwin* framework.

While the advantages in cloud computing for CSS frameworks are numerous (as seen here and in section 5.2), this architecture is not immune from privacy concerns for users and technical barriers for scientists. The first are worried about the confidentiality of their *remote* information, the latter need practical ways to collect and perform analysis on the data.

7.2 Data Control

The control of the ownership of the digital data is difficult. Whenever images are uploaded to photo-sharing services or posts are published on social networks, the user loses the direct control over the shared data. While legally bound by the usage agreements and terms of service, service providers are out of users' direct control. Among the open problems in data disclosure we find *time retention* and *information flow tracking*. Today's frameworks try to solve these issues by *legal means*, such as declarations asserting that "any personal data relating to the participants will not be forwarded to third parties and will be destroyed at the end of the project" [KN11]. In this chapter we show the state-of-the-art of the *technical means* to limit information disclosures and possibilities of integration with CSS frameworks.

Information Flow Control *Information Flow*(IF) is any transfer of information from one entity to another. Not all the *flows* are equally desirable, e.g. a sheet from a CIA top-secret folder shouldn't leak to any another file of lower clearance (a public website page). There are several ways of protecting against information leaks. In computer science, Access Control (AC) is the collection of network and system mechanisms that enforce policies to actively control how the data is accessed, by whom, how and who is accounted for. In softwares, a first approach is the active defence that strengthens the program to limit information flow *before* the leak happens - *a-priori*. Language Based Security techniques and languages are the result of these studies. The complementary approach becomes effective *while* and *after* the leak already happened - *run-time* and *a-posteriori* defences. In this case, the system tries to understand how the data is being (was) accessed, by whom, how and who is (was) interacting with it. Even if this isn't an active approach, it still acts as a defence since the attackers are (usually) conscious know that their behaviour can be traced back. It acts as a deterrent. So, "instead of enforcing correct behaviour through restrictions, *auditing* deters individuals from misbehaving by creating an indisputable log of their actions (unauthorized uses and privileges abuses)". The problem of information flow inside one program has been thoroughly studies in the latest 40 years ([Den76, ML97, VIS96]) and resulted in the development of *language-based* techniques that try to detect/track/understand and avoid unwanted information leak in programming languages, e.g.: data read from a highly sensitive file and written to a less restricted one. *JIF* - a famous example of security-typed programming languages - is an annotated Java extension that checks confidentiality and integrity labels for command and variable in a program before and during its execution. These languages are a useful method to avoid unwanted information disclosure, but they a) require that software developers know in advance every allowed information flow; b) need labels to be manually added to commands and variables; c) can track information flow within a single program. HiStar [ZBWKM06], Asbestos [EKV$^+$05], DBTaint [DC10] and others are Operative Systems (and OS extensions) specifically designed to perform IF control among different processes on the same host. Applications can create "*taints*" that block threads from reading from objects with higher tainting values and equally block threads from writing to file with lower tainting values. While these systems can protect a software from other hostile programs on the same machine trying to steal/modify sensitive data, they do not comply with applications communicating on different machines. Some attempts of building *Decentralized Information Flow Control* systems (DIFC) to track information flows in distributed environments are *JIF* extensions (such as *Jif/Split* [Zda02] and *CIF* [SARL10]) or *HiStar* extensions like *DStart* [ZBWM08], which utilizes special entities at the endpoint of each machine to enforce information exchange control. An interesting approach is taken by *Neon* [ZMM$^+$10] which can control not only information containers (such as files) but also the data that they actually contain (data written inside the files). It is able to track information flows involved in everyday data manipulations such as *cut and paste* from one file to another or file compression. In these cases, privacy policies about participants' records stored in datasets can not be laundered, on purpose or by mistake. *Neon* applies policies at the *byte-level* so whenever a file is accessed, the policy propagates across the network, to and from storage, maintaining the binding between the original file and derived data to which the policy is automatically extended.

Another improvement in managing personal data in CSS studies can be to give participants a closer control of their data, for example letting them select their own policies for data treatment and disclosure. *Garm* [Dem11] is a data provenance analysis tools that allows users to share sensitive files exactly in this way. Special files and encryption are used to determine which data sources are used to derive other files, therefore protecting data from accidental disclosure or malicious tampering. The access policies that contain confidentiality and integrity constraints are defined directly by the users and are enforced across machine boundaries. *Privacy Judge* [KPSW11] is a browser addon for *Online Social Networks* to control access to personal information published online which uses encryption to restrict who should be able to access it. Contents are stored on cloud servers in encrypted form and place-holders are positioned in specific parts in the OSNs layouts. The plugin automatically decrypts and embeds the information if the viewer has been granted the access. The domains of *Garm* and *Privacy Judge* can be extended by similar tools to limit access and disclosure of personal datasets: the participant could remove one subset of his entries from one dataset, affecting all the studies at once.

The complementary approach to the above systems is to ensure control of information disclosure *a-posteriori*. This means that whomever is in possession or processing the data can be supervised by the users, and therefore each misuses or unwanted disclosure can be detected. Among these *auditing systems* we find *SilverLine* [MRF11], a tracking system for cloud architectures that aims to improve data isolation and track data leaks. It can detect if a dataset collected for one experiment is leaked to another co-resident cloud tenant. Therefore, users can keep under direct control where their personal information is stored and who has access to it. *CloudFence* [PKZ+12] is another data flow tracking framework where users can define allowed data flow paths and audit the data treatment monitoring the propagation of sensitive information. This system - which monitors data storage and service providers (any entity processing the stored data) - allows the user to spot deviations from the expected data treatment policies and alert them in case of privacy breaches, inadvertent leaks, or unauthorized access. For maintenance reasons (backups, virus scanning, troubleshooting, etc.), cloud administrators often need privileges to execute arbitrary commands on the virtual machines. This creates the possibility to modify policies and disclose sensitive information. To solve this inconvenience, *H-one* [GL12] creates special logs to record all information flows from the administrator environment to the virtual machines, therefore allowing the users to audit privileged actions. Monitoring systems like *Silverline*, *CloudFence*, and *H-one* can be deployed for CSS frameworks to give the users a high degree of confidence in the management of their remote personal information stored and access by cloud systems.

Unfortunately, these solutions are still not easily deployable since a) many of them require Trusted Computing Bases [KYB+07, SLC+11, Dem11] (to prove trusted hardware, software, and communications) which are not common at the time of writing; b) some requires client extensions that reduce usability and might introduce new flaws [KPSW11]; c) covert channel attacks are not defeated by any IF techniques (e.g. screenshots of sensitive data). In addition, enforcing information flow policies also need to take in account incidentally (and intentionally) human malpractices that can launder the restrictions. We remark that *none* of the CSS frameworks surveyed provided Information Flow controls, and only few of them mentioned auditing schemes. Finally, it is our belief that the future of CSS will surely benefit from a paradigm shift in data treatment where users will *own* their personal sensitive information.

Data Expiration As we have already seen with the recent cases of Google Street View and Facebook, service providers are very reluctant in getting rid of collected data. After user deletion requests, Service Providers prefer to make the data inaccessible - hidden from users' view - instead of physically purging the data from their databases. To aggravate this situation, data is often cached or archived in multiple backup copies to ensure system reliability. Therefore, from users' perspective it is difficult to be completely certain that every bit of her personal information has been deleted. Consequences of unlimited data retention can be potentially catastrophic: "if private data is perpetually available, then the threat for user privacy becomes permanent" [CDCFK11]. The protection against this can be *retroactive privacy*: meaning that data will remain accessible until – and no longer than – a specified time-out.

Here we illustrate some of the most interesting approaches to address the *data expiration* problem, narrowing our focus on systems that can be integrated in CSS frameworks. The criteria in the selection are a) user control and b) ease of integration with existing cloud storage systems. This choice is motivated mainly by the anticipated evolution of the privacy-aware CSS frameworks: a closer user involvement in personal data management and the use of the cloud services, such as virtually unlimited storage space, ubiquitous service availability, and hardware and protocol independence. Cheap storage prices and ubiquitous Internet access increase data redundancy and dispersion, making almost impossible to ensure that every copy of an information has been physically deleted from the system. Therefore, self-destructing data systems prefer to alter data availability instead of its existence, securing data expiration by making the information unreadable after some time. *Boneh* and *Lipton* pioneered the concepts of "self-destructing data" and "assured data deletion" in [BL96]. First, data is encrypted with a secret key and stored somewhere to be accessible to authorized entities. Then, after the specified time has passed, the corresponding decryption key is deleted, making it impossible to obtain meaningful data back. This is a trusted-user approach which relies on the assumption that users does not leak the information through side channels, e.g.: copying protected data into a new non-expiring file. Therefore, these systems are not meant to provide protection against disclosure during data lifetime (before expiration), as *Digital Rights Management* (DRM) systems try to do instead[1]. Self-expiring data systems can be integrated in CSS frameworks to enhance privacy in sharing data, permitting the participants to create personal-expiring data to share with researchers for only a predefined period of time.

Key management – becoming the main concern – can be realized either as a centralized *trusted* entity holding the keys for all the users or keys can be stored across different nodes in a distributed network where no trusted entity is in charge. *Ephemerizer* [Per05a, Per05b] extends the principles outlined in [BL96] to interconnected computers implementing a central server to store the keys with respective time-outs. The server checks the keys periodically for their time-out and delivers if their time is not yet expired. An approach that avoids the necessity for a trusted party is *Vanish* [GKLL09], a distributed solution that spreads the decryption key bits among different hosts: after the usual encryption phase (key creation, file encryption and storing), the key is split into secret shares and distributed across random hosts in a large Distributed Hash Table (DHT). DHT are decentralized systems that store $< key, value >$ mappings among different nodes in a distributed network. The key tells which node is holding the corresponding *value/piece of data* allowing the value retrieval given a key. Accordingly to the secret sharing method [NS95], the recovery of a k (threshold) shares on n total shares permits the reconstruction of the original key and therefore the decryption. What makes the data to expire/vanish is the natural turnover (churn) of DHTs (e.g.: Vuze) where nodes are continuously leaving the network making the pieces of a split key disappears after certain time. When there are not enough key shares available in the network, the encrypted data and all its copies become permanently unreadable. Two are the main limits of Vanish system. First, the requirement for a plug-in that manages the keys reduces its usability. Secondly, the time resolution for expiration is limited on 8 hours - the natural churn rate of the underlying DHT - and expensive to extend due to re-encryption and key distribution.

As pointed out in [WHH$^+$10], the clever idea of turning the nodes instability into a vantage point for data expiration might introduce serious problems. To break data expiration it is enough to continuously crawl the network and pre-emptively harvest as many stored values as possible from the online nodes before they leave the network. Once enough raw material has been collected the attack rebuilds the decryption key, resuscitating the data. Based on the same cache-aging model of Vanish, but immune from that attack, is *EphPub* [CDCFK11] where the key distribution mechanism relies on the Domain Name System (DNS) caching mechanism. The key bits are distributed to random DNS resolvers on the Internet which maintain the information in their caches for the the specified *Time To Live*. This solution

[1]DRMs assume user untrustworthiness limiting the use and/or disclosure of a digital content in all its possible forms e.g.: duplication and sharing.

is transparent to users and applications, not involving additional infrastructure (a DHT or trusted servers) nor extra software (DHT client). Another solution for data expiration is *FADE* [TLLP12] a policy access control system that permits users to specify read/write permissions of authorized users and applications other than data lifetime. The underlying idea is to decouple the encrypted data from the keys: information can be stored in untrusted cloud storage providers and a *quorum of distributed key managers* guarantees distributed access control permissions for the established period of time.

Given data redundancy and dispersion, it is almost impossible to ensure full control over distributed data, especially when users are directly involved[2]. While everlasting data is generally dangerous in any context, the problem becomes even more important for CSS studies, where the amount and the sensitivity of the collected data can be substantial. The described systems can be used for building complete privacy-aware CSS frameworks that can automatically take care of purging old information from the database. Procuring the users with ways to control sharing schemes and information lifetime might attract more participants, who may currently be reluctant in providing their personal data. We would like to emphasize that the mentioned solutions do not provide complete data protection and have been inspected by the scientific community for only a brief period of time. It is not current practice in the examined CSS frameworks to include the data retention procedures and lifetimes in the user agreements or informed consent. While it is still uncertain whether assured deletion and data expiration are technically secure, we are certain that there are limits beyond which only legal means can guarantee the users the conformity to certain procedures in data management and retention.

[2]"It cannot be prevented that human users manually write down the information, memorizes it, or simply take a picture of the screen" and share it in a non-secure manner, as stated in [KPSW11].

Chapter 8

Privacy Actionable Items

In this Chapter we present the executive summary for the CSS practitioners.

Regulations. When a new study is provisioned, it must follow the regulations of the proper authorities. Special attention should be given to the cases where the data may travel across the borders, either as part of the study operation (e.g. data storage) or as a part of research collaboration with foreign institutions. The requirements and guidelines may differ significantly between countries, additionally if the data collection happens in one country and analysis of the dataset happens in another, the data analysis may not be considered *human subjects* study, thus not requiring IRB approval. The regulations and guidelines of the country where the study is conducted, reflect expectations of the participants regarding their data treatment. Researchers need to make sure that those will be respected, even when the data flows across the boarders.

Informed Consent. Informed consent is the central point of the participant - researcher relation. We strongly encourage the publication of the informed consent procedure for the conducted studies, so the best practices can be build around it. We should be working towards the implementation of *living informed consent*, where the users are empowered to better understand and revisit they authorizations and relation with the studies and services in general. This relation should ideally last for as long as the user's data exist in the dataset. As new techniques of data analysis are introduced and new insights can be gained from the same data, the participants should be made aware and possibly in charge of the secondary use. Additionally, we envision a better alignment of business, legal, and technical dimensions of the informed consent, where the user's act of consenting is not only registered for legal purposes, but technically creates the required authorizations (e.g. OAuth2 tokens).

Security. The security of the data is crucial for ensuring privacy. Moving into the cloud may require close examination of the infrastructure provider's policy, as well as the technical solutions limiting the access to the data in the shared-resources environment. One of the solutions is to encrypt the data on a server physically owned by the research unit conducting the studies, and only then pushing it into the cloud. There are different levels of encryption granularity that may be used; The less structure is preserved in the data, the less information can potentially leak, but at the same time, less meaningful queries or data processing can be executed on the encrypted data. It may be an option to encrypt (or one-way hash) only PIIs, keeping the structure of the data and certain raw values (e.g. location coordinates); this allows for effective data querying in the database, but at the same time can expose the information about the participants (the cloud learns where the participants are, but doesn't learn their names or other PIIs). If the data is encrypted in it's entirety, effective queries become difficult and may not be feasible.

Privacy Implementation. The data collected is only valuable if it can be analyzed. It is often desired or necessary to share the data in some form with the 3rd parties, some of which may be hostile. As it has been shown in multiple cases, the de-identification, introducing the noise, implementing $k-anonymity$ etc. may be insufficient against an educated and determined attacker with Internet access (section 6.2). We can expect this problem to grow, as more publicly available information is exposed by the participants in different (non-study related) channels. If it is sufficient to know several locations to uniquely identify the user [dMHVB13], all the other data gathered in the study can be then also linked to the real person. Firstly, we suggest that good practice is to make the type of de-identification performed on the dataset public, helping to establish common practices and understanding of what works and what does not. One of the possible directions is to move away from copying the entire datasets for later offline analysis, which is becoming impractical anyway, due to large size and real-time nature of the data. Instead, researchers and other service can interact with the data through APIs, that allow for control and accountability. If the data dimensionality is additionally reduced before flowing through the API (e.g. city-level location of the user rather than raw GPS trace), privacy can be managed in a more robust way.

Part II

Perfect World

Chapter 9

sensible-data

Latest events abruptly brought to people's attention the problematic relation between Big Data and information privacy management [GUA]. and consequences of such events might affect the scientific community. Computational Social Science (CSS) is an interdisciplinary research process that gathers and mines wealth of *sensitve* data to study human behaviour and social interactions. *"Should scientists fail to defend participant's rights and their data, even in a single case, the consequences for CSS as a field could be catastrophic. A breach of security or abuse of sensitive data, could result in a loss of public confidence and — as a consequence — a decreased ability to carry out studies."* People, who were already reluctant to provide their information for scientific purposes, will now become even more unwilling. This is a turning point for the future of big data and scientific research, a critical moment to avoid a domino effect.

The next paragraphs will show design principles and practical applications of an ongoing project that seeks to establish *new* standards in information privacy management: *sensible-data*.

9.1 Service

A collaboration between DTU and MIT universities gave birth to "*sensible-data*", a service that collects, stores, manipulates, *studies*, and *shares* Big Data. What makes it different from other solutions are the designing principles: *sensible-data* does not only strengthen the security of the data, but it specifically focuses on the respect users' *privacy*. Among the main goals, data treatment *transparency*, fine-grained *control* in data collection/distribution, and *anonymization*. The combination of versatility, modularity and scalability make the integration between the central service and the data sources (e.g.: web applications, databases and mobile apps) highly customizable. The platform can be adopted for different purposes, from scientific studies, to data analysis in e-Commerce, including secure data storage for government institutions. For these reasons, *sensible-data* can be seen as the first realization of a privacy-aware solution in users' data treatment.

Part I showed that *Computation Social Science* is a very active field of research that relies on the mixture of social studies and nowadays technologies (e.g.: smartphones, Online Social Networks). It helps in understanding causes and effects in human behaviour, it gives insights in their interactions, and can explain the inner nature of relationship. CSS's observations and deductions can be adopted in various contexts and can have huge consequences on people, positively affecting their lives. What CSS studies need is the participation of a variegate and large set of people so that studies become more precise and accurate. *Part I* also demonstrated that there is still space for improvements in users' privacy management. Before any *"catastrophic event"* endangers the image of CSS, scientists need a *new* way to do conduct studies in the field.

sensible-data can provide such solution allowing the researchers to conduct social experiments in a privacy-aware manner. More in detail,

- scientists can create studies with fine-grained acceptance conditions instead of "all-or-nothing" approaches that forces participants to accept all the study conditions or to be left out;
- they can monitor the data streaming in/out of the study in real-time;
- they can share datasets and study results in a seamlessly way.

On the participant perspective, the service will improve the management of their sensible information, making easier to understand what is happening to their data. Study participants will be able to:

- grant authorization to third party applications;
- monitor and control their personal data;
- share data from one study to another in a easy way.

An important aspect of *sensible-data* is the concept of *living informed consent*, deeply outlined in *Section 4*. The realization of this concept will allow the participants to *dynamically* change study permissions and to be informed about data treatment/disclosure policies in real-time.

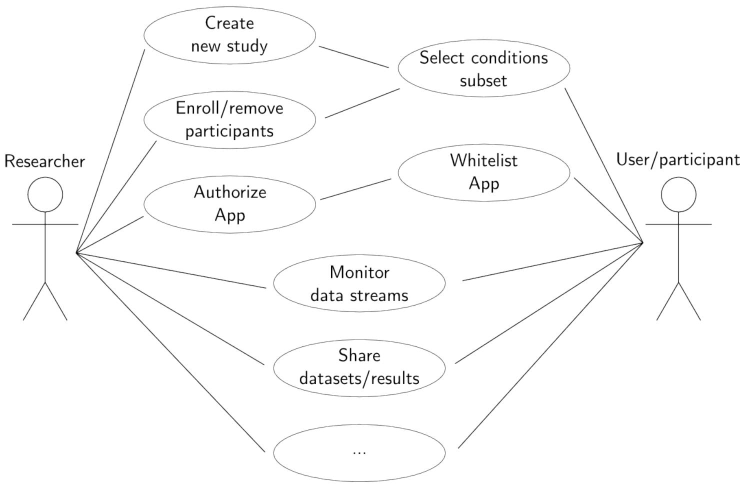


Figure 9.1: *sensible-data* use cases

9.2 Design

sensible-data is a system composed by three layers: Platform, Services and Applications (Apps). Third parties' **applications** gather data from web services and smartphones (e.g.: Facebook app, Funf) and interact with customizable services. The **services** are specialized functionalities that can be performed using *sensible-data* system. These can be study research projects, election polls, or business market researches. **Platform**'s components are (see *Figure 9.3*):

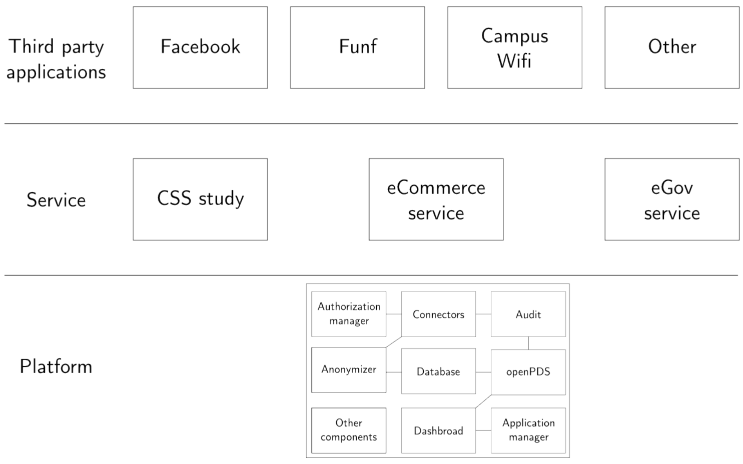


Figure 9.2: *sensible-data* system.

- **Authorization manager**. It holds the user credentials and accounts for the system.
- **Connectors**. Applications use connectors to provide or access the data. These endpoints can be directly managed by the user to regulate the data flows. They are part of system APIs that allows *services* to communicate with *apps*.
- **Audit**. Module that securely records all the events/interactions between users, *apps* and the *system* (e.g.: data flows in/out, users' app authorization, etc). This module will be thoroughly detailed in *Section 13*.
- **Anonymizer**. A Module for anonymizing the data before pushing it to the database.
- **Database**. Physical long-term storage for the data.
- **openPDS**. It is a Personal Data Storage container for the information collected by the Apps. Users can manually review/modify/remove sensible files via web browses and smartphone applications.
- **Dashboard**. Graphical interface that allows the users to monitor and control the flow of information from/to App to/from the System.
- **Application manager**. Module for managing the authorizations between the registered applications (Apps) the *sensible-data* service.

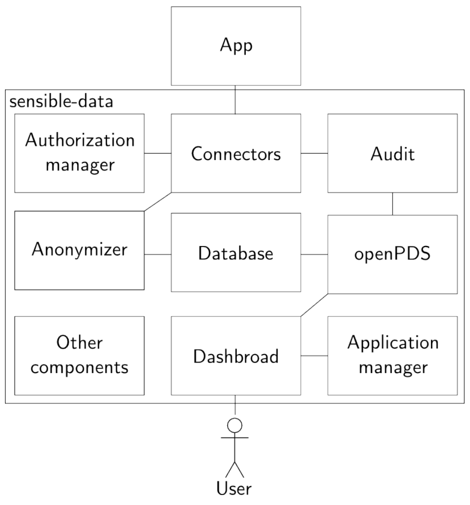


Figure 9.3: *sensible-data* platform.

9.3 Implementation

In its current implementation, *sensible-data* system is a Django web-based solution which provides its services through REST APIs. The system is hosted on a Rackspace server with backed databases on MongoLab (Database As A Service for MongoDB) that will be replaced in the future by custom instances. User authentication is now managed through the Django authentication system, by which each user has is own account registered and authenticated. In the future, *sensible-data* will offer authentication through OpenID open protocol. App authorizations are granted using OAuth2.0.

Part III

Auditing

Chapter 10

Introduction to Auditing

Loggers record the dynamics of a system, keeping track of the noteworthy past events that changed a system's state. Each (digital) record includes when an event occurred, which entity triggered it, and which resource it involved. Examples of such events are users' activities, program execution status, resources usages, data changes, and so on. Logging systems are useful tools for troubleshooting problems, identifying policy violations, enforcing accountability and generally to monitor interactions with resources.

There are scenarios in which logs must be available for public inspection. In these cases, the number and the identities of the verifiers and the frequency of the verifications highly affect the design of logging system. When record reviewing is a fundamental feature, *logging* becomes *auditing*.

Logging systems can be adopted as security components to record – in secure manner – the events in an environment ("*a-posteriori*" mechanism). The quality and the quantity of the retained data make logs important sources of digital evidence for reconstructing complex events, detect intrusions, and investigate malicious activities. For these reasons, they need to be protected from misuses.

Although logs are usually kept behind secured systems and adequately protected, it cannot be guaranteed that those defences will never be compromised. Therefore, it is necessary to strengthen auditing systems to resist security breaches. Unfortunately, once the machine has been taken over by an intruder, there is not much that can be done to prevent him from corrupting the audit trail at his pleasure. On the other hand, it is possible to limit the attacker's ability to corrupt the records generated *prior* the machine's compromise (*tamper resistance*) through the use of *Trusted Computing Modules* or *Write Once Read Many* (WORM) storage disks. Without recurring to such expensive and uncommon solutions, it is also possible to build *tamper evident* audit trails. These are logs that make it impossible to undetectably alter pre-compromise entries. *Integrity* verification of audit logs becomes the primary security requirement for the *secure* auditing systems.

Outsourcing logs Long-term storage of logs can be expensive both in terms of physical resources and integrity revision processes. Therefore more and more often organizations are delegating the management to the cloud, sending the data to remote hosts before purging the local copies [RBS$^+$12] (*Logging As A Service - LaaS*). Outsourcing the maintenance of logs can reduce costs, but remote auditability requires further enhancements to integrity and authenticity. When the owner of the hosting machine is not the same owner of the secrets within the device it is essential that audit mechanisms are able to detect attempted fraud. Remote auditing can benefit from replication, a good measure to safeguard data integrity trough redundancy, based on the assumption that an attacker will not able to compromise all the remote hosts. On the other hand, data replication rises *confidentiality* issues, since all the multiple copies of the audit trails must be secured against unauthorized viewers.

10.1 Definitions

- **Auditing system**. System that comprehends all the components for auditing.
- **Auditing, auditing process**. In computer technology, auditing is defined as the process of storing and examining digital records regarding the events happening in well-defined environment constituted by a set of entities. Typical examples are: the list of read/write operations occurring on a machine's operative system or message transactions among different end-points in a network[1]. The purpose is to collect enough information to be able to carry an assessment to identify and quantify future environment's characteristics.
- **Log event, audit event, event**. With event is defined an occurrence observable in the monitored environment. A logged event is the digital representation

[1]Historically, by auditing, is intended the *"examination of data belonging to an entity performed by an (external) independent examiner"*.

of the event once appended to the audit trail. The set of which log events have to be stored is defined at system design. The format of the logged event must be concise but rich enough for later analysis.

- **Audit, audit trail, audit log, log trail, trail**. It is some form of non-volatile memory such as a local disk or database or a remote log storage that holds the events and can be later verified.
- **Auditor, examiner, verifier**. The entity which performs the examination of the audit trail.
- **Auditing report, report**. Result of the auditing examination process.
- **Append, write, update, store, logging, registration, recording**. Action of appending a new entry at the end of the audit log.
- **Verification, integrity check, check, inspection, review**. Action of assessing the integrity of the audit trail.

Chapter 11

Auditing Model

This section describes the how an auditing system can be modelled. As all the models, it is an abstraction from reality, a representation of how an ideal auditing system should appear.

11.1 Auditing Protocol

Auditing Types. The **Auditing process** can be scheduled according to predefined time-triggered procedure (*Time-driven*) or performed ad-hoc, when a specific event requires the generation and the inspection of the audit trail (*Event-triggered*).

Auditing Trail Format. The **Audit trails** can be *sequential*, when every event is appended to an ever increasing audit trail or *circular*, where the maximum length of the audit trail is defined and when reached, new entries overwrite old ones.

Event Format. **Log event**'s format are not required to have a specific format, but it must be defined in a way that a later reader will unambiguously understand.

Design has important consequences in the implementation. In fact, it affects memory consumption (storage) and performances (time) in logging and retrieving. For these reasons, it is crucial to determine the necessary and sufficient format for both trail and log entries. Common fields are:

- **Timestamp** : identifies when the event occurred. This is usually expressed in the format of date and time of day, with different degrees of accuracy (i.e.the granularity can variate from hours to milliseconds).
- **Description** : human readable description of the event, used for later trail inspection.
- **Entry type** : tag used to classify different events under the same category. Useful during inspection operations for fast retrieval (e.g.: administration event, data event, ...).
- **Severity level** : keyword that defines the severity of the event. Common cases, in increasing order are: debug, informational, notice, warning, error, critical, alert, emergency.
- **Author** : is the entity that writes into the audit trail.
- **Id** : event unique identifier.
- **Other** : optional information peculiar to the particular auditing case.
- **Security fields** : optional fields regarding confidentiality, integrity checks or other security properties.

Roles

- **Event generator** : the entity/device that creates the event.
- **Relay** : (optional) is a proxy that forwards the events from devices to collectors.
- **Event receiver** : receives the events.
- **Filter** : decides whether the log event must be recorded or not according to the rules specified by the auditing designer.
- **Logger**: generates the corresponding log file (also called audit trail). It might apply cryptographic operations to protect recorded entries against illegal reading, deletion, modification.

- **Verifier, auditor**. This principal asserts the authenticity of the audit trail and/or singular log entries.
- **Audit trail**. The physical file holding the log records.

Trust With trust is often meant *"the assumption that an entity will behave substantially as expected"*. A trust level is the degree to which other participants in the system trust this entity. Whenever a security assessment is to be made, these labels declare security assumptions.

- **Trusted entity**. A trusted entity is defined as *"an independent, unbiased party that contributes to, or provides, important security assurances that enhance the admissibility, enforceability and reliability of information in electronic form."*[46-]. For secure auditing, it means that this entity can not be compromised. Empirical evidences show that this property is very hard to hold, therefore a more realistic definition for a trusted entity could be: an entity which is "*reasonably* secure from intrusion and misuse"; which can provide a "*reasonable* level of availability, reliability, and correct operation". The whole system relies one the fact that this entity will always behave according to the established policies and protocols, in other words, its behaviour does *not* vary. Other relying parties use this entity to secure their interactions, exchange messages, and keep secrets.
- **Untrusted entity**. This entity is not physically secure or sufficiently tamper-resistant to guarantee that it will not be taken over by a malicious user.
- **Semi-trusted entity**. It often presents some of the trusted entity properties, such as "*reasonably* secured from intrusion", but the likelihood of being successfully compromised is higher. It has granted only with limited privileges and not trusted as far as holding key material. If compromised (or if behaving in an unforeseen manner) it does not compromise the security of the entire system.

11.2 Properties

This section describes the properties of an ideal auditing system.

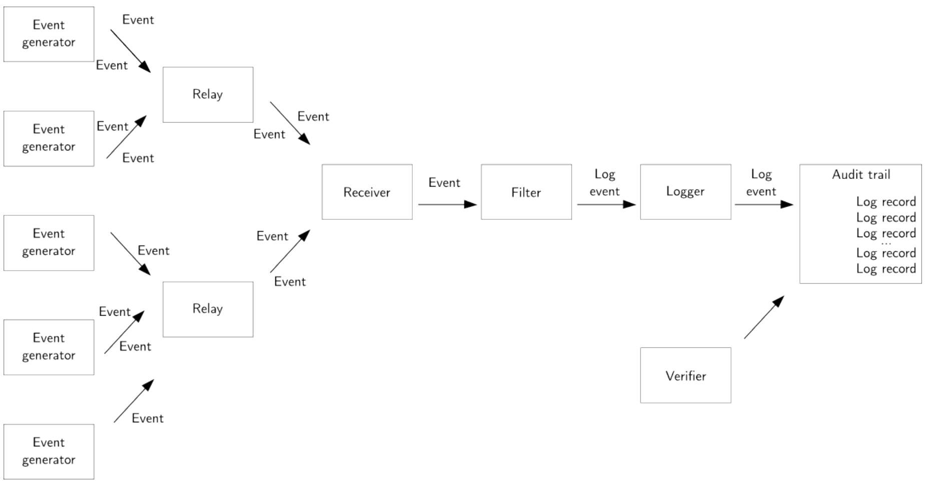


Figure 11.1: Auditing roles.

Integrity. The property of maintaining the consistency of data over the entire life-cycle[1]: data cannot be modified in an unauthorized manner. For auditing, this can be translated into the fact that audit trails cannot be changed once recorded. Integrity can be further analysed into atomic properties, such as *accuracy* (entries are not modified), *completeness* (entries are not deleted), *compactness* (entries are not illegally appended), *uniqueness* (each message is logged once). It is important to extend these properties from the single entries to the whole audit trail, in other words, re-ordering should not be possible (*stream-integrity*). To recap, it is not possible to (over)write, delete, add, re-order (part of the) entries and the whole audit trail. Unfortunately, such a property is almost impossible to achieve: no security measures can protect an audit trails *after* an attacker has gained control of the hosting machine. The intruder will be able to force the system - and so the auditing mechanism - to act as he wants. In this case, it will not possible to prevent the attacker from adding new log entries, alter or discard new events coming before they are store in the trail. Nevertheless, strong statements can be claimed about log entries made before the compromise. What an auditing system can achieve is **Forward Integrity**. Proposed by [BY97], it assures that pre-compromise data will be safe from post-compromise insertion, deletion, modification, and reordering. In other words, the integrity of the log entries accumulated *before* the attack should always be protected even when an adversary successfully compromise the machine[2].

[1]Without going to into details, with consistency is intended that data does not contain contradictions.

[2]Forward is a misleading word, since the integrity actually pertains to the past, but has been chosen by Bellare and Yee according to security literature.

Verifiability. Practically, forward integrity is hard to achieve since *tamper resistant* computers are not common and almost every system can be broken. What auditing system provide instead is *tamper evidence*: audit trails can not be modified without detection. By verifiability is intended the possibility to be able to check the integrity of the audit trail[3]. The dimension of the auditor set determine if an audit has *restricted verifiability* or is public verifiable. In fact, audit records might need to be made available to a great number of unknown outside auditors for public auditing (e.g.: public financial auditing, electronic voting, ...).

Accountability. Audit log need to include information about which entity is responsible for adding the entry to the trail. Accountability means that it is always possible to determine the author of an appended entry.

Searchability. Audit logs are a mixture of archives and backups, since read and write capabilities must be considered in addition to the respect of the integrity property[4]. Events are automatically appended to trails, without user intervention. Verification on the other hand might require human interaction.

Confidentiality. While confidentiality is not a strictly mandatory requirement for auditing, it is often needed for sensitive data. Only legitimate users should be allowed to browse the log entries. Usually, confidentiality is achieved by authorization lists and/or encryption mechanisms (see *Section 11.3*), which can negatively impact searchability.

Append-only. Property that guarantees that new entries will be always added at then end of what already present.

11.3 Cryptographic Primitives

Cryptographic primitives are building blocks used to create security systems and protocols. They are used for encryption algorithms, hashing functions and digital signatures schemes. Although many of the following concepts have vast application scopes, greater relevance has been given to characteristics regarding auditing system designs.

[3]Integrity by itself does have not much importance, if it can not be verified.

[4]Archives rely on good browsability, with fast searches, normal writing and integrity. Backups are for restore, so they can write and read slow but data integrity is fundamental.

11.3.1 Cryptographic Key

Key. A key is an auxiliary input parameter for cryptographic algorithms. For the same input message, a cryptographic algorithm produces different outputs according to different keys. Concrete examples are encryption/decryption cipher schemes, digital signatures and message authentication codes. According to a widely accepted security engineering principle, *"system security must depend only on the secrecy of the key, and not on the secrecy of the algorithm"*[5]. Giving the importance of keys, their management (generation, exchange, storage, use, replacement, expiration, deletion) is crucial to the security of a system. If an attacker obtains the key, cryptographic systems cannot hold security properties.

Evolving Keys. As explained in [Fra06] a viable way to limit the damage of key exposure, is to change the key over time. In this way, even if an attacker learns the secret, he will be only able to operate on messages belonging to the timespan in which the key was kept the same. Keys are usually evolved by non-reversible functions that have as input the previous key.

11.3.2 Symmetric Key Algorithms

Algorithms that use the same key for encryption and decryption are called symmetric-key algorithms. Encryption algorithms needs to be reversible:

$$D(E_k(m), k) = m$$

A first example of symmetric key algorithm are **stream ciphers**, where each bit of the original message is encrypted with the corresponding bit in the key to output the ciphertext; **block ciphers** instead, take as input a fixed-size block of data and output another fixed-size block according to a key. Symmetric encryption schemes, require that the two involved parties pre-share a secret key to maintain their communication private. For logs which contains sensitive data, the entries can be encrypted using a secret key shared between the verifier and the logger. Most commonly used symmetric encryption algorithms are 3DES and AES.

Message Authentication Codes. MACs are symmetric constructions that detect message alterations. While encryption provides confidentiality, it cannot prevent

[5]This is a re-formulation given in [FSK12] of the Kerckhoffs's principle "*Cryptosystem should be secure even if everything about the system, except the key, is public knowledge*" and Shannon's principle "*The enemy knows the system*".

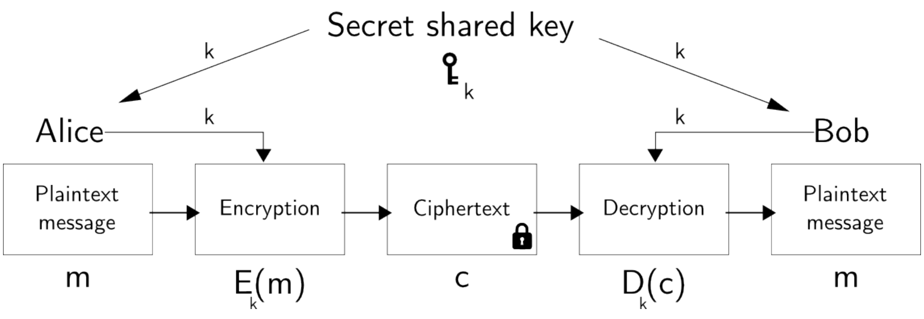


Figure 11.2: Symmetric encryption.

manipulation. When the sender wants to communicate with another entity, it computes the MAC of the message using his key and sends this MAC tag with the message to the receiver. The receiver recomputes the MAC of the message - with the key that he also possesses - and determines if the message is genuine comparing the new calculated value with the received one.

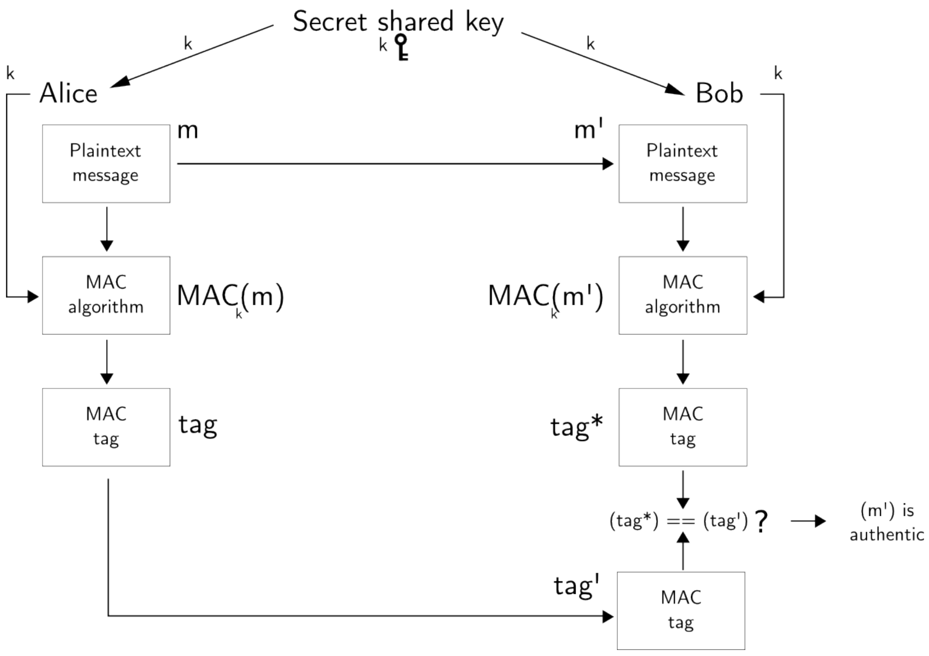


Figure 11.3: Message Authentication Code verification.

For an attacker, it is computationally hard to forge a message and the relative tag so that they would appear genuine – without knowing the secret key.

11.3.3 Hashing Function.

Hash Function A hash function is a many-to-one function that maps arbitrarily long input to a fixed-length output (hash value).

$$Hash : \{0,1\}^* \rightarrow \{0,1\}^k$$

Different implementations of hash function can be used for file fingerprinting, error checksum, data structures, password storage, digital signatures[6], etc. To be employed for secure auditing, four properties must be met. It should be computably:

- Hard to find different inputs that generates the same output. That is, it should be difficult to guess a second pre-image whose image colludes with the first image (*strong collision resistance*);
- Hard to reverse the hashing function (to guess the input based on the output);
- Hard to modify the input without modifying the output;
- Easy to apply the hashing function (to go from pre-image to image).

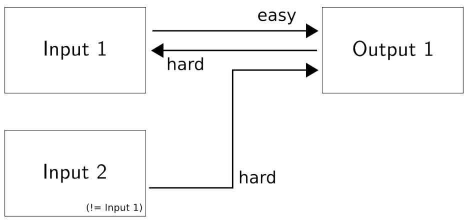


Figure 11.4: Properties of a secure hash function.

If these properties are guaranteed, hashing functions can be used in secure auditing as one-way function to evolve the secret keys or to compute hash chains. Examples of cryptographic hash functions are MD5, SHA-256.

Keyed Hash Function or Hash-based Message Authentication Code functions are MAC algorithms that uses hash functions to provide authentication. The output value is often called *tag* or HMAC value and - sent with the original message - is used to provide integrity (can detect message modifications) and authentication (can show message's origin). Examples of HMAC are HMAC-MD5 or HMAC-SHA-256, according to the cryptographic function.

[6]Instead of calculating the digital signature over a big file, the signature is applied to its shorter hash value.

$$HMAC_{key}(message) : \{0,1\}^k \times \{0,1\}^* \rightarrow \{0,1\}^t$$

with a k-bit key *key* and an arbitrary message *message* outputs a *t*-bit tag.

Hash Chain A hash chain is the successive application of a hash function to a

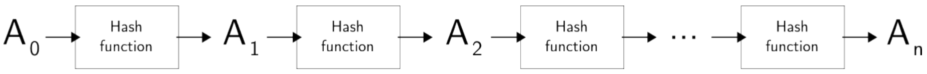


Figure 11.5: Hash chain.

message. In secure auditing, hash chains are employed to provide *Forward Integrity*: they create an interdependency between an entry and its predecessor, linking entries to each other. Since element in the hash chain can be seen as then checksums of the previous entries, the verification of a whole hash chain asserts whether any entries has been modified: if the chain can be traversed without incurring in broken links, it means that elements have not been tampered with. For more details on the various uses of hash chains, [Pag09, Wou12]

11.3.4 Asymmetric key algorithms

or Public-Key Cryptography (PKC) is a family of algorithms that uses two mathematically related keys, one to encrypt and the other to decrypt a message (asymmetric). Each entity has a public key known by everyone and a relative private key kept secret. There are several uses of public-key cryptography for secure auditing:

Public-key encryption. As for the symmetric case, to encrypt log entries when confidentiality is needed. Here, the device creating the log event ensures message confidentiality encrypting it with the verifier's public key and the record is stored in the log. The verifier uses his private key to decrypt the message and check the content. Examples of widely used asymmetric key encryption algorithms are ElGamal and RSA.

Digital Signature. Digital signatures are the public-key equivalent of MACs. The message it signed with the device's private key and the verifier can assess the authenticity of the message using the device's public key. Digital signatures provide integrity (correct decryption of the encoded message digest proves the message has not been tampered with), authentication (the sender has access to the private key, so he is the person associated with the public key), and non-repudiation (there is only one signer, who possesses the correct key and that can compute such a signature, so it proves the message was signed by none other than that him). The most common digital signature scheme is the DSA.

Aggregated Signatures Aggregated signatures are a special kind of signatures that can be aggregated together. It is not the purpose of this paragrah to specify the characteristics of such cryptographic systems, therefore suffice it to say that these special kind of signatures, combines different signatures generated by n signers into a unique aggregate signature that, when verified, simultaneously verifies all the component signatures. The same concept can be extended to aggregate – in a single signature – multiple signatures computed in different periods with different keys by the same signer [BLS01, MT07].

Key Distribution. Asymmetric key algorithms do not require pre-shared secret keys, but are computationally more expensive than their symmetric counterpart. For these reasons, they are often used to create and shared private secret keys to use later with symmetric algorithms 11.3.2. Example: Diffie-Hellman key exchange.

Identity-Based Cryptography. Identity-based cryptography [Sha85] is a type of PKC in which any string can be used as a public key to represent an entity. In this sense, the email address can bee seen as the equivalent of a certificate in the normal PKC. The corresponding private key is generated by a trusted third party (Key Generation Center). Every time an entity need its private key to verifying a sign or decrypt a message, this entity needs to authenticate to the KGC and after it gets the private key. This is one of the main drawbacks of IBC, since the PKG generates the key for the users, it also may sign and/or decrypt messages without their authorization (cannot provide non-repudiation)

11.4 Design

Depending on the underlying cryptography, there are two main design patterns for auditing systems.

11.4.1 Symmetric-key based

Auditing systems built upon symmetric key cryptography use evolving keys, (keyed) hashing functions, hash chains, and symmetric key algorithms. A simplified example of how symmetric auditing work, can be extracted from the *Schneider-Kelsey (SK)* scheme, used as a reference by a number of successive secure logging systems [SK99]. The scheme involves three actors:

- *U*. An *untrusted logging* machine that creates and stores the log records. It replies to *V*'s queries.
- *T*. A *trusted* machine holding a *secret key*. It can verify the log trail by itself or it helps the verifier in a different verification process. Interactions with this entity are minimized.
- *V*. A *semi-trusted verifier*, that interacting with *T*, can verify the integrity of the audit log stored on the logging machine.

Log Entry Format. The particular format of *SK* log entries (see *Section 11.1*) is: (*D*) the event data to be logged; (*Y*) a hash chain element that links to the previous element; (*Z*) a forward secure HMAC (*Z*) computed over *Y* with the evolving key starting with the value A_0[7].

Procotol. A trusted server and the logger share a secret key A_0 out-of-band. The logger *U* computes a forward secure MAC for each log entry and evolve the secret key *A*. All the entities who know the secret are able to verify MACs interacting with the trusted server.

Logging.

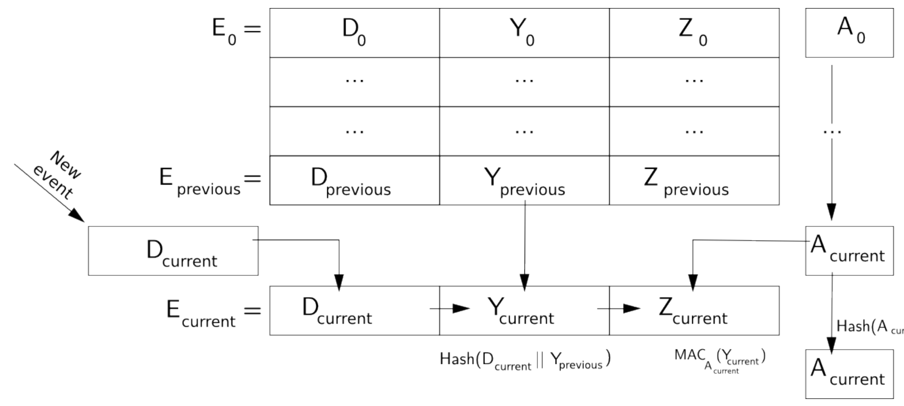


Figure 11.6: Append in a SK-like auditing system.

1. When a new event $D_{current}$ is registered on *U*, the hash chain link $Y_{current}$ is calculated as the hash function of $D_{current}$ and $Y_{previous}$ [8]:

[7] In the original *SK* scheme, *D* is encrypted and an authorization mask provides a fine-grained an access control to the log entries.

[8] Y_0 is a pre-defined seed value.

$$Y_{current} = Hash(D_{current} \| Y_{previous})$$

2. The relative forward secure HMAC $Z_{current}$ is computed over $Y_{current}$ with the current value of the evolving key:

$$Z_{current} = MAC_{A_{current}}(Y_{current})$$

3. As soon as the *Z* value is created, *U* evolves the key $A_{current}$ by means of a one-way hash function and securely deletes the previous one to guarantee the forward integrity property:

$$A_{current} = Hash(A_{current})$$

4. Finally, the entry $< D, Y, Z >$, is appended to the trail.

Verification. There are two different verification systems. The first one involves only the trusted entity *T* and the logger *U*. *T* requires the audit trail from the logger *U* and recomputes *Y* and *Z* values.

One of *SK* scheme's assumptions is that there is no *"reliable, high-bandwidth channel constantly available between T and U"*, therefore the interactions with *T* must be minimized. In the second verification protocol, the semi-trusted entity *V* interacts with *T* – which securely holds A_0 – only to verify the *Z* value of the last entry. More in details:

1. *V* requires to verify records stored on *U*, receives them and goes through the hash chain (recomputing *Y* values) verifying that each link has not been broken.

2. When the last *Y* value $Y_{lastByV}$ value is computed, *V* sends the tuple $< Y_{lastByV}, Z_{last} >$ to *T*[9].

3. Upon reception of the tuple, *T* calculates $Z_{recomputed}$ using its secret key $A_{current}$ with the received value $Y_{lastByV}$:

$$Z_{recomputed} = MAC_{A_{current}}(Y_{lastByV})$$

4. If $Z_{recomputed} == Z_{last}$, the audit trail is correct (the chain is not broken). In any other cases, the audit trail has been tampered with. The answer it sent back to *V*.

[9]It has to be noted that *V* cannot compute Z_{last} since *V* does not possess the necessary key A_0.

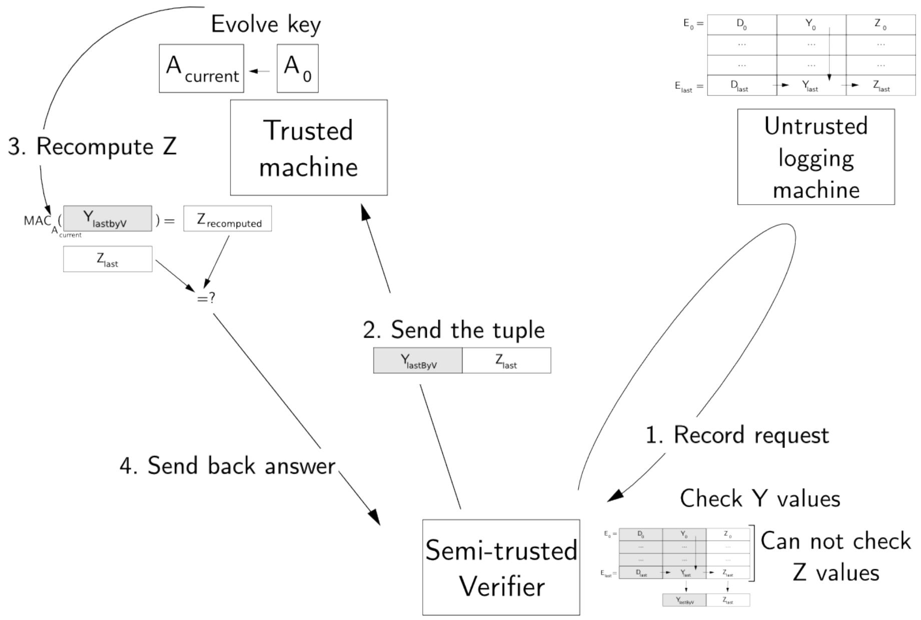


Figure 11.7: Three party verification protocol.

Evaluation

Advantages of symmetric-key based auditing systems:

- Simple to design, implement and deploy;
- Computationally and memory efficient in logging and verification.

Disadvantages:

- No public verification. Only fully trusted entities can hold the secret key.
- No non-repudiation. Any entity possessing the secret key A_0 has the ability to falsify the log entries.
- Verification require online trusted server *T*.
- It suffers from truncation and delayed detection attacks[10] (see later in *Paragraph 11.5*).

[10] A slightly improved version of *SK* ([MT07]) prevents truncation attack and permits verification without online trusted server but doubles the use of forward-secure MACs.

11.4.2 Asymmetric-key based

The mathematical properties of a*symmetric aggregated signatures* can be exploited to replace the use of keyed hash chains for integrity verification. Instead of using MAC tags containing values from predecessors, signatures of each entry are sequentially combined into a single *aggregate signature*. The verification of such aggregate signature implies the integrity of the whole trail. An example of asymmetric scheme for auditing is proposed in [MT09].

Logging. (Prior to logging, a Certification Authority binds *U*'s identity to its public key through a certificate). When a new event is generated, the respective signature is computed over the new data and the previous signature:

$$aggregateSignature = SIGN_{sk_{current}}(\sigma_{previous} \| Entry_{current}),$$

As for the symmetric scheme, *U*'s private signing sk is updated at every event through a one-way function. Therefore an intruder is not able to recover any previous keys and so is unable to forge signatures for previous entries. The audit trail is compound by the the list of all the log entries and a single signature-so-far, which is recomputed at every new event.

$$Audit\ trail =< [E_{start}, ..., E_{last}],\ aggregateSignature >$$

Verification. The verifier requests the log trail from *U* and, if not already in possession, it also retrieves the certificate holding *U*'s public key. The special verification algorithm takes as input the single aggregate signature, the list of entries to verify and *U*'s public key (extracted from *U*'s certificate). The successful verification of one aggregate signature is equivalent to that of each previous one, thereby asserting the integrity of the whole audit log.

Evaluation Advantages:

- Delayed detection attack-free. Each signature entry can be immediately verified using the corresponding signature so far and the public key, so there is no time gap between event generation and verification.
- Allows public verifiability without online trusted server. Anyone who can obtain a copy of the audit trail can verify it.
- PKC properties provide non-repudiation. Since only a specific entry know the private signing key, that specific entity must be the one who signed the event, and no one else.

Disadvantages:

- Computationally expensive for loggers and verifiers.
- Necessity for Certificate Authority to create *U*'s certificate and a Public Key Infrastructure for later key management.
- In its basic version, it suffers from truncation attack.

11.5 Attacks

When audit logs are employed as security components to record events in a system, they the first target after an attacker intrusion since the attacker will try to delete his traces in the most recent log events. But not only malicious intrusions can endanger the audit. Also system malfunctions or human errors can alter the logs.

11.5.1 Threat Model

The purpose of the following section is to give an high level understanding of which are the security threats to consider in the design of a secure auditing system and not to give mathematical proof of the primitives (see *Section 11.3*). With attack, is defined *"any attempt to destroy, expose, alter, disable, steal or gain unauthorized access to or make unauthorized use of an asset"* [WAT]. The threat model is this representation of the set of possible attacks. The most common threat model is the "honest but curious" threat model. That is, the logging service correctly provides the service as expected, but it may try to breach confidentiality and get information from the log entries.

An attacker situated between the devices and the logger might intercept, eavesdrop, replicate, eavesdrop, synthesize, insert, delay, schedule, modify, block, discard, messages. These actions are specified by the *attacker model*, a representation used by security scientists to prove the validity of the security systems[11]. For the storage phase, the penetrator is already inside the machine holding the audit trail and has control over the system. So he can read, (over)write, re(move), delete (fields of the) log entries.

[11]These concept and the mathematical proof have been studied in the paper with the *Dolev-Yao* attacker model, where a computationally bounded enemy is in control of the communication channel and can be a legitimate participant or try to impersonate a legitimate one.

There are no security measures that can protect the log audit after the attacker succeeds in compromising the system. In fact, if the attacker gains control over the machine holding the audit trail, no one can prevent her from alerting/deleting post attack log entries.

11.5.2 Assumptions

- The primitives involved in the audit (*Section 11.3*) always hold the expected properties. For example, it is infeasible to intentionally cause collision of hash values or calculate the pre-image of hash function (strong one-way hash property holds); the decryption of messages requires the appropriate cryptographic key (the attacker can decrypt messages and get the content of an encrypted message/log entry only if he possess the corresponding key).
- The hardware and the hosts' operating systems are not corrupted at the beginning of the protocols.
- The pre-shared key are not compromised at the beginning and the are shared off-the-band (email, sms, other ways) or set up through secure methods. As we have already seen, if an attacker gains access to these secrets, no security properties can be guaranteed.
- An attacker is anyone who behaves maliciously and does *not* know the secret key.
- If an attacker compromises a machine, he obtains the secret key at the time of compromise.

11.5.3 Types of Attack

The attacker's goal is to tamper with the log file by removing, altering, appending or reordering log entries. Most commons attacks against auditing systems are:

Key Stealing and Log Alteration. With symmetric schemes, verifiers must posses the secret used as the hash chain seed, in order to rebuild the hash chain during the verification process. If the attacker can get access to this key, for example, penetrating the verifier's machine, this will give him the ability to falsify the log entries *before* the compromise time.

Truncation Attack. Removing or altering data within the hash chain – where every record is linked to the previous one – is detectable by verifiers, since hash chains provide tamper evidence. On the other hand, removing a continuous set of entries from the end of the audit trail (tail-truncation) will go undetected, since no fake MACs are recomputed and consequently no links in the hash chain are broken.

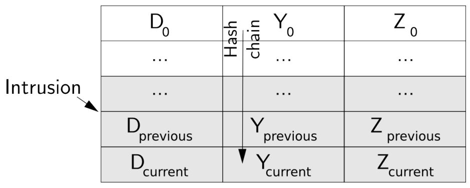

Intrusion
Hash chain
D_0 Y_0 Z_0
$D_{previous}$ $Y_{previous}$ $Z_{previous}$
$D_{current}$ $Y_{current}$ $Z_{current}$

Figure 11.8: Truncation attack. Log entries with darker background are removed.

This attack can be avoided in two ways. A first solution is to make the verifier aware of the length of the hash chain, so it will notice the lack of entries. The second manner is to integrate a tag which protects the integrity of the entire audit trail.

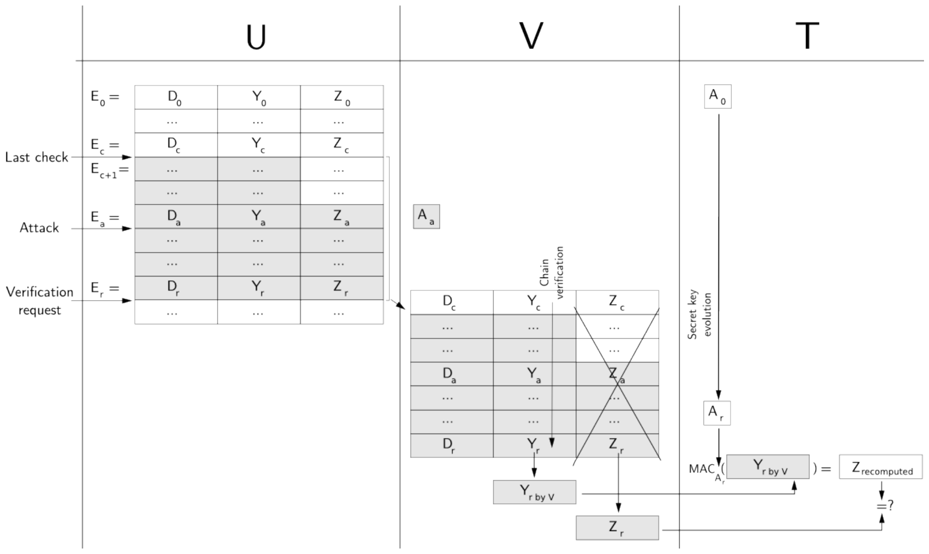


Figure 11.9: Delayed detection attack.

Delayed Detection Attack. This attack is a consequence of the three party verification protocol and therefore will be eventually detected the first time two party verification is executed. The time span in which this attack goes undetected depends on the frequency of the interactions between *U* and *T*.

The attack is illustrated in *Figure 11.9*. At time *a* the attacker breaks into *U* and obtain the current key A_a which has not been evolved yet. The attacker modifies *D* values and *Y* values (not Z values) of pre-compromise entries from E_{c+1} to E_{a-1}. The attacker can modify *D*,*Y* and *Z* values from the time *a* to *r* (request time), since now in possession of the secret key A_a.

After time *r*, *V* starts the three party verification and obtains the entries from E_c to E_r. It recomputes the hash chain values from Y_c to Y_r and it sends to *T* the last result $< Y_{lastByV}, Z_r >$. *T* recomputes $Z_{recomputed} = MAC_{A_{current}}(Y_{lastByV})$ and the identity $(Z_{recomputed} = Z_r)$ will be verified.

Since the *V* was not able to verify the *Z* values from E_a to E_r and *T* only verifies the last entry $Z_{recomputed} == Z_r$, no broken links are detected. On the other hand, the first time *T* will proceed with the autonomous verification of the whole chain, it will encounter a wrong correspondence between the altered Y_{c+1} and the Z_{c+1} which the attacker could not recompute without the knowledge of A_{a-1}.

Chapter 12

State of the Art

The roots of digital auditing can be tracked down to the first logging solution, *Syslog*. Developed in the 1980s as auxiliary component for the early mail protocol, *Syslog* rapidly became standard de-facto as multi-purpose network monitoring tool, because capable of supporting a great variety of operating systems and network devices (routers, printers, servers, ...). In its simple protocol, each message specifies the network facility that generates the event and a severity level (e.g.: *<auth, alert>*, for an alert alarm regarding the authentication system). Relying on UDP Transmission protocol (no reliable delivery) and without any secure protection for transmission between the end-points, it was a very bare-bone logging system, not suitable for secure auditing (more recent solutions have been developed since then to add confidentiality and reliable delivery, [RBS+12]).

The pioneer auditing solution has been presented by *Bellare et Yee* in their foundational paper [BY97]. Their goal was to create a new kind of MAC that was unforgeable even if an attacker could get the keys, protecting the integrity of the entries pertaining to the past: the *Forward Integrity* property. Audit records were tagged and secure together by means of these special MAC tags computed with evolving keys.

Schneider et Kelsey took the work from *Bellare and Yee* and proposed a new scheme *SK* (detailed in 11.3.2) for distributed systems [SK99]. Like *BY* solution, *SK* is based on forward secure MACs, but the rekeying is done after each single entry.

Now each entry is linked to the previous one in a *hash chain*. In addition, they provided confidentiality and authorization to verification process, employing symmetric encryption and a permission mask for selective disclosure of the entries. *SK* scheme requires an online trusted server for the verification process. In other words, T must be available whenever V needs to verify the integrity of the audit trail. Being a centralized solution, SK suffers fro architectural design risks, primarily Single Point Of Failure. Both previous solutions suffer from *delayed detection attack*.

Holt removed the necessity for the online server in the verification process with a public key approach. In *LogCrypto* [Hol06] the concepts shown in the *BY* and *SK* schemes are extended to the PKC replacing MACs with digital signatures. Authentic signatures correspond to the verification of hashed MACs in the previous schemes. In addition, the use of digital signature also provides *public verifiability*: now entries are signed and stored with one key and verified with public one.

Unfortunately, the three previous solutions suffer from the *truncation attack*. Also, the verification of one particular record in the log forces the verifier to check the whole chain, which is computationally linear with the entry number.

Ma et Tsudik solved the problem of truncation attack while at the same time improving the performances of the public key solutions. In their solution *FssAgg* [MT09], they proposed a new scheme for *signature aggregation* that avoids the storage/-transmission of the authentication tag for each single entry. Individual signatures generated by the same signer are sequentially combined into a single aggregate one, and all the component signatures are deleted. The successful verification of the aggregate signature is equivalent to that of each component signature, thereby implying no tampering on the audit trail. An attacker that breaks into the system can not craft this signature without knowing the previous (deleted) component signatures. Nevertheless, it remains very expensive to verifying a single entry, because the chain need to be checked from the beginning.

A different solution is given by *Accorsi* in his work [Acc11] which is focused on improving the performances for public verification trough the use of Trusted Computing Platform components. It recurs to the utilization of a hash-table containing the pre-images for all the computed entries. While this memory-time trade-off lowers the computational demand, it introduces a security vulnerability: the security of this table.

A first work by *Yavuz et al.* [YN09], called *BAF* (Blind-Aggregate-Forward) improved the performances of aggregate signatures for the signing process. A successive work [MT09] lowered the computational demand for the verifying process. *LogFAS* achieves faster verification storing all the signatures in their original form, without aggregating them. Every time a verifier needs to check the integrity of the log trail, it reconstructs the aggregate signature.

Among other complementary works: *Chong*'s application of the *SK* scheme to realize tamper resistant hardware [CPH03]; *Water*'s work in auditing with encrypted searches [WBDS04]; secure data structures, such as the authenticated hash tables in [PTT08] and the tamper-evident data structure shown in [CW09]. More recently, a solution proposed in [RBS$^+$12] leverages Tor as anonymization infrastructure to transmit and retrieve log entries in an privacy-aware manner.

CHAPTER 13

sensible-auditor

Here are exposed the design, the implementation and the evaluation of *sensible-data*'s auditing system.

Based on the properties acknowledged in the previous chapters, *"sensible-auditor"* has been designed as a **symmetric tamper evident** logging application. Its role is to record – in a secure manner – events happening whiting the *sensible-data* system. It inherits from the *Schneider-Kelsey* scheme, which guarantees *forward integrity* of the audit trail and allows such property to be verified by the users of the system (e.g.: study owners, sysadmins, participants).

sensible-auditor has been implemented as a Django application integrated in *sensible-data* web service. Its design permits the audit trail to be hosted locally on the same machine with *sensible-data* (as a MongoDB instance) or onto a remote database. Data transport among the parts is secured over HTTPS.

13.1 Design

This section illustrates the design of the **symmetric tamper-evident** solution for the auditing of *sensible-data* service. Its architecture relies on the principles shown in the *Schneider-Kelsey* scheme [SK99]. Logging and verification are computationally cheap respect the asymmetric alternatives, a fundamental factor for a system such as *sensible-data* which needs to bear numerous simultaneous records to append/verify. In addition, the specific configuration of the *sensible-data* platform made possible to adapt *SK* scheme in a different – simpler but secure – fashion. The distinguishing peculiarities have been outlined in the following paragraphs, while evaluations of the architectural design and security properties are given in *Section 13.3*.

13.1.1 Principals.

In sensible-auditor, with the term *principal* is intended a representation of an entity that needs authentication. Those entities constitute a subset of the roles established by the sensible-data system[1]. For the specific case of a CSS study, the categories for these entities are: administrator (who is in charge of the management of the system, study owner (who creates and manages the study), and study participant (who provides the data).

To each principal are assigned rights and privileges over the services offered by *sensible-data*. *sensible-data* system identifies principals, check their rights and grant authorizations, therefore providing *sensible-auditor* with the concept of (un)authorized entities.

13.1.2 Log Events.

Whenever such principals affect *sensible-data* resources (data, services, and processes), a representation the event is stored by the logger. Each record contains information about the interaction with the system and specifies the principals involved through their associated identifiers, written in the record field *D*, as shown in *Figure 13.1*.

Some examples of events are study creation, user secret key update, participant add/removal to/from the enrolment list, app add/removal to/from whitelists. In addition, the logger records data flows in and out of the system (from the application towards the system or viceversa, e.g.: alert messages through the smartphone app); app authorizations, audit trail integrity check.

[1]To not be confused with the term *role* intended in the auditing literature.

13.1.3 Protocol

Secure Log Entry Format. As specified in *Section 11.1*, each log record may contain fields employed for security reasons. In the *sensible-auditor* design, these fields are necessary to provide the audit trail with the *forward integrity* property. This security property is achieved through the use of "MAC hash chains" (the use of hash chains is illustrated in *Paragraph 11.3.3*). Each entry in the log contains a value from the previous record creating a chain of interdependent links. If data get modified, the chain breaks (re-computed fields will have different value from the stored ones) making alterations evident.

sensible-auditor's design is:

- **Symmetric** : the same evolving key is employed to compute a MAC tag and to verify it;
- **Tamper evident** : for the properties explained in *Section 11.3* and demonstrated by the works of *Bellare*, *Yee*, *Schneier*, and *Kelsey* [BY97, SK99].

The structure of the log entries is as follows:

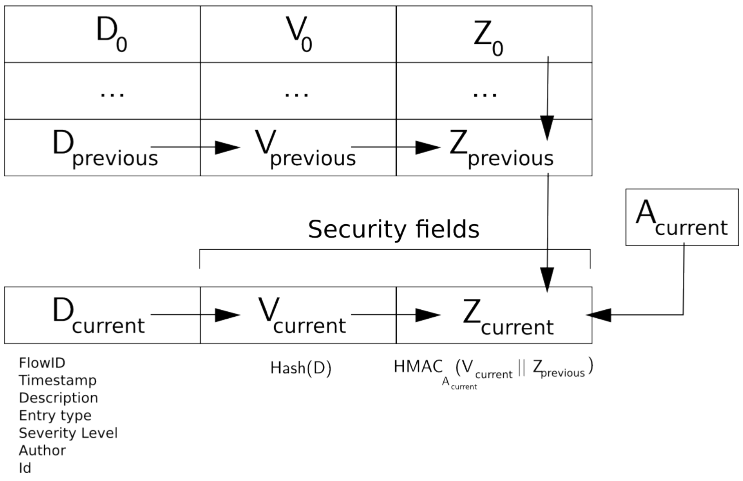


Figure 13.1: *sensible-auditor* log entry.

- **D** : contains all the fields as specified in *Section 11.1* for later analysis. Events' payload are stored as digest values (smaller and of fixed length). The payloads are stored in *sensible-data*'s long-term storage database of for later analysis.

D is stored in clear-text, but the users' information identifying a specific Personally Identifiable Information user are anonymized by the *sensible-data* anonymizer. *D* contains also a **FlowID**, the unique identifier of each event in *sensible-data* system for retrieval in log examination. The integrity is protected by the checksum value store in *V*.

- **V** : is the checksum of *D*. Any alteration of the data saved in the *D* value, is later detected in the process of chain verification by this value. Being the output of a hashing function, *V*'s value is not distinguishable from a random string.
- **Z** : is a HMAC tag that constitutes the hash chain link. It integrates value regarding the current data to log (*V*) and a value (*Z*) from the previous entry to form the hash chain. Its value depends on the evolving key *A*. The correct value of a MAC tag on a hash chain link is equal to a MAC of all the previous entries, therefore assessing the integrity of the whole chain (the verification process is shown later in *Paragraph 13.1.3*).

$$Z = HMAC_{A_{current}}(V_{current} \| Z_{previous})$$

Entities. *Figure 13.2* shows the entities involved in the auditing process. These are a subset the software components of *sensible-data* platform (*Section 9.2*) paired with their corresponding auditing roles and trust levels (*Section 11.1*):

- **Third party application** : is the *trusted event generator* which generates authenticated streams of data through smartphone apps or web applications.
- **Connector**: *sensible-data*'s connector is the *trusted event receiver* which forwards the events to both the long-term storage and the *audit module*.
- **Database** : is the *untrusted* long-term storage for the data used in the studies. If an intruder penetrates the database and changes the entries, the checksums contained in the secure audit's records will show mismatches.
- **Audit module**: *semi-trusted*, filters the event and decides which of them need to be stored in the trail.
- **sensible-data server** : it incorporates the *connector* and the audit modules. It also helps the user in the verification process as a proxy between him and the remote log. There are differences with *SK* scheme. First, in sensible-auditor scheme this server needs to be always running for logging and verifying. Affecting server's availability compromises the whole system function. Therefore, it has been assumed that there is a constantly available high-bandwidth channel between the logger *U* and the user *T*. Consequently,

user and *sensible-data* server can communicate frequently. Second, the server needs to acquire the lowest level of trust among the component it is hosting (*untrusted*). It provides the service as needed but it is *not* trusted to store the seed key.

- **Audit trail** : is the physical storage for the audit trail. The machine hosting the tamper-evident audit trail can be considered *untrusted or semi-trusted*, in fact audit trail's forward integrity property can tolerate an intrusion by an attacker.

- **Key Store** : it holds the key for the next entry. $< username, nextKey >$. Even if compromised, it does not endanger log's forward integrity property (*Semi-trusted*).

- **User** : the *trusted* verifier, as opposed to *SK* scheme where the verifier is *semi-trusted*.

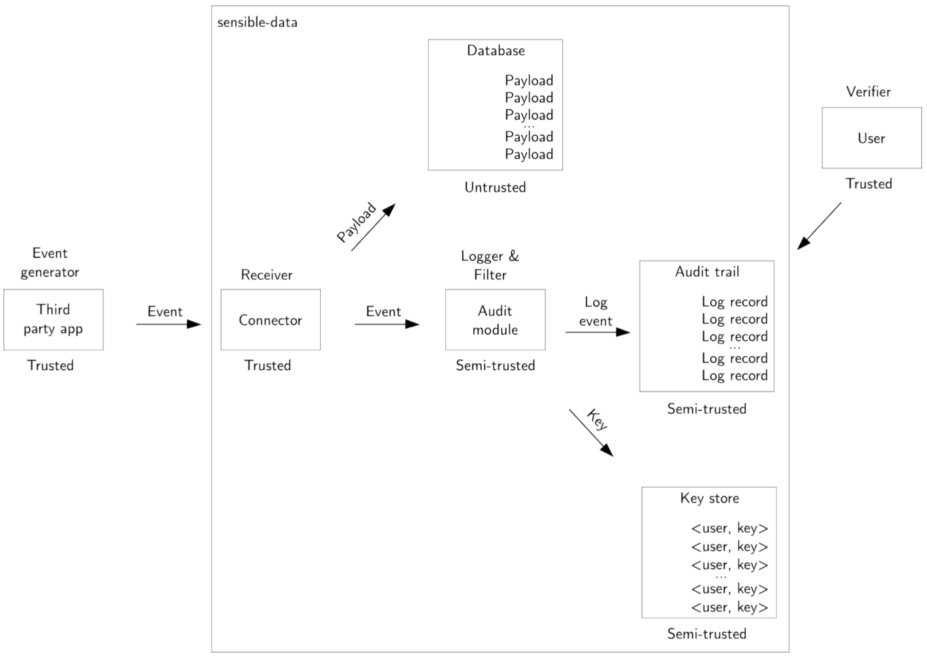


Figure 13.2: *sensible-auditor* roles.

Protocol steps. The protocol outlined below follows the concepts delineated in *SK* scheme 11.4.1:

A) Logging. When a user signs up to sensible-data system, he creates a passphrase from which the hash chain key is derived. It has to be noted that the secret key is *never* stored within sensible-data server.

1. **Event generation.** A new data event is generated (e.g.: a study questionnaire is filled out by a participant using an app on his smartphones).

2. **Event collection.** Data is collected on *sensible-data* server (push or pull policy) and checked for authenticity by *sensible-data*'s authentication module. *sensible-data* calls the logger method to append the event data.

3. **Creation of D value**. The logger extracts the payload and computes its checksum. All the information required by the log entry format (see 11.1) are now filled and stored into the field $D_{current}$.

4. **Creation of V value**. Once *D* has been created, the logger generates the corresponding digest and saves the $V_{current}$.

5. **Creation of Z value**. *(5a)* The logger retrieves the current value of the evolving key $A_{current}$ from the key-store. *(5b)* It also extracts from the audit log the *Z* value of the previous entry ($Z_{previous}$). *(5c)* Then, it computes the forward secure HMAC for hash chain link: $Z_{current}$ is computed over the current data $V_{current}$ and $Z_{previous}$ with the current value of the evolving key, resulting in:

$$Z_{current} = MAC_{A_{current}}(V_{current} \| Z_{previous})$$

6. **Key overwrite.** As soon as the *Z* value is created, the key $A_{current}$ is immediately evolved and overwritten in the key-store by means of a one-way hash function to guarantee the forward integrity property:

$$A_{current} = Hash(A_{current})$$

7. **Record append.** The logger proceeds in writing the security fields with the computed data and appended the resulting entry at the end of the log trail:

$$LogEntry = \quad < D,\ V,\ Z >$$

8. **Data storage.** Finally, the data payload is stored in the database for later purposes (e.g.: CSS study analysis).

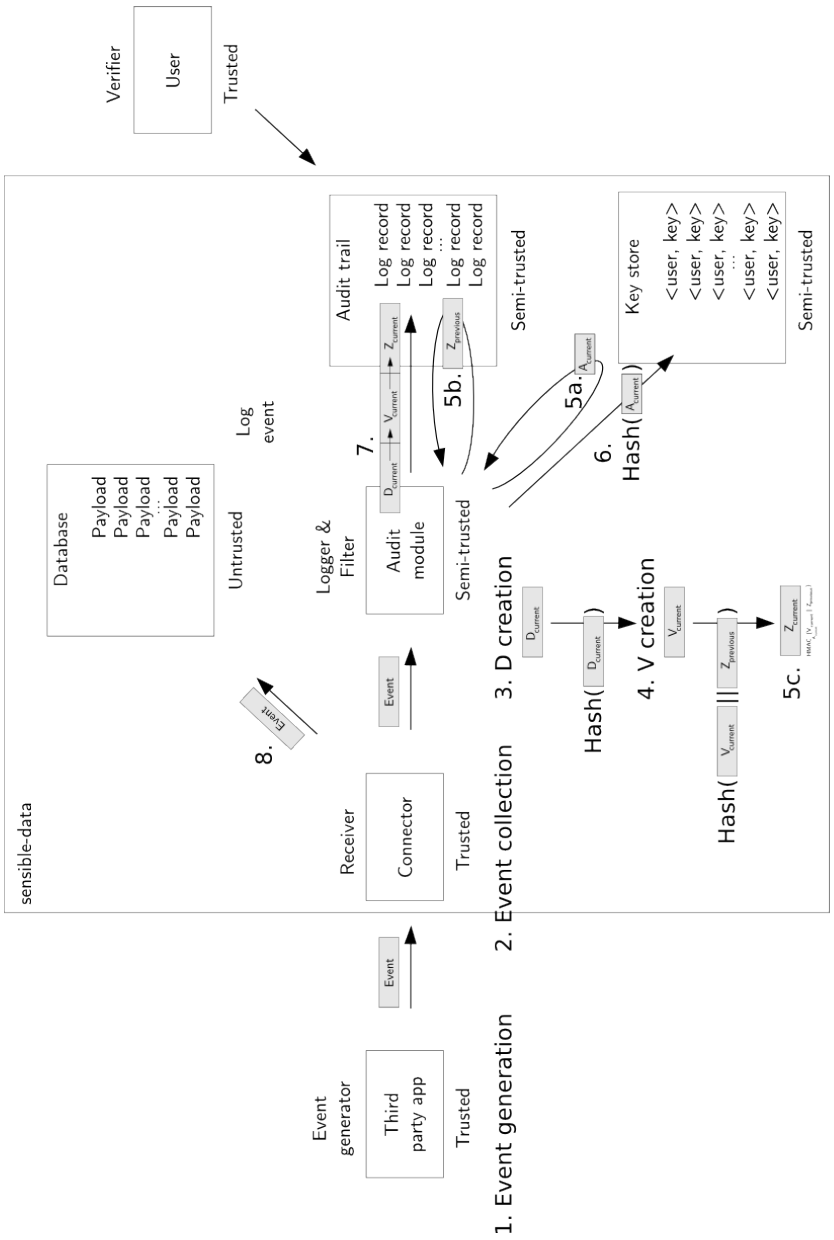


Figure 13.3: *sensible-auditor* logging protocol.

B) Verification. As stated, any entity in possession of the secret key A is allowed to verify the integrity of the audit trail. In *sensible-auditor*'s scheme, this entity is the user, the owner of the data created by the applications. Since the verifier is a trusted entity, *sensible-auditor*'s protocol does *not* require a two party verification as seen in the *SK* scheme.

1. **Integrity verification request**. A user requires sensible-auditor to check the integrity of the entries in the range $flowID = firstID$ to $flowID = lastID$, with $lastID >= firstID$. He also communicates the secret key to *sensible-auditor* over a secure channel.

2. **Records retrieval**. *sensible-auditor* retrieves all the entries that must be checked from the log plus the entry that precedes the first one:

$$[logEntry_{(firstID-1)} \ldots logEntry_{(lastID)}]$$

3. **Secret key evolution**. The secret key is evolved (one-way hashed) as many time as required to be aligned with that very first entry.

4. **Verification algorithm**. The following pseudo-code traverses the hash chain (*Z* values) verifying that each link is correct.
   After retrieving the necessary records (*current* and *previous*), it recomputes the checksum V_{test} using the data from the current entry $D_{current}$.
   If $(V_{test} == V_{current})$, the algorithm goes on, otherwise it means that or $D_{current}$ or $V_{current}$ (or both) have been modified. Since the record integrity has been nullified, the algorithm exists.
   Then, it extracts the *Z* values from the current entry $Z_{current}$ and from the previous entry in order to recompute $Z_{test} = MAC_{A_{current}}(V_{test} \| Z_{previous})$. This time the logger checks whether $(Z_{test} == Z_{current})$. If these two value do not correspond, the chain has been broken: or $Z_{current}$ or $Z_{previous}$ have been altered[2]
   If the chain link is not broken, the algorithm evolves *A* and increments the *flowID* to check the next entry.

[2]It is taken for granted that *A* has not been compromised since originally provided by the trusted user.

```
begin
  while $(flowID <= lastID)$ do
        $current$ = getCurrentEntry($flowID$)
        $previous$ = getPreviousEntry($flowID - 1$)
        $V_{test}$ = checksum($D_{current}$)
        if $(V_{test} != V_{current})$
          then exit fi
        $Z_{test}$ = $\text{MAC}_{A_{current}}(V_{test} \| Z_{previous})$
        if $(Z_{test} != Z_{current})$
          then exit fi
        $A_{current}$ = hash($A$)
        $flowID = flowID + 1$
  od
end
```

Finally, if at the exit of the loop $(flowID == (lastID + 1))$, the chain has been fully traversed without incurring in any broken links, therefore the audit trail has not been tampered, implying its integrity.

13.2 Implementation

This section illustrates *how* the design specifications previously outlined have been technically realized.

13.2.1 Sensible-auditor modules

Modules. sensible-auditor is a Django application written in Python programming language for the *sensible-data* web service. The code – presented in Appendix B – has not been optimized but written to be self-explanatory and easy to maintain[3].

The **helperModule.py** contains all the functions used by both logging and verification processes. These are instructions to handle data structures (*"convert"*, *"extract"*) and the methods for computing the security field for the log entries (*"createV"*, *"createD"*, *"createZ"*). These latter methods implement *steps 4, 5,*

[3] *"Premature optimization is the root of all evil"*, Donald Knuth, the "father" of the analysis of algorithms.

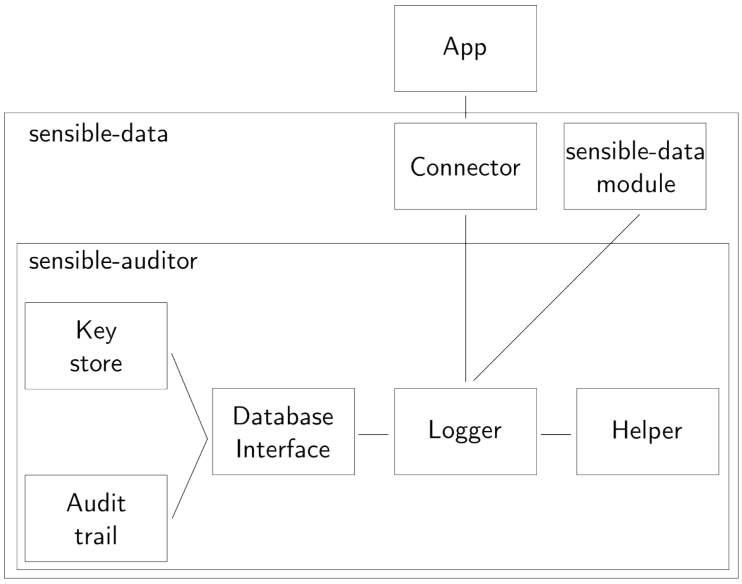


Figure 13.4: A simplified class diagram for *sensible-auditor*.

6 shown in the logging protocol. The python cryptography library PyCrypto[4] has been used To calculate hashes and HMACs (*"calculateHash"*).

The **loggerModule.py** is the main class of *sensible-audito*r app. It contains the setup method *"createUser"* which is invoked when a new participant join the service. It creates e new personal log trail and updates the relative secret key, created by the user himself at registration moment[5].

The *"append"* method is the callable function used by other modules in *sensible-data* whenever an event needs to be stored. This function takes different inputs according to which event needs to be stored (e.g.: participant's data coming into the *sensible-data* server, system owner update of the study detail, etc.). Whenever it gets called, it first checks whther the calling principal has the right authorizations, then using the helperModule's method, it creates a new log entry and appends the new entry to the log trail. Finally, it evolves the secret key and overwrite the old one with the new value in the key-store.

For the trail integrity verification, there are two main methods. First, *"getDataForCheck"* retrieves data that need to be checked and caches them into memory to speed-up

[4]`https://www.dlitz.net/software/pycrypto/`

[5]In the current implementation, it also holds the connections with the remote audit log and with the remote key-store.

the verification process. Then, the *"startCheck"*, recomputes the hash chain as previously outlined in the protocol steps.

"logAPI.py" module groups all the functions related to the specific back-end where the audit trail is hosted. Whenever a different back-end is employed, methods in this module needs to be overridden[6].

13.2.2 Storage

Audit trail and **key-store** are both implemented as MongoDB instances. MongoDB[7] is a NoSQL database system that stores data in JSON-like documents: documents are like SQL records and collections correspond to SQL tables. Its flexible schema is very well suited for archiving and event logging. In addition, it can be queried through REST and HTTP interfaces. For the auditing, the correspondence between MongoDB and *sensible-auditor* is simple: to each user has been assigned one collection (audit trail) which contains all the relative documents (events). The key-store is a unique collection containing document/pairs $< user, evolvingKey >$. As established at design phase, the audit trail is sequential, therefore it is assumed that MongoDB instances are sufficiently large to hold all the documents (events).

13.3 Evaluation

13.3.1 Design Evaluation

Forward integrity and tamper evidence rely on the hash chain properties demonstrated by Bellare, Yee, Schneider and Kelsey in their works [BY97, SK99].

SK scheme is prone to truncation attacks, delayed detection attacks and it also requires an online trusted server during verification. Nevertheless, the particular scenario of *sensible-data* permitted to create a secure design for its auditing system.

Truncation attack. As shown in *Section 11.5*, a truncation attack consists of deleting a continuous set of entries from the tail of the audit log. This attack is possible if there is no *"reliable, high-bandwidth channel constantly available between*

[6]Currently it implements the connection to the remote log hosted on MongoLab.
[7]`http://www.mongodb.org/`

T and U" [SK99]. In other words it the trusted verifier *T* is *not* aware of the total length of the trail on the logger *U*. For *sensible-auditor* the trusted verifier is the user, and the untrusted logger is the machine hosting the audit trail. Each user of *sensible-data* service is aware of what is happening to his personal data by the use of the dashboard, a live monitor of his data streams. This guarantees that the user is always aware of the last entry that must be appended to the trail. Any deletion will be detected as soon as the integrity of the chain is verified. The user is able to check the integrity of his chain at any time, since the verification is immediate and only needs his secret key.

There is also another possible solution to truncation attacks. This method employs special entries that surrounds block of records as shown in [RBS$^+$12]. At the beginning of each block, a random number *n* is generated and hashed with the rest of the security fields. For each subsequent entry in the block, this number is decreased until it reaches the value of zero. At this point, a special closing entry is appended to seal the block of entries. This procedure is repeated during the entire data stream. If an intruder tries to cut entries from the tail, he will not be able to forge an authentic hash since he does not know the value of *n* at that moment. In addition, the lack of these special entries (open/close) will be detected during the verification process.

Delayed detection attack. In *SK* scheme, delayed detection attack can be perpetrated since the semi-trusted entity *V* does not possess the secret key A_0 to verify the MAC tags. In *sensible-auditor*, the trusted verifier possesses the secret key and can immediately very each single entry in the log, therefore defeating this attack.

Online trusted server. The last difference with *SK* is the requirement for the online trusted server *T* for the verification process. Once again, in *sensible-auditor* the user is the trusted entity holding the key, so it redundant that he must be present during the verification process: without him, there is no key, so there is no verification.

Other peculiar security characteristics of *sensible-auditor* design are:

Key security. The security of this particular design relies on the secret key. Therefore it is necessary to mitigate potential key compromise. First, there is only one entity which holds the key, the user *T*. Secondly, *sensible-auditor* employs an "evolve and delete" key update strategy to preserve the forward integrity property. During the logging process the key is retrieved, used for computing the secure MAC and immediately one-way hashed (irretrievably deleting the previous value in the keystore). Therefore, even if an intruder breaks into the machine holding the keystore, he will not be able to retrieve the past keys and modify pre-compromise entries.

Users' personal audit trail. In *sensible-auditor* design, instead of having one single audit trail for the whole system and for all the users, each principal has his own personal log (hash chain). This particular design has two main consequences. First, scalability, allowing the concurrent logging of events of different users without accessing a unique chain. Second, if – for whatever reason – a chain gets compromised, only that particular principal, and therefore that particular chain, looses its integrity, avoiding a domino effect to all the others. This principle is reflected in the implementation, since each og trail has been implemented as a different collection in MongoDB.

Auditing Properties. Finally, we conclude with an analysis of the auditing main properties shown in *Section 11.2*:

- **Integrity**. Auditing log's main purpose is to be a truthful representation of the events happened in a system. *sensible-auditor* achieves this goal storing sensible-data's events in a secure form. Relying on *SK*'s design, it employs MAC tags to build hash chains that create dependencies between each record. Any modification of the hash chain makes tampering detectable to verifiers (*tamper evidence*). This design crystallises the past state of the trail at every new entry, making it impossible for an intruder to modify what already stored in the log (pre-compromise). Such property is called *forward-integrity* and protects the integrity of log entries accumulated *before* the attack, from insertion, alteration, deletion and re-order and can be verified through a chain verification process which detects broken links.

- **Verifiability**. *sensible-auditor* provides restricted verifiability, not public verifiability. Each user possess a secret that permits log integrity verification. Since this secret makes the alteration in the chain detectable, its knowledge must be restricted to trusted entities only.

- **Accountability**. The D field in each entry specifies the author of the appended event.

- **Searchability**. Each log entry contains a field (D) that permits browsability. Data contained in such field are previously anonymised by *sensible-data*'s anonymizer component.

- **Confidentiality**. Even if not the primary purpose of auditing system, confidentiality is guaranteed on the limits of what provided by *sensible-data* system. *sensible-data* system encrypts and/or hashes information before storing them in the audit log making them anonymized.
- **Append-only**. The forward integrity property guarantees that no record can be inserted before the last-in-the-chain event.

13.3.2 Implementation evaluation - Architectural Analysis

sensible-data is a system meant to support the concurrent access of numerous users. The performances of the auditing component need to be adequate to the data throughput that *sensible-data* service might generate, otherwise it will turn out to be an unreliable representation of the events. Often, in the implementation process of adapting an abstracted design to the concrete reality, constraints are met in *using*, *transmitting* and *storing* the data. The following problematics have been considered due to the distributed nature of sensible-auditor.

Assumptions

For both data in use and data at rest, two main assumptions have been made:

- **Memory erasure**. Audit log's forward integrity property relies on the secrecy of the evolving key. It is assumed that the operative systems hosting *sensible-auditor*, *audit trail* and the *key store* completely remove from the memory any traces of old data, such as the keys used during logging and verification processes.
- **Memory isolation**. For the same reason as above, it is assumed that the operative systems are strengthened against covert, side, and time-channel attacks.

Data in use

Excessive **computational cost** might slow down logging and verification processes, making the auditing system unable to record or verify. The symmetric architecture preferred at design phase allows to keep the computationally requirements low (no cumbersome PKC signing procedures).

The decisions about which algorithms should have been employed for the hash chain generation and the key evolution are implementation details. The current implementation of *sensible-auditor* uses SHA-512 hashing function for evolving the secret key A and for computing Y and Z values. SHA-512 is a secure hashing algorithm designed by the NSA and published as NIST standard [FIP]. It is a general purpose hash functions, designed to compute digests of big amounts of data in as short a time as possible. Being fast makes it very suitable for ensuring the integrity of hash chains in the log audit [SHA]. In addition, data from other *sensible-data*'s modules come already hashed or encrypted, therefore those expensive operations are not necessary.

Data in transit

sensible-auditor's audit trail is located onto the same machine hosting *sensible-data* service. In scenarios where *remote* hosting is necessary[8], data throughput might become a problem. In these cases, the minimization of the **communication overhead** is fundamental to avoid bottlenecks situations especially with bursts of data at about the same timings. For example, it is reasonable to assume that a lot of Bluetooth dyads can be uploaded as soon as students meet each other at the university. There are three events in which sensible-auditor would interact with the remote machine: storing new entries, verification and key retrieval. A possible manner to minimize connections to the remote machine would be to implement ad-hoc caching mechanisms. The verification process, for example, could benefit from the retrieving of (parts of the) whole block of entries to check, instead of establishing different singular requests. All data transmissions among *sensible-data* components are carried over TCP protocol which implements reliable delivery and manages data retransmission.

Data at rest

As shown in *Section 13.1.3*, *sensible-auditor* provides forward security to the audit trail by means of security fields in each log entry. Two concepts must be considered to assess the negative impact of such **storage overhead**. First, the proportion of the overhead must be related to the size of what to store. Events in the log trail have been represented in the efficient ways to keep the storage to minimum size. The security fields Y and Z are short-size digests that ensure data integrity. As explained above, Y and Z are the outputs of SHA-512 hashing algorithm, therefore each of them requires 512 bits. On MongoDB these values have been stored as

[8]Like during the development phase.

hexadecimal representation with the result of 16 bytes each (each digit codes holds 4 bits, 512/4 = 128 bits).

The second factor to consider is the storage capacity of the system. During the development phase of *sensible-auditor*, it has been adopted a third party *Database-as-a-Service* that provides a large storage at a low pricing. For such scenario, it is reasonable to store enough information to make retrieval an analysis fast and easy. Efficiency is an important concern for practical audit log systems.

It is worth to mention that confidentiality of the stored data and availability of the auditing service (logging and verification) depend on the security of the MongoDB instances.

13.4 Future Work

Section 13.3 has shown room for improvement in both architectural and security design of *sensible-auditor* module.

Security Requirements

- Adding automatic **routines to verifying log's integrity** check could be a valid security enhancement. As shown in [SK99, MT09], this could be easily achieved. First, another security field (MAC tag) needs to be added in each log entry; then to a new *semi-trusted* entity will be integrated the knowledge of the key for the verification of this new security field.
- Some deployment of *sensible-data* service could benefit from p**ublicly verifiability** (e.g.: electronic vote). In such scenario, an asymmetric solution could replace the actual symmetric design. A very simplified protocol has been explained in *Section 11.3.4* and some fitting examples have been mentioned in *Section 12*. Further studies are needed to assess a) security implications and b) the feasibility of the necessary Public Key Infrastructure.

Architectural improvements

- First of all, it has be mentioned that *sensible-auditor* has been tested only in closed development environments. Benchmarks about performance, scalability, and robustness must be run in a live "production" environment before discussing any other improvements.

- If those tests successfully validate the capabilities of *sensible-auditor*, the module can be developed as a remote auditing web service accessible through REST APIs. In this way, its auditing capabilities would be usable by more applications.

- It is worth to explore more efficient-data structures for storing/retrieving records and also to investigate the possibility of encrypted queries to improve confidentiality when the content of an audit log contains sensitive information (see [WBDS04] on searchable encrypted audit logs).

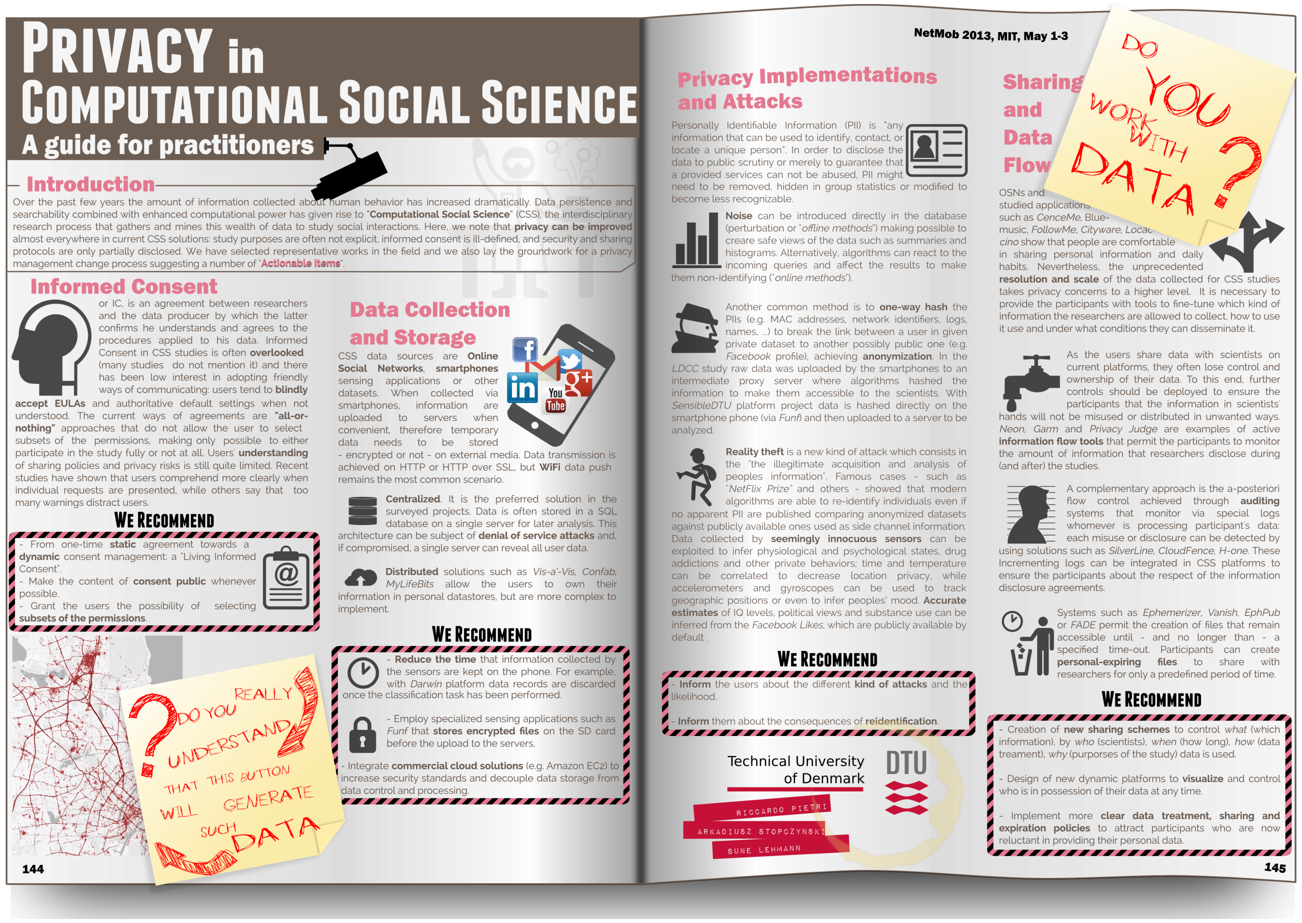
Appendix A

PiCSS Poster

The poster *"Privacy in Computational Social Science: A guide for practitioners"* has been exposed and presented at:

- **NetMob 2013**,
  "Third conference on the Analysis of Mobile Phone Datasets",
  MIT - Massachusetts Institute of Technology,
  May 1-3 2013,
  `http://perso.uclouvain.be/vincent.blondel/netmob/2013/NetMob2013-program.pdf`

- **NetSci 2013**,
  "International School and Conference on Network Science",
  Copenhagen, Denmark,
  June 3-7 2013,
  `http://netsci2013.net/wordpress/wp-content/uploads/2012/09/booklet_NetSci_2013_forWeb_4.pdf`



PRIVACY in COMPUTATIONAL SOCIAL SCIENCE

A guide for practitioners

NetMob 2013, MIT, May 1-3

Introduction

Over the past few years the amount of information collected about human behavior has increased dramatically. Data persistence and searchability combined with enhanced computational power has given rise to "**Computational Social Science**" (CSS), the interdisciplinary research process that gathers and mines this wealth of data to study social interactions. Here, we note that **privacy can be improved** almost everywhere in current CSS solutions: study purposes are often not explicit, informed consent is ill-defined, and security and sharing protocols are only partially disclosed. We have selected representative works in the field and we also lay the groundwork for a privacy management change process suggesting a number of "**Actionable Items**".

Informed Consent

or IC, is an agreement between researchers and the data producer by which the latter confirms he understands and agrees to the procedures applied to his data. Informed Consent in CSS studies is often **overlooked** (many studies do not mention it) and there has been low interest in adopting friendly ways of communicating: users tend to **blindly accept EULAs** and authoritative default settings when not understood. The current ways of agreements are **"all-or-nothing"** approaches that do not allow the user to select subsets of the permissions, making only possible to either participate in the study fully or not at all. Users' **understanding** of sharing policies and privacy risks is still quite limited. Recent studies have shown that users comprehend more clearly when individual requests are presented, while others say that too many warnings distract users.

WE RECOMMEND

- From one-time **static** agreement towards a **dynamic** consent management: a "Living Informed Consent".
- Make the content of **consent public** whenever possible.
- Grant the users the possibility of selecting **subsets of the permissions**.

DO YOU REALLY UNDERSTAND THAT THIS BUTTON WILL GENERATE SUCH DATA?

Data Collection and Storage

CSS data sources are **Online Social Networks**, **smartphones** sensing applications or other datasets. When collected via smartphones, information are uploaded to servers when convenient, therefore temporary data needs to be stored - encrypted or not - on external media. Data transmission is achieved on HTTP or HTTP over SSL, but **WiFi** data push remains the most common scenario.

Centralized. It is the preferred solution in the surveyed projects. Data is often stored in a SQL database on a single server for later analysis. This architecture can be subject of **denial of service attacks** and, if compromised, a single server can reveal all user data.

Distributed solutions such as *Vis-a'-Vis*, *Confab*, *MyLifeBits* allow the users to own their information in personal datastores, but are more complex to implement.

WE RECOMMEND

- **Reduce the time** that information collected by the sensors are kept on the phone. For example, with *Darwin* platform data records are discarded once the classification task has been performed.

- Employ specialized sensing applications such as *Funf* that **stores encrypted files** on the SD card before the upload to the servers.

- Integrate **commercial cloud solutions** (e.g. Amazon EC2) to increase security standards and decouple data storage from data control and processing.

144

Privacy Implementations and Attacks

Personally Identifiable Information (PII) is "any information that can be used to identify, contact, or locate a unique person". In order to disclose the data to public scrutiny or merely to guarantee that a provided services can not be abused, PII might need to be removed, hidden in group statistics or modified to become less recognizable.

Noise can be introduced directly in the database (perturbation or "*offline methods*") making possible to creare safe views of the data such as summaries and histograms. Alternatively, algorithms can react to the incoming queries and affect the results to make them non-identifying ("*online methods*").

Another common method is to **one-way hash** the PIIs (e.g. MAC addresses, network identifiers, logs, names, ...) to break the link between a user in given private dataset to another possibly public one (e.g. *Facebook* profile), achieving **anonymization**. In the *LDCC* study raw data was uploaded by the smartphones to an intermediate proxy server where algorithms hashed the information to make them accessible to the scientists. With *SensibleDTU* platform project data is hashed directly on the smartphone phone (via *Funf*) and then uploaded to a server to be analyzed.

Reality theft is a new kind of attack which consists in the "the illegitimate acquisition and analysis of peoples information". Famous cases - such as "*NetFlix Prize*" and others - showed that modern algorithms are able to re-identify individuals even if no apparent PII are published comparing anonymized datasets against publicly available ones used as side channel information. Data collected by **seemingly innocuous sensors** can be exploited to infer physiological and psychological states, drug addictions and other private behaviors; time and temperature can be correlated to decrease location privacy, while accelerometers and gyroscopes can be used to track geographic positions or even to infer peoples' mood. **Accurate estimates** of IQ levels, political views and substance use can be inferred from the *Facebook Likes*, which are publicly available by default .

WE RECOMMEND

- **Inform** the users about the different **kind of attacks** and the likelihood.

- **Inform** them about the consequences of **reidentification**.

Technical University of Denmark

DTU

RICCARDO PIETRI
ARKADIUSZ STOPCZYNSKI
SUNE LEHMANN

Sharing and Data Flow

DO YOU WORK WITH DATA?

OSNs and studied applications such as *CenceMe*, Blue-music, *FollowMe*, *Cityware*, *Locac-cino* show that people are comfortable in sharing personal information and daily habits. Nevertheless, the unprecedented **resolution and scale** of the data collected for CSS studies takes privacy concerns to a higher level. It is necessary to provide the participants with tools to fine-tune which kind of information the researchers are allowed to collect, how to use it use and under what conditions they can disseminate it.

As the users share data with scientists on current platforms, they often lose control and ownership of their data. To this end, further controls should be deployed to ensure the participants that the information in scientists' hands will not be misused or distributed in unwanted ways. *Neon*, *Garm* and *Privacy Judge* are examples of active **information flow tools** that permit the participants to monitor the amount of information that researchers disclose during (and after) the studies.

A complementary approach is the a-posteriori flow control achieved through **auditing** systems that monitor via special logs whomever is processing participant's data: each misuse or disclosure can be detected by using solutions such as *SilverLine*, *CloudFence*, *H-one*. These Incrementing logs can be integrated in CSS platforms to ensure the participants about the respect of the information disclosure agreements.

Systems such as *Ephemerizer*, *Vanish*, *EphPub* or *FADE* permit the creation of files that remain accessible until - and no longer than - a specified time-out. Participants can create **personal-expiring files** to share with researchers for only a predefined period of time.

WE RECOMMEND

- Creation of **new sharing schemes** to control *what* (which information), by *who* (scientists), *when* (how long), *how* (data treament), *why* (purporses of the study) data is used.

- Design of new dynamic platforms to **visualize** and control who is in possession of their data at any time.

- Implement more **clear data treatment, sharing and expiration policies** to attract participants who are now reluctant in providing their personal data.

145

Appendix B

sensible-auditor Module

```
from Crypto.Hash import SHA512
from Crypto.Hash import HMAC
import os
from utils import log_config as CONFIG
from Crypto import Random
from Crypto.PublicKey import RSA

import string
import random
import time
import datetime

from django.contrib.auth.models import User

def convert(input):
    if isinstance(input, dict):
        return {convert(key): convert(value) for key, value in
    input.iteritems()}
    elif isinstance(input, list):
        return [convert(element) for element in input]
    elif isinstance(input, unicode):
        return input.encode('utf-8')
    else:
        return input


def extract(myDict, myList):
    if myDict is None:
        return
    for value in myDict.values():
        if isinstance(value, dict):
            extract(value, myList)
        else:
            myList.append(value)


def create_V(D):
    outputList = []
    extract(D,outputList)
    resultList = convert(outputList)
    resultList.sort() # sort the values, done so when later it will
     be checked, the hashes will be the same
    return ''.join(resultList) # from list to string


def create_D(username, data):
    appID = data['appID']
    payload = data['payload']
    return {"userID" : username, "appID" : appID, "payload" :
    payload}

def create_Z(current_V, previous_Z, previous_A):
    hmac = HMAC.new(previous_A)
    hmac.update(current_V)
    hmac.update(previous_Z)
    return hmac.hexdigest()


def calculateHash_A(rounds, A):
    for i in range(0,rounds):
        h = SHA512.new()
        h.update(A)
        A = h.hexdigest()
    return str(A)


def getTimestamp():
    return datetime.datetime.fromtimestamp(time.time()).strftime('%
    Y-%m-%d %H:%M:%S') # Format : "2013-06-08 12:01:15"


def permissionCheck(_request, _permission):
    if _request.user.is_authenticated():
        auth = True
    else:
        print "User NOT authenticated"
        auth = False

    if _request.user.has_perm(_permission):
        perm = True
    else:
        print "he has NO permissions"
        perm = False
    return (auth and perm)
```

```


def isSuperUser(_request):
    superUser = _request.user.is_superuser
    if not superUser:
        print "Not a superuser"
        return False
    return superUser
```

code/helperModule.py

```
import pymongo
import service_config
import log_config
from utils import log_config as CONFIG
from loggerApp import helperModule

class Keystore(object):

    keytable = None

    def __init__(self):
        self.client = pymongo.MongoClient(log_config.LOGGER_DATABASE['params']['url']%(log_config.LOGGER_DATABASE['params']['username'],log_config.LOGGER_DATABASE['params']['password']))
        self.db = self.client[log_config.LOGGER_DATABASE['params']['database']]
        self.keytable = self.db[log_config.LOGGER_KEYS['params']['collection']]

    def addUserKey(self, _username, _key):
        return self.keytable.insert({"username": _username, "key" : _key})

    def getUserKey(self, _username):
        return self.keytable.find_one({"username" : _username}).get("key")

    def reset(self):
        return self.keytable.remove()

    def update_A(self, _username):
        A = self.getUserKey(_username)
        A = helperModule.calculateHash_A(1,A)
        self.keytable.update({ "username": _username }, { "$set": { "key": A } } )
        return True

    def existsUser(self, _username):
        exists = False
        if (self.keytable.find({ "username": _username}).count() != 0):
            exists = True
        return exists

    def deleteUser(self, _username):
        return self.keytable.remove({"username" : _username})
```

code/keystore.py

```
import pymongo
import service_config
import log_config
from utils import log_config as CONFIG
from loggerApp import helperModule

# SQL <==> NoSQL
# Table <==> Collection
# Column <==> Field
# Entry <==> Document

# Log entry for debugging = <flowID, D, V, Z>

class LogDatabase(object):

  client = None
  db = None
        collectionList = None

        def __init__(self):
            self.client = pymongo.MongoClient(log_config.LOGGER_DATABASE['params']['url']%(log_config.LOGGER_DATABASE['params']['username'],log_config.LOGGER_DATABASE['params']['password']))
            self.db = self.client[log_config.LOGGER_DATABASE['params']['database']]


        def createCollection(self, _username):
            collectionName = _username
            self.writeEntry(collectionName, CONFIG.FIRST_ENTRY, CONFIG.D0, CONFIG.V0, CONFIG.Z0) # Seed entry for a given user
            return None


        def writeEntry(self, _collectionName, flowID, D, V, Z):
            returned = self.db[_collectionName].insert({"flowID" : flowID, "D" : D, "V" : V, "Z" : Z})
            return str(returned)


        def getMaxFlowID(self, _collectionName):
            maxFlowID = 0
            resultEntry = self.db[_collectionName].find_one(sort=[("flowID", -1)])
            if (resultEntry is not None):
                maxFlowID = resultEntry['flowID']
            return maxFlowID

```

```

    def getPrevious(self, _collectionName, flowID):
        previous = flowID - 1
        return self.db[_collectionName].find_one({"flowID" : previous})


    def getLast_Z(self, _collectionName):
        maxFlowID = self.getMaxFlowID(_collectionName)
        return self.getEntry(_collectionName, maxFlowID).get("Z")


    def getEntry(self, _collectionName, flowID):
        return self.db[_collectionName].find_one({"flowID" : flowID})


    def getZ0(self):
        return CONFIG.Z0
```

code/log_database.py

```
import pymongo
from utils.log_database import LogDatabase
import json
from utils import log_config as CONFIG
from Crypto.Hash import SHA512
import helperModule
from utils.keystore import Keystore
from django.contrib.auth.decorators import user_passes_test
import bson.json_util as json
from django.http import HttpResponse

class Logger(object):

    logDatabase = None
    keystore = None
    flowID_to_end = 1

    def __init__(self):
        self.logDatabase = LogDatabase()
        self.keystore = Keystore()


# Get request, check permissions, create pair in keystore, start the log
    def createNewUser(self, _request, newUser, newBaseKey):
        if not helperModule.isSuperUser(_request):
            return False
        if self.keystore.existsUser(newUser): # Only if the user is not already in the db
            print "User already present in th DB"
            return False
```

```python
        self.keystore.addUserKey(newUser, newBaseKey) # Add new pair <user, key>
        self.logDatabase.createCollection(newUser) # Start the log
        return "Created"


    def append(self, _request, _data): # who is requesting to append, the data to append, who are we talking about
        if not helperModule.permissionCheck(_request, "loggerApp.auditEntry_append"):
            return False
        _username = str(_request.user)
        if not self.keystore.existsUser(_username): # Integrate this check in the previous one
            return False
        st = helperModule.getTimestamp() # add this in the log entry
        current_flowID = self.logDatabase.getMaxFlowID(_username) + 1
        current_D = helperModule.create_D(_username, _data)
        current_V = helperModule.create_V(current_D)
        previous_A = self.keystore.getUserKey(_username)
        previous_Z = self.logDatabase.getPrevious(_username, current_flowID).get("Z")
        current_Z = helperModule.create_Z(current_V, previous_Z, str(previous_A))
        self.keystore.update_A(_username) # evolve the key
        mongo_id_string = self.logDatabase.writeEntry(_username, current_flowID, current_D, current_V, current_Z)
        return (current_flowID, " ", mongo_id_string)


    def userRegistration(self, _registered, _secretKey): # _registered = True/False
        pass

    def appAuthorized(self, _authorized, _appName, _request):
        pass


    def deleteUser(self, _request):
        if not helperModule.isSuperUser(_request):
            return False
        username = _request.GET.get("username")
        return self.keystore.deleteUser(username)


    def updateUserKey(self, _request):
        pass


    def check(self, _request, _key):
        if not helperModule.permissionCheck(_request, "loggerApp.auditTrail_verify"):
            return False
        username = str(_request.user)
        if not self.keystore.existsUser(username):
            return False
        return self.startCheck_Z(username, self.logDatabase.getZ0(), self.logDatabase.getLast_Z(username), self.flowID_to_end, _key)


    def checkGroup(self, _request, _group_name):
        groups_ValuesListQuerySet = _request.user.groups.values_list('name',flat=True)
        groups_list = list(groups_ValuesListQuerySet)
        in_group = False
        if _group_name in groups_list:
            in_group = True
        return in_group


    def dataIntegrityCheck(self,current_D, current_V):
        status = False # false=KO, True=OK
        temp_V = helperModule.create_V(current_D)
        if (temp_V == current_V):
                status = True
        return {"status" : status, "temp_V" : temp_V}


    def chainIntegrityCheck_Z(self, previous_Z, current_V, current_Z, A, flowID):
        status = False # false=KO, True=OK
        previous_A = helperModule.calculateHash_A(flowID -1, A)
        temp_Z = helperModule.create_Z(current_V, previous_Z, previous_A)
        if ( temp_Z == current_Z):
                status = True
        return {"status" : status, "temp_Z" : temp_Z}


    def startCheck_Z(self, username, previous_Z, Z_last, flowID, A):
        keepLooking = True
        audit = {}
        while ( (flowID <= self.logDatabase.getMaxFlowID(username)) and (keepLooking==True) ):
            dataForCheck = self.getDataForCheck(username, flowID)
            integrityDict = self.dataIntegrityCheck(dataForCheck["current_D"], dataForCheck["current_V"])
            if ( not integrityDict["status"] ):
                keepLooking = False
            chainDict = self.chainIntegrityCheck_Z(previous_Z, dataForCheck["current_V"], dataForCheck["current_Z"], A, flowID)
            if ( not chainDict["status"] ):
                keepLooking = False
            audit = {flowID : [integrityDict["status"], chainDict["status"]]}
            previous_Z = chainDict["temp_Z"]
            flowID = flowID + 1
        return audit


    def getDataForCheck(self, username, flowID):
        if (flowID < 1):
            print "flowID = " + str(flowID) + " can not be checked"
            exit(-1)
        current = self.logDatabase.getEntry(username, flowID)
        return {"current_D" : current.get("D"), "current_V" :
    current.get("V"),  "current_Z" : current.get("Z")}



########################################################
# Call examples:

def createNewUser(request):
    logger = Logger()
    user = request.GET.get("user")
    key = request.GET.get("key")
    returned = logger.createNewUser(request, user, key) # Call this
    return HttpResponse(returned)


def append(request):
    logger = Logger()
    fakeJson = {"appID": "FUNF", "payload": "FUNF_dummyPayload"} #
    get this from caller
    returned = logger.append(request, fakeJson)
    return HttpResponse(returned)


def check(request):
    key = request.GET.get("key") # Get from the caller
    logger = Logger()
    returned = logger.check(request, key)
    return HttpResponse(returned)
```

code/loggerModule.py

Bibliography

[46-] Utah Code 46-3-103(38).

[AA01] Dakshi Agrawal and Charu C Aggarwal. On the design and quantification of privacy preserving data mining algorithms. In *Proceedings of the twentieth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 247–255. ACM, 2001.

[AA11] Charu C Aggarwal and Tarek Abdelzaher. Social sensing. *Managing and Mining Sensor Data*, 2011.

[AAE$^+$11] Y. Altshuler, N. Aharony, Y. Elovici, A. Pentland, and M. Cebrian. Stealing reality: when criminals become data scientists (or vice versa). *Security and Privacy in Social Networks*, pages 133–151, 2011.

[ABK09] S. Avancha, A. Baxi, and D. Kotz. Privacy in mobile technology for personal healthcare. *Submitted to ACM Computing Surveys*, 2009.

[Acc11] Rafael Accorsi. Bbox: a distributed secure log architecture. In *Public Key Infrastructures, Services and Applications*, pages 109–124. Springer, 2011.

[AG05] A. Acquisti and J. Grossklags. Privacy and rationality in individual decision making. *Security & Privacy, IEEE*, 3(1):26–33, 2005.

[AHFG10] Amittai Aviram, Sen Hu, Bryan Ford, and Ramakrishna Gummadi. Determinating timing channels in compute clouds. In *Proceedings of the 2010 ACM workshop on Cloud computing security workshop*, pages 103–108. ACM, 2010.

[AN10] I. Aad and V. Niemi. Nrc data collection and the privacy by design principles. *Proc. of PhoneSense*, pages 41–45, 2010.

[API$^+$11] N. Aharony, W. Pan, C. Ip, I. Khayal, and A. Pentland. Social fmri: Investigating and shaping social mechanisms in the real world. *Pervasive and Mobile Computing*, 2011.

[AS00] R. Agrawal and R. Srikant. Privacy-preserving data mining. In *ACM Sigmod Record*, volume 29, pages 439–450. ACM, 2000.

[BDMN05] Avrim Blum, Cynthia Dwork, Frank McSherry, and Kobbi Nissim. Practical privacy: the sulq framework. In *Proceedings of the twenty-fourth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 128–138. ACM, 2005.

[BK10] R. Böhme and S. Köpsell. Trained to accept?: A field experiment on consent dialogs. In *Proceedings of the 28th international conference on Human factors in computing systems*, pages 2403–2406. ACM, 2010.

[BL96] Dan Boneh and Richard Lipton. A revocable backup system. In *USENIX Security Symposium*, pages 91–96, 1996.

[BLS01] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the weil pairing. In *Advances in Cryptology—ASIACRYPT 2001*, pages 514–532. Springer, 2001.

[BY97] Mihir Bellare and Bennet Yee. Forward integrity for secure audit logs. Technical report, Citeseer, 1997.

[BZH06] M. Barbaro, T. Zeller, and S. Hansell. A face is exposed for aol searcher no. 4417749. *New York Times*, 9(2008):8For, 2006.

[Cat97] Fred H Cate. *Privacy in the information age*. Brookings Inst Press, 1997.

[CDCFK11] Claude Castelluccia, Emiliano De Cristofaro, Aurélien Francillon, and M-A Kaafar. Ephpub: Toward robust ephemeral publishing. In *Network Protocols (ICNP), 2011 19th IEEE International Conference on*, pages 165–175. IEEE, 2011.

[CDM$^+$05] Shuchi Chawla, Cynthia Dwork, Frank McSherry, Adam Smith, and Hoeteck Wee. Toward privacy in public databases. *Theory of Cryptography*, pages 363–385, 2005.

[CDMT12] Shuchi Chawla, Cynthia Dwork, Frank McSherry, and Kunal Talwar. On privacy-preserving histograms. *arXiv preprint arXiv:1207.1371*, 2012.

[CEE11] Ann Cavoukian and Khaled El Emam. *Dispelling the Myths Surrounding De-identification: Anonymization Remains a Strong Tool for Protecting Privacy*. Information and Privacy Commissioner of Ontario, Canada, 2011.

[CKK+08] C. Cornelius, A. Kapadia, D. Kotz, D. Peebles, M. Shin, and N. Triandopoulos. Anonysense: privacy-aware people-centric sensing. In *Proceedings of the 6th international conference on Mobile systems, applications, and services*, pages 211–224. ACM, 2008.

[CMP09] I. Chronis, A. Madan, and A.S. Pentland. Socialcircuits: the art of using mobile phones for modeling personal interactions. In *Proceedings of the ICMI-MLMI'09 Workshop on Multimodal Sensor-Based Systems and Mobile Phones for Social Computing*, page 1. ACM, 2009.

[CPH03] Cheun N Chong, Zhonghong Peng, and Pieter H Hartel. Secure audit logging with tamper-resistant hardware. In *18th IFIP International Information Security Conference (IFIPSEC)*, volume 250, pages 73–84, 2003.

[Cra06] L.F. Cranor. What do they indicate?: evaluating security and privacy indicators. *interactions*, 13(3):45–47, 2006.

[CRI10] Narendran Calluru Rajasekar and Chris Imafidon. Exploitation of vulnerabilities in cloud storage. In *CLOUD COMPUTING 2010, The First International Conference on Cloud Computing, GRIDs, and Virtualization*, pages 122–127, 2010.

[CW09] Scott A Crosby and Dan S Wallach. Efficient data structures for tamper evident logging. In *Proc. 18th USENIX Security Symposium*, 2009.

[DC10] Benjamin Davis and Hao Chen. Dbtaint: cross-application information flow tracking via databases. In *2010 USENIX Conference on Web Application Development*, 2010.

[Dem11] Brian Demsky. Cross-application data provenance and policy enforcement. *ACM Transactions on Information and System Security (TISSEC)*, 14(1):6, 2011.

[Den76] Dorothy E Denning. A lattice model of secure information flow. *Communications of the ACM*, 19(5):236–243, 1976.

[DKM+06] Cynthia Dwork, Krishnaram Kenthapadi, Frank McSherry, Ilya Mironov, and Moni Naor. Our data, ourselves: Privacy via distributed noise generation. *Advances in Cryptology-EUROCRYPT 2006*, pages 486–503, 2006.

[dMHVB13] Yves-Alexandre de Montjoye, César A Hidalgo, Michel Verleysen, and Vincent D Blondel. Unique in the crowd: The privacy bounds of human mobility. *Scientific reports*, 3, 2013.

[dMQRP13] Yves-Alexandre de Montjoye, Jordi Quoidbach, Florent Robic, and Alex Sandy Pentland. Predicting personality using novel mobile phone-based metrics. In *Social Computing, Behavioral-Cultural Modeling and Prediction*, pages 48–55. Springer, 2013.

[DN03] Irit Dinur and Kobbi Nissim. Revealing information while preserving privacy. In *Proceedings of the twenty-second ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 202–210. ACM, 2003.

[DN04] Cynthia Dwork and Kobbi Nissim. Privacy-preserving datamining on vertically partitioned databases. In *Advances in Cryptology–CRYPTO 2004*, pages 134–138. Springer, 2004.

[Duc10] Matt Duckham. Moving forward: location privacy and location awareness. In *Proceedings of the 3rd ACM SIGSPATIAL International Workshop on Security and Privacy in GIS and LBS*, pages 1–3. ACM, 2010.

[EFW12] S. Egelman, A.P. Felt, and D. Wagner. Choice architecture and smartphone privacy: There's a price for that. In *Workshop on the Economics of Information Security (WEIS)*, 2012.

[EGS03] Alexandre Evfimievski, Johannes Gehrke, and Ramakrishnan Srikant. Limiting privacy breaches in privacy preserving data mining. In *Proceedings of the twenty-second ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 211–222. ACM, 2003.

[EKV$^+$05] Petros Efstathopoulos, Maxwell Krohn, Steve VanDeBogart, Cliff Frey, David Ziegler, Eddie Kohler, David Mazieres, Frans Kaashoek, and Robert Morris. Labels and event processes in the asbestos operating system. *ACM SIGOPS Operating Systems Review*, 39(5):17–30, 2005.

[EP03] Nathan Eagle and Alex Pentland. Social network computing. In *UbiComp 2003: Ubiquitous Computing*, pages 289–296. Springer, 2003.

[EP06] N. Eagle and A. Pentland. Reality mining: sensing complex social systems. *Personal and Ubiquitous Computing*, 10(4):255–268, 2006.

[EPL09] Nathan Eagle, Alex Sandy Pentland, and David Lazer. Inferring friendship network structure by using mobile phone data. *Proceedings of the National Academy of Sciences*, 106(36):15274–15278, 2009.

[FDH$^+$12] M Franz, B Deiseroth, K Hamacher, S Jha, S Katzenbeisser, and H SchröDer. Secure computations on non-integer values with applications to privacy-preserving sequence analysis. *Information Security Technical Report*, 2012.

[FGW11] A.P. Felt, K. Greenwood, and D. Wagner. The effectiveness of application permissions. In *Proc. of the USENIX Conference on Web Application Development*, 2011.

[FHE$^+$12] A.P. Felt, E. Ha, S. Egelman, A. Haney, E. Chin, and D. Wagner. Android permissions: User attention, comprehension, and behavior. In *Proceedings of the Eighth Symposium on Usable Privacy and Security*, page 3. ACM, 2012.

[FIP] `http://csrc.nist.gov/publications/fips/fips180-4/fips-180-4.pdf`.

[FLM05] B. Friedman, P. Lin, and J.K. Miller. Informed consent by design. *Security and Usability*, pages 495–521, 2005.

[FLR12] Niroshinie Fernando, Seng W Loke, and Wenny Rahayu. Mobile cloud computing: A survey. *Future Generation Computer Systems*, 2012.

[Fra06] Matt Franklin. A survey of key evolving cryptosystems. *International Journal of Security and Networks*, 1(1):46–53, 2006.

[FSK12] Niels Ferguson, Bruce Schneier, and Tadayoshi Kohno. *Cryptography engineering: design principles and practical applications*. Wiley, 2012.

[FWCY10] Benjamin Fung, Ke Wang, Rui Chen, and Philip S Yu. Privacy-preserving data publishing: A survey of recent developments. *ACM Computing Surveys (CSUR)*, 42(4):14, 2010.

[GBL$^+$02] Jim Gemmell, Gordon Bell, Roger Lueder, Steven Drucker, and Curtis Wong. Mylifebits: fulfilling the memex vision. In *Proceedings of the tenth ACM international conference on Multimedia*, pages 235–238. ACM, 2002.

[GBL06] Jim Gemmell, Gordon Bell, and Roger Lueder. Mylifebits: a personal database for everything. *Communications of the ACM*, 49(1):88–95, 2006.

[Gen09] Craig Gentry. *A fully homomorphic encryption scheme*. PhD thesis, Stanford University, 2009.

[GKLL09] Roxana Geambasu, Tadayoshi Kohno, Amit Levy, and Henry M Levy. Vanish: Increasing data privacy with self-destructing data. In *Proc. of the 18th USENIX Security Symposium*, page 56, 2009.

[GL12] Afshar Ganjali and David Lie. Auditing cloud management using information flow tracking. In *Proceedings of the seventh ACM workshop on Scalable trusted computing*, pages 79–84. ACM, 2012.

[GPA+10] R.K. Ganti, N. Pham, H. Ahmadi, S. Nangia, and T.F. Abdelzaher. Greengps: A participatory sensing fuel-efficient maps application. In *Proceedings of the 8th international conference on Mobile systems, applications, and services*, pages 151–164. ACM, 2010.

[GPTA08] Raghu K Ganti, Nam Pham, Yu-En Tsai, and Tarek F Abdelzaher. Poolview: stream privacy for grassroots participatory sensing. In *Proceedings of the 6th ACM conference on Embedded network sensor systems*, pages 281–294. ACM, 2008.

[GUA] `http://www.guardian.co.uk/world/2013/jun/06/nsa-phone-records-verizon-court-order`.

[HBZ+06] Bret Hull, Vladimir Bychkovsky, Yang Zhang, Kevin Chen, Michel Goraczko, Allen Miu, Eugene Shih, Hari Balakrishnan, and Samuel Madden. Cartel: a distributed mobile sensor computing system. In *Proceedings of the 4th international conference on Embedded networked sensor systems*, pages 125–138. ACM, 2006.

[HL04] J.I. Hong and J.A. Landay. An architecture for privacy-sensitive ubiquitous computing. In *Proceedings of the 2nd international conference on Mobile systems, applications, and services*, pages 177–189. ACM, 2004.

[HNB11] Brian Hay, Kara Nance, and Matt Bishop. Storm clouds rising: security challenges for iaas cloud computing. In *System Sciences (HICSS), 2011 44th Hawaii International Conference on*, pages 1–7. IEEE, 2011.

[Hol06] Jason E Holt. Logcrypt: forward security and public verification for secure audit logs. In *Proceedings of the 2006 Australasian workshops on Grid computing and e-research-Volume 54*, pages 203–211. Australian Computer Society, Inc., 2006.

[HRFMF13] Keiko Hashizume, David G Rosado, Eduardo Fernández-Medina, and Eduardo B Fernandez. An analysis of security issues for cloud computing. *Journal of Internet Services and Applications*, 4(1):5, 2013.

[KAB09] David Kotz, Sasikanth Avancha, and Amit Baxi. A privacy framework for mobile health and home-care systems. In *Proceedings of the first ACM workshop on Security and privacy in medical and home-care systems*, pages 1–12. ACM, 2009.

[KBD$^+$10] N. Kiukkonen, J. Blom, O. Dousse, D. Gatica-Perez, and J. Laurila. Towards rich mobile phone datasets: Lausanne data collection campaign. *Proc. ICPS, Berlin*, 2010.

[KBN11] A. Korth, S. Baumann, and A. Nürnberger. An interdisciplinary problem taxonomy for user privacy in social networking services. In *Workshop on Privacy for a Networked World*, 2011.

[KC06] Samuel T King and Peter M Chen. Subvirt: Implementing malware with virtual machines. In *Security and Privacy, 2006 IEEE Symposium on*, pages 14–pp. IEEE, 2006.

[KCC$^+$09] P. Klasnja, S. Consolvo, T. Choudhury, R. Beckwith, and J. Hightower. Exploring privacy concerns about personal sensing. *Pervasive Computing*, pages 176–183, 2009.

[KFJ03] L. Kagal, T. Finin, and A. Joshi. A policy based approach to security for the semantic web. *The Semantic Web-ISWC 2003*, pages 402–418, 2003.

[KHFK07] Apu Kapadia, Tristan Henderson, Jeffrey Fielding, and David Kotz. Virtual walls: Protecting digital privacy in pervasive environments. *Pervasive Computing*, pages 162–179, 2007.

[KN11] Juuso Karikoski and Matti Nelimarkka. Measuring social relations with multiple datasets. *International Journal of Social Computing and Cyber-Physical Systems*, 1(1):98–113, 2011.

[KO08] V. Kostakos and E. O'Neill. Cityware: Urban computing to bridge online and real-world social networks. *Handbook of research on urban informatics: The practice and promise of the real-time city*, pages 195–204, 2008.

[Kot11] D. Kotz. A threat taxonomy for mhealth privacy. In *Communication Systems and Networks (COMSNETS), 2011 Third International Conference on*, pages 1–6. IEEE, 2011.

[KPSW11] B Konings, David Piendl, Florian Schaub, and Michael Weber. Privacyjudge: Effective privacy controls for online published information. In *Privacy, security, risk and trust (passat), 2011 ieee third international conference on and 2011 ieee third international conference on social computing (socialcom)*, pages 935–941. IEEE, 2011.

[Kru09] J. Krumm. A survey of computational location privacy. *Personal and Ubiquitous Computing*, 13(6):391–399, 2009.

[KTC$^+$08] A. Kapadia, N. Triandopoulos, C. Cornelius, D. Peebles, and D. Kotz. Anonysense: Opportunistic and privacy-preserving context collection. *Pervasive Computing*, pages 280–297, 2008.

[KYB$^+$07] Maxwell Krohn, Alexander Yip, Micah Brodsky, Natan Cliffer, M Frans Kaashoek, Eddie Kohler, and Robert Morris. Information flow control for standard os abstractions. In *ACM SIGOPS Operating Systems Review*, volume 41, pages 321–334. ACM, 2007.

[KZTO05] Pandurang Kamat, Yanyong Zhang, Wade Trappe, and Celal Ozturk. Enhancing source-location privacy in sensor network routing. In *Distributed Computing Systems, 2005. ICDCS 2005. Proceedings. 25th IEEE International Conference on*, pages 599–608. IEEE, 2005.

[LCW$^+$11] Janne Lindqvist, Justin Cranshaw, Jason Wiese, Jason Hong, and John Zimmerman. I'm the mayor of my house: examining why people use foursquare-a social-driven location sharing application. In *Proceedings of the 2011 annual conference on Human factors in computing systems*, pages 2409–2418. ACM, 2011.

[LGPA$^+$12] J.K. Laurila, D. Gatica-Perez, I. Aad, J. Blom, O. Bornet, T.M.T. Do, O. Dousse, J. Eberle, and M. Miettinen. The mobile data challenge: Big data for mobile computing research. In *Mobile Data Challenge by Nokia Workshop, in conjunction with Int. Conf. on Pervasive Computing, Newcastle, UK*, 2012.

[LL09] T. Li and N. Li. On the tradeoff between privacy and utility in data publishing. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 517–526. ACM, 2009.

[LLV07] N. Li, T. Li, and S. Venkatasubramanian. t-closeness: Privacy beyond k-anonymity and l-diversity. In *Data Engineering, 2007. ICDE 2007. IEEE 23rd International Conference on*, pages 106–115. IEEE, 2007.

[LPA$^+$09] D. Lazer, A.S. Pentland, L. Adamic, S. Aral, A.L. Barabasi, D. Brewer, N. Christakis, N. Contractor, J. Fowler, M. Gutmann, et al. Life in the network: the coming age of computational social science. *Science (New York, NY)*, 323(5915):721, 2009.

[LXMZ12] N.D. Lane, J. Xie, T. Moscibroda, and F. Zhao. On the feasibility of user de-anonymization from shared mobile sensor data. In *Proceedings of the Third International Workshop on Sensing Applications on Mobile Phones*, page 3. ACM, 2012.

[MCM$^+$11] A. Madan, M. Cebrian, S. Moturu, K. Farrahi, and S. Pentland. Sensing the 'health state'of a community. *Pervasive Computing*, 2011.

[MCR$^+$10] E. Miluzzo, C.T. Cornelius, A. Ramaswamy, T. Choudhury, Z. Liu, and A.T. Campbell. Darwin phones: the evolution of sensing and inference on mobile phones. In *Proceedings of the 8th international* *conference on Mobile systems, applications, and services*, pages 5–20. ACM, 2010.

[MFGPP11] A. Madan, K. Farrahi, D. Gatica-Perez, and A. Pentland. Pervasive sensing to model political opinions in face-to-face networks. *Pervasive Computing*, pages 214–231, 2011.

[MKGV07] A. Machanavajjhala, D. Kifer, J. Gehrke, and M. Venkitasubramaniam. l-diversity: Privacy beyond k-anonymity. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 1(1):3, 2007.

[MKHS08] H. Mahato, D. Kern, P. Holleis, and A. Schmidt. Implicit personalization of public environments using bluetooth. In *CHI'08 extended abstracts on Human factors in computing systems*, pages 3093–3098. ACM, 2008.

[ML97] Andrew C Myers and Barbara Liskov. A decentralized model for information flow control. In *ACM SIGOPS Operating Systems Review*, volume 31, pages 129–142. ACM, 1997.

[MLA12] A. Mazzia, K. LeFevre, and E. Adar. The pviz comprehension tool for social network privacy settings. In *Proceedings of the Eighth Symposium on Usable Privacy and Security*, page 13. ACM, 2012.

[MLF$^+$08] E. Miluzzo, N.D. Lane, K. Fodor, R. Peterson, H. Lu, M. Musolesi, S.B. Eisenman, X. Zheng, and A.T. Campbell. Sensing meets mobile social networks: the design, implementation and evaluation of the cenceme application. In *Proceedings of the 6th ACM conference on Embedded network sensor systems*, pages 337–350. ACM, 2008.

[MMLP10] Anmol Madan, Sai T Moturu, David Lazer, and Alex Sandy Pentland. Social sensing: obesity, unhealthy eating and exercise in face-to-face networks. In *Wireless Health 2010*, pages 104–110. ACM, 2010.

[MRF11] Yogesh Mundada, Anirudh Ramachandran, and Nick Feamster. Silverline: Data and network isolation for cloud services. *Proc. of 3rd HotCloud*, 2011.

[MSF09] A.D. Molina, M. Salajegheh, and K. Fu. Hiccups: health information collaborative collection using privacy and security. In *Proceedings of the first ACM workshop on Security and privacy in medical and home-care systems*, pages 21–30. ACM, 2009.

[MT07] Di Ma and Gene Tsudik. Forward-secure sequential aggregate authentication. In *Security and Privacy, 2007. SP'07. IEEE Symposium on*, pages 86–91. IEEE, 2007.

[MT09] Di Ma and Gene Tsudik. A new approach to secure logging. *ACM Transactions on Storage (TOS)*, 5(1):2, 2009.

[MVGD10] Alan Mislove, Bimal Viswanath, Krishna P Gummadi, and Peter Druschel. You are who you know: inferring user profiles in online social networks. In *Proceedings of the third ACM international conference on Web search and data mining*, pages 251–260. ACM, 2010.

[NLV11] Michael Naehrig, Kristin Lauter, and Vinod Vaikuntanathan. Can homomorphic encryption be practical? In *Proceedings of the 3rd ACM workshop on Cloud computing security workshop*, pages 113–124. ACM, 2011.

[NRP] `http://en.wikipedia.org/wiki/Non-repudiation`.

[NS95] Moni Naor and Adi Shamir. Visual cryptography. In *Advances in Cryptology—EUROCRYPT'94*, pages 1–12. Springer, 1995.

[NS08] Arvind Narayanan and Vitaly Shmatikov. Robust de-anonymization of large sparse datasets. In *Security and Privacy, 2008. SP 2008. IEEE Symposium on*, pages 111–125. IEEE, 2008.

[Ohm10] Paul Ohm. Broken promises of privacy: Responding to the surprising failure of anonymization. *UCLA Law Review*, 57:1701, 2010.

[OMCP11] Daniel Olguín, Anmol Madan, Manuel Cebrian, and Alex Pentland. Mobile sensing technologies and computational methods for collective intelligence. *Next Generation Data Technologies for Collective Computational Intelligence*, pages 575–597, 2011.

[OWK$^+$09] D.O. Olguín, B.N. Waber, T. Kim, A. Mohan, K. Ara, and A. Pentland. Sensible organizations: Technology and methodology for automatically measuring organizational behavior. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 39(1):43–55, 2009.

[Pag09] Thomas Page. *The application of hash chains and hash structures to cryptography*. PhD thesis, Citeseer, 2009.

[PBB09] R.A. Popa, H. Balakrishnan, and A. Blumberg. Vpriv: protecting privacy in location-based vehicular services. In *Proceedings of the 18th conference on USENIX security symposium*, pages 335–350. USENIX Association, 2009.

[Pen08] Alex Sandy Pentland. *Honest signals: how they shape our world*. MIT Press, 2008.

[Per05a] Radia Perlman. The ephemerizer: Making data disappear. ., 2005.

[Per05b] Radia Perlman. File system design with assured delete. In *Security in Storage Workshop, 2005. SISW'05. Third IEEE International*, pages 6–pp. IEEE, 2005.

[PKZ$^+$12] Vasilis Pappas, Vasileios Kemerlis, Angeliki Zavou, Michalis Polychronakis, and Angelos D Keromytis. Cloudfence: Enabling users to audit the use of their cloud-resident data. ., 2012.

[PTT08] Charalampos Papamanthou, Roberto Tamassia, and Nikos Triandopoulos. Authenticated hash tables. In *Proceedings of the 15th ACM conference on Computer and communications security*, pages 437–448. ACM, 2008.

[QC09] D. Quercia and L. Capra. Friendsensing: recommending friends using mobile phones. In *Proceedings of the third ACM conference on Recommender systems*, pages 273–276. ACM, 2009.

[RAD78] Ronald L Rivest, Len Adleman, and Michael L Dertouzos. On data banks and privacy homomorphisms. *Foundations of secure computation*, 4(11):169–180, 1978.

[RBS$^+$12] Indrajit Ray, Kirill Belyaev, Mikhail Strizhov, Dieudonne Mulamba, and Mariappan Rajaram. Secure logging as a service—delegating log management to the cloud. ., 2012.

[RGKS11] A. Raij, A. Ghosh, S. Kumar, and M. Srivastava. Privacy risks emerging from the adoption of innocuous wearable sensors in the mobile environment. In *Proceedings of the 2011 annual conference on Human factors in computing systems*, pages 11–20. ACM, 2011.

[ROE09] Mika Raento, Antti Oulasvirta, and Nathan Eagle. Smartphones an emerging tool for social scientists. *Sociological methods & research*, 37(3):426–454, 2009.

[RPD$^+$06] Deb Roy, Rupal Patel, Philip DeCamp, Rony Kubat, Michael Fleischman, Brandon Roy, Nikolaos Mavridis, Stefanie Tellex, Alexia Salata, Jethran Guinness, et al. The human speechome project. *Symbol Grounding and Beyond*, pages 192–196, 2006.

[RTSS09] Thomas Ristenpart, Eran Tromer, Hovav Shacham, and Stefan Savage. Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds. In *Proceedings of the 16th ACM conference on Computer and communications security*, pages 199–212. ACM, 2009.

[SARL10] Lilia Sfaxi, Takoua Abdellatif, Riadh Robbana, and Yassine Lakhnech. Information flow control of component-based distributed systems. *Concurrency and Computation: Practice and Experience*, 2010.

[SGA13] Fred Stutzman, Ralph Gross, and Alessandro Acquisti. Silent listeners: The evolution of privacy and disclosure on facebook. *Journal of Privacy and Confidentiality*, 4(2):2, 2013.

[SH09] Karen Scarfone and Paul Hoffman. Guidelines on firewalls and firewall policy. *NIST Special Publication*, 800:41, 2009.

[SH12] Mudhakar Srivatsa and Mike Hicks. Deanonymizing mobility traces: using social network as a side-channel. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 628–637. ACM, 2012.

[SHA] `http://goo.gl/6vkN8`.

[Sha85] Adi Shamir. Identity-based cryptosystems and signature schemes. In *Advances in cryptology*, pages 47–53. Springer, 1985.

[Sha06] Paul Shabajee. Informed consent on the semantic web-issues for interaction and interface designers. In *3rd International Semantic Web User Interaction Workshop.< http://swui. semanticweb. org/swui06/papers/Shabajee/Shabajee. pdf>(retrieved 15.11. 10)*, 2006.

[SK99] Bruce Schneier and John Kelsey. Secure audit logs to support computer forensics. *ACM Transactions on Information and System Security (TISSEC)*, 2(2):159–176, 1999.

[SLC$^+$11] Amre Shakimov, Harold Lim, Ramón Cáceres, Landon P Cox, Kevin Li, Dongtao Liu, and Alexander Varshavsky. Vis-a-vis: Privacy-preserving online social networking via virtual individual servers. In *Communication Systems and Networks (COMSNETS), 2011 Third International Conference on*, pages 1–10. IEEE, 2011.

[SM07] Karen Scarfone and Peter Mell. Guide to intrusion detection and prevention systems (idps). *NIST Special Publication*, 800(2007):94, 2007.

[SO13] R.B. Shapiro and P.N. Ossorio. Regulation of online social network studies. *Science*, 339(6116):144–145, 2013.

[STLBH11] Reza Shokri, George Theodorakopoulos, J Le Boudec, and J Hubaux. Quantifying location privacy. In *Security and Privacy (SP), 2011 IEEE Symposium on*, pages 247–262. IEEE, 2011.

[Swe00] Latanya Sweeney. Simple demographics often identify people uniquely. *Health (San Francisco)*, pages 1–34, 2000.

[Swe02] L. Sweeney. k-anonymity: A model for protecting privacy. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 10(05):557–570, 2002.

[TCD$^+$10] Eran Toch, Justin Cranshaw, Paul Hankes Drielsma, Janice Y Tsai, Patrick Gage Kelley, James Springfield, Lorrie Cranor, Jason Hong, and Norman Sadeh. Empirical models of privacy in location sharing. In *Proceedings of the 12th ACM international conference on Ubiquitous computing*, pages 129–138, 2010.

[TEHEG12] Maha TEBAA, Saïd EL HAJJI, and Abdellatif EL GHAZI. Homomorphic encryption applied to the cloud computing security. *Lecture Notes in Engineering and Computer Science*, 2197, 2012.

[TLLP12] Yang Tang, P Lee, J Lui, and Radia Perlman. Secure overlay cloud storage with access control and assured deletion. ., 2012.

[TP12] O. Tene and J. Polonetsky. Big data for all: Privacy and user control in the age of analytics. *Northwestern Journal of Technology and Intellectual Property, Forthcoming*, 2012.

[TSH10] E. Toch, N.M. Sadeh, and J. Hong. Generating default privacy policies for online social networks. In *Proceedings of the 28th of the international conference extended abstracts on Human factors in computing systems*, pages 4243–4248. ACM, 2010.

[URM$^+$12] Marion K Underwood, Lisa H Rosen, David More, Samuel E Ehrenreich, and Joanna K Gentsch. The blackberry project: Capturing the content of adolescents' text messaging. *Developmental psychology*, 48(2):295, 2012.

[Var12] Various Authors. The value of privacy. *MIT Techonology Review - Business Report*, June 2012.

[VGWR12] Soroush Vosoughi, Matthew S Goodwin, Bill Washabaugh, and Deb Roy. A portable audio/video recorder for longitudinal study of child development. In *Proceedings of the 14th ACM international conference on Multimodal interaction*, pages 193–200. ACM, 2012.

[VIS96] Dennis Volpano, Cynthia Irvine, and Geoffrey Smith. A sound type system for secure flow analysis. *Journal of computer security*, 4(2):167–187, 1996.

[WAT] `http://en.wikipedia.org/wiki/Attack-(computing)`.

[WB90] Samuel D Warren and Louis D Brandeis. The right to privacy. *Harvard law review*, 4(5):193–220, 1890.

[WBC70] Alan F Westin and Louis Blom-Cooper. *Privacy and freedom*. Bodley Head London, 1970.

[WBDS04] Brent R Waters, Dirk Balfanz, Glenn Durfee, and Diana K Smetters. Building an encrypted and searchable audit log. In *Proceedings of 11th Annual Network and Distributed System Security Symposium (NDSS 2004)*, volume 6, 2004.

[WHH$^+$10] Scott Wolchok, Owen S Hofmann, Nadia Heninger, Edward W Felten, J Alex Halderman, Christopher J Rossbach, Brent Waters, and Emmett Witchel. Defeating vanish with low-cost sybil attacks against large dhts. In *Proc. of NDSS*, 2010.

[Wou12] Karel Wouters. Hash-chain based protocols for time-stamping and secure logging: formats, analysis and design. ., 2012.

[YL10] P. Ypodimatopoulos and A. Lippman. 'follow me': a web-based, location-sharing architecture for large, indoor environments. In *Proceedings of the 19th international conference on World wide web*, pages 1375–1378. ACM, 2010.

[YN09] Attila A Yavuz and Peng Ning. Baf: An efficient publicly verifiable secure audit logging scheme for distributed systems. In *Computer Security Applications Conference, 2009. ACSAC'09. Annual*, pages 219–228. IEEE, 2009.

[ZBWKM06] Nickolai Zeldovich, Silas Boyd-Wickizer, Eddie Kohler, and David Mazières. Making information flow explicit in histar. In *Proceedings of the 7th symposium on Operating systems design and implementation*, pages 263–278. USENIX Association, 2006.

[ZBWM08] Nickolai Zeldovich, Silas Boyd-Wickizer, and David Mazieres. Securing distributed systems with information flow control. In *.* NSDI, 2008.

[Zda02] Stephan Arthur Zdancewic. *Programming languages for information security*. PhD thesis, Cornell University, 2002.

[ZJRR12] Yinqian Zhang, Ari Juels, Michael K Reiter, and Thomas Ristenpart. Cross-vm side channels and their use to extract private keys. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 305–316. ACM, 2012.

[ZMM$^+$10] Qing Zhang, John McCullough, Justin Ma, Nabil Schear, Michael Vrable, Amin Vahdat, Alex C Snoeren, Geoffrey M Voelker, and Stefan Savage. Neon: system support for derived data management. In *ACM Sigplan Notices*, volume 45, pages 63–74. ACM, 2010.