Analysis of Data-At-Rest Security In Smartphones

by

Ildar Muslukhov

B. Information Technology, Ufa State Aviation and Technical University, 2003
M. Information Technology, Ufa State Aviation and Technical University, 2005
Ph.D., Ufa State Aviation and Technical University, 2008

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF

Doctor of Philosophy

in

THE FACULTY OF GRADUATE AND POSTDOCTORAL STUDIES
(Electrical and Computer Engineering)

The University of British Columbia
(Vancouver)

August 2018

The following individuals certify that they have read, and recommend to the Faculty of Graduate and Postdoctoral Studies for acceptance, the dissertation entitled:

Analysis of Data-At-Rest Security In Smartphones

submitted by Ildar Muslukhov in partial fulfillment of the requirements for

the degree of Doctor of Philosophy

in Electrical and Computer Engineering

Examining Committee:

Prof Konstantin Beznosov
Supervisor

Prof Sidney Fels
Supervisory Committee Member

Prof Julia Rubin
Supervisory Committee Member

Prof Sathish Gopalakrishnan
University Examiner

Prof Reid Holmes
University Examiner

Abstract

With almost two billion users worldwide, smartphones are used for almost everything – booking a hotel, ordering a cup of coffee, or paying in a shop. However, small size and high mobility makes these devices prone to theft and loss. In this work we aim to broaden our understanding of how smartphone users and application developers protect sensitive data on smartphones.

To understand how well users are protecting their data in smartphones, we conducted several studies. The results revealed that 50% of the subjects locked their smartphone with an unlocking secret and 95% of them chose unlocking secrets that could be guessed within minutes.

To understand how well application developers protect sensitive data in smartphones, we analyzed 132K Android applications. We focused on identifying misuse of cryptography in applications and libraries. The study results revealed that developers often misuse cryptographic API. In fact, 9 out of 10 Android applications contained code that used a symmetric cipher with a static encryption key. Further, source attribution revealed that libraries are the main consumer of cryptography and the major contributor of misuse cases. Finally, an in-depth analysis of the top libraries highlighted the need for improvement in the way we define and detect misuse of cryptography.

Based on these results we designed and evaluated a system for encryption keys management that uses wearable devices as an additional source of entropy. Evaluation results showed that the proposal introduces insignificant overhead in power consumption and latency.

Lay Summary

This thesis presents the results of research on how secure user data in smartphones against data thieves. We studied this question from two perspectives, i.e., users and application developers. With end-users we focused on how they choose their passwords to lock smartphones. The results of the study revealed that more than 95% of users choose passwords that are easy to guess, i.e., a thief can guess it in under an hour. With application developers we looked at how often developers put user data at risk, by incorrectly using certain security functions. Our findings show that application developers do put user data at risk. Overall, this research shows that when it comes to data security in smartphones, we are still far from having adequate protection.

Preface

This research was the product of a fruitful collaboration between the author of the dissertation and the following people: Yazan Boshmaf, San-Tsai Sun, Primal Wijesekera, Ivan Cherepau and Konstantin Beznosov (advisor) from the University of British Columbia, and Cynthia Kuo and Jonathan Lester from Nokia Research. I am deeply grateful to my mentors Michael Halcrow and Andrew Honig from Google for an opportunity to work on improving Linux Kernel fuzzing and adding encryption to the EXT4 file system.

Work presented herein consists of research studies that have been published or are under review in peer-reviewed international conferences and workshops.

The user studies on characterization of smartphone end users presented in Chapter 2, and partly discussed in Chapter 5, led to the following publications:

- I. Muslukhov, Y. Boshmaf, C. Kuo, J. Lester, K. Beznosov. Understanding Users? Requirements for Data Protection in Smartphones. In *Proceedings of Data Engineering Workshops of the 28th IEEE International Conference on Data Engineering (ICDEW'12)*, pp. 228–235, Arlington, VA, USA, 2012.

- I. Muslukhov, Y. Boshmaf, C. Kuo, J. Lester, K. Beznosov. Know Your Enemy: The Risk of Unauthorized Access in Smartphones by Insiders. In *Proceedings of the 15th International Conference on Human-Computer Interaction with Mobile Devices and Services*, Munich, Germany, 2013, Pages 271-280, 22% acceptance rate.

- I. Cherepau, I. Muslukhov, N. Asanka, and K. Beznosov. On The Impact of Touch ID on iPhone Passcodes. In *Proceedings of the 11th Symposium On Usable Privacy and Security (SOUPS '15)*, Ottawa, ON, Canada, 2015, Pages 257-276, 24% acceptance rate.

- D. Marques, I. Muslukhov, T. Guerreiro, L. Carriço, K. Beznosov. Snooping on Mobile Phones: Prevalence and Trends. In *Proceedings of the 12th Symposium On Usable Privacy and Security (SOUPS '16)*, Denver, CO, US, 2016, 28% acceptance rate. Distinguished Paper Award.

- A. Mahfouz, I. Muslukhov, and K. Beznosov. Android users in the wild: Their authentication and usage behavior. Pervasive and Mobile Computing. *Special Issue on Mobile Security, Privacy and Forensics*, Volume 32, Pages 50-61, Elsevier, October 2016.

I was responsible for designing and conducting the interview-based user study, where Yazan Boshmaf actively participated in the interviewing process. Me and Yazan separately coded interview data to reduce personal bias. Other project members actively participated in the discussion of the interview guide, discussion of the data analysis, and the paper writing process (the first paper on the list above). For this study I obtained ethics approval from the Behavioural Research Ethics Board (BREB) at UBC. Approval H11-02230, titled "Mobile Data Protection."

After the completion of the interview-based study, I proceeded with the design of an online survey. Administration of the survey and data analysis was done by me. All co-authors actively participated in the discussion of the survey structure, questions, results and paper writing process (the second paper on the list above). For this follow-up study I obtained ethics approval from the BREB. Approval H11-03512, titled "Mobile Data Protection - Follow up."

I have significantly contributed to the study design, data analysis and paper writing process for papers three and five on the list above. For both of these publications we obtained approvals from BREB. Approvals H12-02254 titled "Smartphone Unlock in a Wild" and H14-02759 titled "TouchID."

My contributions to the publication with Diogo Marques were limited to the discussion of research questions, study design and paper writing process. I did not participate in data analysis and data collection processes.

The measurement study on how smartphone applications (mis)use cryptographic API, presented in chapter 3, resulted in the following publication:

- I. Muslukhov, Y. Boshmaf and K. Beznosov. Source Attribution of Cryptographic API Misuse in Android Applications. In Proceedings of the 2018 on Asia Conference on Computer and Communications Security (ASIACCS '18). Incheon, Republic of Korea, 2018, Pages 133-146, 20% acceptance rate.

The cryptography (mis)use study was designed, implemented and executed by me. I was also responsible for all data analysis. Co-authors, Yazan Boshmaf and Konstantin Beznosov, contributed to the preliminary research discussion and paper writing process.

The design of the Sidekick system – a *user-space* approach at decoupling data-at-rest encryption and smartphone unlocking, presented in Chapter 4, led to the following publication in a peer-review journal:

- I. Muslukhov, S-T. Sun, P. Wijesekera, Y. Boshmaf, and K. Beznosov. Using Wearable Devices to Secure Data-At-Rest in Stolen Tablets and Smartphones. Pervasive and Mobile Computing. *Special Issue on Mobile Security, Privacy and Forensics*, Volume 32, Pages 26-34, Elsevier, October 2016.

Work on this project was mainly done during my collaboration with the Fusionpipe company through Engage and Engage+ grants. The idea of the project was conceived by me through discussion of the needs of Fusionpipe's clients. All co-authors contributed to the discussion of research questions, study design and paper writing process.

While working on the research presented in this thesis, I also participated in relevant industry led projects, that resulted in the following patents:

- H. Khosravi, I. Muslukhov, P. Luong. Method and System for Decoupling User Authentication and Data Encryption on Mobile Devices. US Patent Application. 13/943,070, Patent number US20140321641 A1. Publication date 16 July, 2013.

- U. Savagaonkar, M. Halcrow, T. Y. Ts'o and I. Muslukhov. Method And System of Encrypting File System Directories. US Patent Application. US Patent Application.US 14/829,095, Patent number US 9639708 B2. Publication date 5 Feb, 2017.

The discussion in Chapter 5 is partially influenced by ideas and findings that led to the following publication:

- Serge Egelman, Andreas Sotirakopoulos, Ildar Muslukhov, Konstantin Beznosov, and Cormac Herley. 2013. Does my password go up to eleven?: the impact of password meters on password selection. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '13). ACM, New York, NY, USA, 2379-2388.

Table of Contents

List of Tables

List of Figures

Acknowledgments

First and foremost, I would like to thank my advisor, Konstantin Beznosov, for giving me the opportunity to venture into different topics and disciplines, and for patiently guiding me through this journey.

Second, I would like to thank all of my collaborators, colleagues and supportive friends. In no particular order, I give my special thanks to you all: Yazan Boshmaf, San-Tsai Sun, Primal Wijesekera, Ivan Cherapau, Cynthia Kuo, Jonathan Lester, Diogo Marques, Ahmed Mahfouz and Lina Qiu.

Third, I would like to thank all members of LERSSE for their feedback and constructive discussions. I would like to thank Ross Sheppard for his invaluable help with proofreading this dissertation.

Fourth, I would like to give special thanks to Dmitry Samosseiko from Sophos for providing vital support for the study of Android applications. I would also like to thank my internships hosts in Google, Andrew Honig and Michael Halcrow, for the opportunity to work on cool projects for Linux OS.

Last but not least, I would like to thank my wife Albina, who patiently supported me during this journey, and my kids David and Daniel, who surrounded me no matter what.

Dedication

To my wonderful family Albina, David and Daniel, who were always beside me and cheered me up no matter what. To my parents and Albina's parents, who were of significant help during Daniel's appearance to this world.

Chapter 1

Introduction

Smartphones have become ubiquitous, highly personal and versatile devices that are used by almost two billion users [15, 24]. Additionally, useful and diverse sets of applications and features, in combination with vast internal storage, have made these devices appealing for organizations [5]. The diverse sets of data that users can store on their smartphones may of course include sensitive or confidential data. For example, photos, videos, emails, or saved passwords are considered sensitive or confidential by various stakeholders. Unfortunately, due to high mobility, smartphones are prone to theft and loss [2, 4], which implies that this sensitive data needs adequate protection.

Indeed, recent reports provide evidence that the threat of smartphone theft and the consequential risk of sensitive information disclosure have a significant impact on companies and users today. In the US, every tenth smartphone owner has been a victim of a smartphone theft at least once [4], more than 30% of all street robberies involve a smartphone [2], the number of stolen smartphones has doubled in 2014, reaching 3.1 million devices [1], and in 96% of cases when a lost device is found, the person who finds it attempts accessing sensitive information on the lost device [9].

One way to protect confidentiality of sensitive data on smartphones is through file system level encryption. For example, the current implementation of full-disk encryption in Android provides such a service as part of the storage IO stack [91].

Another approach is to implement custom, application-specific solutions using a supported cryptographic application program interface (API), or Crypto API for short. For example, application developers can encrypt user data before storing it on a device or transmitting it over a network. Unfortunately, neither one of these approaches is problem-free. Users often choose such unlocking secrets that are vulnerable to unsophisticated password guessing attacks [56, 85].

Significant improvements in general purpose GPU technologies, such as CUDA by NVidia [26] and availability of the tools that can harness GPU's hardware (e.g., HashCat [33]), password guessing attacks became highly practical. For instance, recent experiment demonstrated[1] that NVidia GTX 1070 GPU can probe around four million secret candidates for Android OS. This allows attackers to try all possible combinations for a 6-digit PIN code in under a second. When it comes to iOS, Apple decided to slow down password based key derivation function by employing a specialized hardware with embedded key. Such design allowed them to slow down a single encryption key derivation call to 80 milliseconds [37], which allows probing 1.08 million secrets in a day. Considering that unlocking secret guessing attack for iOS stack is significantly slower, I use iOS encryption key derivation speed as a baseline to define *easy-to-guess* term, which is a secret that can be guessed by trying one million most probable secrets. Note, that if a user chose to use a 6-digit PIN as an unlocking secret, then this definition implies that an attacker will be able to search the entire space.

To mount a password guessing attack an attacker would first obtain a bit-by-bit image of the internal storage [104]. We can safely assume that attackers are aware of formats and data structures for the used file system, thus, knows how to extract the encrypted version of the master key. This assumption is sound, considering that both iOS and Android use well documented file systems. Once the attacker obtains encrypted master key, he launches password guessing attack

[1]Experiment setup and all stats can be found at the following address http://www.netmux.com/blog/how-to-build-a-password-cracking-rig

on external hardware, by trying most probable combinations of the secret and decrypting the master key. The attacker verifies the correctness of the decrypted master key by decrypting content with known data structure. For iOS devices, unlocking secret guessing attack has to be partially executed on the stolen device, since key derivation process iOS relies on the embedded key, which is stored in the specialized hardware. This requires that the attacker is able to hijack the booting sequence, in order to gain control over specialized hardware.

In addition to the aforementioned password guessing attack, *easy-to-guess* secrets are also vulnerable to shoulder surfing attacks [93]. Recent improvements in smartphone authentication technology have targeted certain usability problems with the use of complex secrets. For instance, the Touch ID sensor in iPhones reduces the frequency of secret-based authentications, thus, making it easier for users to choose longer alphanumeric passwords for unlocking secrets. The results of a recent study showed, however, that even when users enable the Touch ID sensor, they still tend to choose *easy-to-guess* unlocking secrets [45].

As for application-specific solutions, the CryptoLint study report from 2012 showed that 88% of Android applications misuse Crypto API [57]. In particular, application developers often use static encryption keys or initialization vectors (IVs), which violate cryptographic notions of security, such as indistinguishability under a chosen-plaintext attack (IND-CPA) [52]. The problem of Crypto API misuse, however, is still far from being fully understood. First, the CryptoLint study was limited in the analysis of libraries, which increases the risk of counting the same bug multiple times, especially in light of recent results from the LibScout study [39], which showed that third-party libraries also misuse Crypto API. Second, the results from the CryptoLint study are five years old, and it is unclear if misuse rates have changed since then, and if they have, in which direction they have changed.

1.1 Goals and Methodology

The main objective of this research is three fold. First, this work aims to widen understanding of issues and threats users face when it comes to protecting their smartphones with an unlocking secret. Second, we aim to provide a deeper analysis of the current state for the problem of Crypto API misuse in mobile applications, especially when it comes to the source of the code from which the misuse originates. Finally, we evaluate technical feasibility of using wearable devices for improving encryption key management system in mobile devices, such as tablets and smartphones.

To achieve these goals, I adopted the following research methodology. First, a set of user studies was conducted to gain a better understanding of the user experience with the existing data protection systems in smartphones. Second, 135,590 Android applications were analyzed to (a) measure how the misuse rates of Crypto API have changed since the CryptoLint study [57], and (b) identify the responsible party for each misuse case, by attributing it to its source, i.e., a third-party library or an application. The results of both studies revealed that the key management subsystem of data encryption in smartphones is the weakest link, which renders data protection insecure. Third, a key management system based on wearable devices was designed, implemented and evaluated in terms of technical impact on data access latency and power consumption.

The results of the user studies showed that smartphone users tend to choose easy-to-guess unlocking secrets, which makes it trivial for an attacker to derive a proper encryption key. The analysis of Android applications revealed that 9 out of 10 applications use static encryption keys, i.e., an attacker can extract these encryption keys with any of the existing tools (such as ApkTool [17]), and, thus, decrypt data. The experimental evaluation of the proposed key management system for smartphones and tables revealed that one can find a trade off between increasing data access latency and power consumption that allows the wearable device to run for more than a year on a single coin cell battery.

1.2 Research Summary

The research presented in this dissertation consists of three parts. The first part presents the results of the studies on user experience in regards to data protection in smartphones. The second part of this thesis presents the results of the analysis of 132K Android applications focusing on the misuse rates of Crypto API. This includes the source attribution analysis results and the results of the trend analysis based on the differences between the applications collected in 2012 and 2016. The third part, contained in Chapter 4, explores a possible approach to addressing the identified limitations of data encryption in smartphones for specialized domains, such as health care. In what follows I summarize an approach taken for each of the research projects and the key findings.

1.2.1 Smartphone Users' Experiences with Data Protection

With the user base of smartphones slowly approaching two billion, there are still open research questions on how to protect users' data in smartphones. In particular, it is not clear what kinds of experiences users have with the existing data protection systems today. To fill this knowledge gap we performed a set of user studies. First, we conducted a set of interviews. The study was designed to collect qualitative data on users' experiences with data protection systems, such as smartphone locking. In modern smartphones, the smartphone locking system is the corner stone for data-at-rest protection, since the unlocking secret is used to protect the data encryption key. If the unlocking secret is *easy-to-guess*, then an attacker can mount an offline password-guessing attack, which eventually will allow the attacker to decrypt all encrypted data. In addition to user experience, we looked at how different types of attackers impacted user perception of associated risks. These interviews allowed us to get a better understanding of a variety of issues that smartphone users face every day. To corroborate the results of the interviews and to assess the prevalence of different experiences, we designed and administered a follow up study in the form of an on-line survey.

The results of the studies revealed that users are divided into three categories (a) those who store sensitive data on their devices and use unlocking secrets (around 50%), (b) those who store sensitive data, but do not use unlocking secrets due to usability issues (20%), and, finally, (c) those who choose not to store sensitive data on smartphones (the remaining 30%).

Both of the above studies broadened our understanding on which data types are considered sensitive and why. In particular, the results showed that personal messages, account credentials, photos, and videos are among the most sensitive types of data. Further, studies showed that data sensitivity depends on the type of adversary. For instance, SMS messages were considered to have a higher sensitivity when the intruder was someone from the victim's acquaintances, i.e., an *insider*. Contact details, on the other hand, were only considered sensitive for *strangers*, i.e., someone who does not know the victim.

The results of the study revealed that while 50% of smartphone users employed authentication-based smartphone locking, 95% of them chose an easy-to-guess unlocking secret. Since unlocking secrets are used to protect data encryption keys, the use of easy-to-guess secrets makes it simple for an attacker to obtain the key by mounting an inexpensive password-guessing attack [56]. The results of a follow up study, led by Ivan Cherapau [45], showed that even if a fingerprint sensor was available, smartphone users still preferred to use easy-to-guess unlocking secrets for various reasons, mainly rooted in usability issues.

Finally, the results of the studies showed that apart from the usual thieves, smartphone users are facing *insider* attackers. In particular, the results obtained in the studies provide evidence that 1 in 10 of smartphone users have accessed someone's device without permission. The accuracy of these results was improved in a follow up study, led by Diogo Marques [83]. In particular, the results of this study revealed a higher rate; that 1 in 5 smartphone users had snooped into someone else's smartphone.

The main contributions of the conducted user studies are the following: (a) investigation of users' experiences with existing data protection systems in smartphones, (b) sensitivity assessment for various types of data in the presence of *strangers* and *insiders*, and (c) measured prevalence of snooping behavior. The results suggest that assumptions made by previous research studies about the safety of certain locations are, to say the least, questionable. For instance, while both Riva et al. [94] and Hayashi et al. [72] have suggested that work or home are safe locations and that smartphones should disable smartphone locking completely at these locations, the results from our studies suggest the opposite; these locations having plenty of insider attackers. Furthermore, because users tend to choose easy-to-guess unlocking secrets for their smartphones, existing data protection systems in smartphones are rendered insecure due to the central role of unlocking secrets in protecting encrypted data.

1.2.2 Analyzing Cryptographic API use in Android Applications

While users can protect their data by enabling encryption[2] and choosing sufficiently complex unlocking secrets, application developers can also play a role in data protection. In particular, application developers can choose to encrypt user data themselves, by calling Crypto API directly. With millions of applications available to smartphone users, it is important to understand how these applications (mis)use Crypto API. Unfortunately, a recent study showed that 88% of Android applications made at least one mistake while consuming Crypto API [57]. Further, it is unclear if these misuse rates have changed since the mentioned study took place. In addition, the report [57] had limited analysis of libraries and did not look into the security implications of the misuse cases.

To bridge this knowledge gap we designed and developed the BinSight system - a system that uses static analysis and program slicing in order to identify Crypto API misuse cases in Android applications. We analyzed 132K Android applications in total, which originated from three datasets collected in 2012, 2015, and

[2]In iOS disk encryption is always enabled, while in Android it is an option a user can enable or disable.

2016. The dataset from 2012 was given to us by the authors of the CryptoLint study [57], which allowed us to replicate the original study and assess the ratio of over-counted bugs in the CryptoLint report.

Our analysis results revealed that 9 out of 10 calls to Crypto API originated from third-party libraries and that the original CryptoLint study had missed 249 (or 96%) libraries in their dataset. Further, the results showed that 222 of the missed libraries were responsible for 70% of the flagged Android applications. This strongly suggests that source attribution is crucial for the accuracy of Crypto API misuse analysis. Comparison of the misuse rates between applications collected in 2012 and 2016 showed that while applications and libraries have improved in certain aspects of Crypto API use, they worsened in others. In particular, while libraries have significantly reduced the use of ECB mode for symmetric ciphers, libraries significantly increased their reliance on static IVs and static encryption keys. In addition, the RC4 cipher, a cipher with a known vulnerability, gained popularity in 2016 and became the third most commonly used cipher.

Analysis of the applications collected in 2016 revealed that 89.5% of the flagged applications had Crypto API misuse cases in only third-party libraries. In other words, 507 libraries were responsible for introducing Crypto API misuses to 79,207 (out of 88,510) Android applications. Unfortunately, such dominance makes the current approach of measuring misuse rates of Crypto API highly biased towards libraries. The root cause of the bias is that libraries, especially the popular ones, inflate the ratio of APK files that misuse Crypto API. To address this limitation, we proposed to use the ratio of call-sites with and without mistakes for each source type. This allowed us to identify cases when the original metric, i.e., the ratio of Android applications with misuses, conveyed a misleading message. In particular, according to the ratio of Android applications, applications themselves have significantly improved in terms of not using static encryption keys. At the same time, the call-sites ratio suggests the opposite, i.e., the code of applications worsened.

Finally, the results of manual in-depth analysis of the top libraries revealed that a misuse of Crypto API does not necessarily imply security vulnerability. In particular, one might use Crypto API for reasons other than confidentiality or integrity protection. As we show, the Google Play SDK library used a symmetric cipher to obfuscate code. In addition, we found an edge case to the use of ECB mode, when a single block of random data was encrypted.

The present study makes the following contributions: (1) replication of previously published research through obtaining the original data from the CryptoLint study, (2) comparative analysis of Crypto API misuse in Android applications between 2012 and 2016, (3) improvements to the analysis framework by introducing source attribution and de-duplications, (4) and analysis of security implications of misuses in the top libraries.

In comparison with existing tools, e.g., Soot [32], neither CryptoLint [57] nor BinSight introduce anything novel to the field of static analysis itself. They are both highly specialized tools tailored towards the analysis of specific issues in Android applications. Nevertheless, while CryptoLint failed to analyze 23% of the APK files, BinSight was able to analyze all but six of APK files out of a ten times larger dataset. Furthermore, the results of the analysis demonstrated that the previously used method of measuring misuse rates is biased towards libraries. To address this issue, we proposed to use the ratio of call-sites with mistakes to all call-sites, which provides intuition into how probable it is that a call to Crypto API from an application itself or a library would make a mistake.

Overall, when it comes to protecting user data in smartphones, both application and library developers are doing a poor job. In particular, 50% of all calls to symmetric cipher API end up using a static, i.e., hard-coded, encryption key. This suggests that the encryption key management system is currently the weakest link, since (a) smartphone users tend to choose easy-to-guess unlocking secrets and (b) application and library developers rely on static encryption keys.

1.2.3 Storing Encryption Keys on Wearable Devices

The results from both user studies and the analysis study on the rates of Crypto API misuse in Android applications revealed that the encryption key is not effectively protected. On one hand, we see that users tend to choose easy-to-guess unlocking secrets, making password-guessing attacks trivial. On the other hand, application and library developers often use static encryption keys, which can be extracted from binaries with any of the existing reverse engineering tools. To address this limitation we designed and evaluated a system we named Sidekick. The Sidekick system uses wearable devices to store encryption keys, eliminating the dependency of data encryption security in smartphones on the entropy of unlocking secrets, and providing a secure location for developers to store encryption keys.

The evaluation of the Sidekick system revealed that our proposal is both effective and efficient. In particular, the system it allows fetching a 256-bit long encryption key from the wearable device in under a second. This is a significant improvement over a commonly chosen 4-digit PIN-code, which, in comparison, provides around 13-bits of entropy (assuming that a PIN is selected randomly). Sidekick's power consumption impact on smartphones was below 1% of battery capacity and allowed the wearable device to function for up to 400 days on a single coin-cell battery.

The contributions of this study include: (a) the design and evaluation of a system that decouples user authentication and data encryption in smartphones, (b) recommendations on the value for configuration parameters for the wireless communication stack, and (c) making the system available as open source and incorporating key parts of it in a real product. We envision the Sidekick system's inclusion in features provided by existing wearable devices, such as smart watches or fitness trackers, so that users would not need to use yet another device. Considering that the proposed system needs 20Kb of ROM and 4Kb of RAM on a wearable device and can run on an 8-bit CPU, such an integration should be simple.

1.3 Contributions Summary

To summarize the previously stated, this thesis makes the following contributions to research:

First, we **studied user experiences with data protection in smartphones**. We show that half of smartphone users do not use locking systems due to various usability issues or security concerns. We also show that the majority of the users that lock their smartphones choose easy-to-guess unlocking secrets, which makes data decryption a simple exercise for an attacker. Finally, our results suggest that the assumptions of recent research about certain environments are highly questionable. In particular, while several authors have suggested that work and home are safe environments, we showed that these locations commonly experience *insiders*. The studies presented in this dissertation provide evidence that users experience attacks by insiders in real life. These results imply that the security of data-at-rest in smarpthones needs to be re-evaluated without assuming that users will choose a hard-to-guess unlocking secret. Specifically, if developers of the data encryption layer use unlocking secret to generate or protect their master key (the actual encryption key used for data protection) they should not use unlocking secret as a single source of randomness and should use other sources as well.

Second, we **replicated previous study on Crypto API (mis)use rates in Android applications**. By obtaining the set of applications used in the previously published research, we were able to replicate the original study and confirm its findings. While doing so, we identified certain limitations in the methodology. In particular, we showed that the authors of the CryptoLint study missed 96% of the libraries in their dataset, which resulted in a misleading message. That is, while they reported that 88% of the applications misused Crypto API, 70% of them were due to 222 libraries.

Third, we **conducted analysis of Crypto API (mis)use rates in Android applications.** By collecting new applications in 2016 we were able to compare how the misuse rates have changed since 2012. In particular, we showed that while applications and libraries have improved in certain areas, e.g., the use of

ECB mode for symmetric ciphers, they have become worse in others, e.g., the use of static encryption keys.

Analysis of Crypto API misuse revealed that while applications developers improved in certain aspects (e.g., random number generation), they made more mistakes when it comes to the use of symmetric ciphers. Specifically, the use of static encryption keys and initialization vectors has increased between 2012 and 2016. Even more, the popularity of long time considered insecure (e.g., DES or RC4) ciphers has also increased. Combination of with users preferences on unlocking secret choices with frequency of Crypto API misuse by applications developers suggest that both parties fail to secure sensitive data in smartphones, thus further research is needed into various aspects of the problem. First, usable security research community should investigate if it is possible to have a usable, yet secure authentication method on smartphones, such that be used as a source of entropy for encryption key derivation function.

Finally, we **evaluated technical feasibility of encryption key management system based on wearable devices for smartphones.** We designed, implemented and evaluated the technical aspects of using wearable devices for encryption keys management in smartphones. The results of the lab experiment revealed how one can approach the trade-off between data access latency, session key refreshing schedule and power consumption on both wearable devices and smartphones. Such an approach can be used to address both users preference of easy-to-guess unlocking secrets and misuse of Crypto API. In particular, by acting as an additional source of entropy for key derivation process, the proposed system can eliminate the sole dependency of security of master encryption key on security of unlocking secret. This can be achieved by encrypting the master encryption key with a randomly generated key encryption key (KEK), which is then stored on a wearable device. Second, by integrating such system with existing file system (e.g., EXT4) one can simplify data encryption for application developers. In such design, developers would declare a file as encrypted and use the proposed encryption keys management system to manage encryption keys.

Chapter 2

Smartphone Users Experience with Data Protection

This chapter presents the results of two user studies conducted to gain a better understanding of how users perceive threats associated with disclosure of data-at-rest stored on the smartphones they own. This chapter begins with a discussion of research questions. It then proceeds to the design and results of the first user study, which was qualitative in nature based on the results of semi-structured interviews. We then present the quantitative results of the second user study, which was based on the results of online surveys. The chapter concludes with overview of related work and a summary of three follow up studies and a discussion of results and conclusions.

2.1 Research Questions

By studying smartphone users one can get a deeper understanding of how users perceive smartphone threats, such as loss and theft, and risks associated with theft and loss, e.g., confidential and sensitive information disclosure or reputation damages. The following research questions were defined, in order to fill this knowledge gap:

- **RQ1** - *What types of sensitive data do users store on their smartphones?*
- **RQ2** - *What practices do users employ and do not employ for data confidentiality protection?*
- **RQ3** - *Why do users choose to use (or choose not to use) certain security practices?*
- **RQ4** - *How concerned are users with unauthorized access of their data or smartphone functionality by an attacker?*
- **RQ5** - *How many users have experienced unauthorized access to sensitive data?*

Throughout this study we define data as being *sensitive* if a user wants it to be available to a limited number of persons, including just to him/herself. For instance, while users might want to keep photos of their children accessible to family members, they might want to limit access to personal messages or browsing history to themselves.

Answering **RQ1** provides a better understanding of the variety of sensitive data that users store on their smartphones. Knowing the types of data we need to protect makes it easier for the research community to design an effective and efficient data protection system. Answering **RQ2** and **RQ3** would improve our understanding about which security practices in smartphones users employ or do not employ and, most importantly, why they choose to do so. Having a better understanding on these two important research questions would give us and the wider research community a clearer picture of everyday issues that users face when they try to protect their smartphone data.

Finally, answering **RQ4** and **RQ5** provides a better understanding of users' perception and previous experiences with theft and loss threats. In particular, **RQ4** provides a deeper insight into whether or not users consider the risk of confidential data disclosure important.

Answering **RQ5** provides empirical evidence to the question of whether or not users have experienced theft and loss of their smartphones, which results in confidential data disclosure. Even more, **RQ5** is looked at from two perspectives. First, subjects were treated as victims, i.e., asked if someone else had unauthorized access to their smartphones. Second, subjects were treated as attackers and were asked if they had accessed someone else's smartphone without permission.

2.2 Approach

To answer the research questions defined in the previous section, two user studies were conducted. First, a qualitative study based on a set of semi-structured interviews was conducted ("Study 1" throughout the rest of this chapter). Second, in order to corroborate the results of Study 1 and to gain statistical power, a quantitative study was administered in the form of an online survey ("Study 2" throughout the rest of this chapter).

2.3 Study 1 – Interviews

This section presents the methodology and the results of the qualitative study based on interviews with 22 subjects.

2.3.1 Methodology

In the exploratory study we used semi-structured interviews for data collection. The main objective of this study was to gather qualitative answers, rather than quantitative. The decision to begin with a qualitative study is based on the fact that qualitative studies give a better opportunity to explore the problem domain without restricting ourselves. That is, by starting with a qualitative study, we can conduct a well-informed quantitative study. One crucial advantage of semi-structured interviews is that an interviewer can easily deviate from the initial interview structure, which allows researchers to dig deeper into unanticipated topics that emerge during the interviews.

We used theoretical sampling [67] rather than random sampling during the selection process of participants. We made this choice, because it was more important for us to recruit a diverse pool of subjects, rather than having a representative sample of general population. Diversity is often more important during qualitative studies, especially when questions on variability of some parameters need to be answered. Accordingly, before scheduling an interview with a candidate we asked each of them to fill out a pre-interview questionnaire. Once we obtained their answers, we checked if demographic parameters added to the diversity of our subject pool. In the questionnaire, we asked seven questions about age, gender, completed education, job position(s) and area of work, hobbies, annual household income, and native language. The list of questions is provided in Section A.1. Each interviewed participant was paid $25 CAD for a one-hour long interview. We applied and received approval from UBC Behavioural Research Ethics Board to conduct this study (application H11-02230).

All interviews began with a set of simple questions, such as *"what applications did you use during the last few days?"* or *"what was the first thing you did with your smartphone today?"* Subjects were then asked about the applications they were using on their smartphones, while interviewers recorded the names of all mentioned applications. For the applications that could have been used for business or work, such as calendar, emails, and messengers, subjects were also asked to clarify if they used these applications for work or business related activities. Afterwards, each subject was asked about how he or she used these applications and what kinds of data each application stored.

In what followed, subjects were asked if there was a user account associated with each application and, if so, if they need to authenticate every time they launch the application. This was asked in order to uncover applications where users chose to save their credentials, allowing application launching without re-entering login information. For example, if a participant used an email client, we asked her about the kinds of emails she would usually receive to the accounts she registered in that application. We also asked whether she saved passwords from any of the used email accounts or typed them in each time she needed to access her emails. In most cases, to validate responses from subjects, we asked them to launch the application in front of us.

During a pilot study, we found that it is difficult for participants to provide a clear answer about the sensitivity and the value of their data without a scenario. To address this issue, we gave several scenarios to participants, aimed to communicate probable risks more clearly. For the sensitivity of each data type we asked participants about the consequences of disclosing the information to a stranger or an insider. We explained these two terms to subjects as follows - a *stranger* is someone you do not know and he/she does not know you (e.g., a thief on a bus), while an *insider* is someone from your social circle or someone who knows you (e.g., a coworker).

Finally, by the end of the interview, participants were asked about the practices they used to ensure that their sensitive data are kept confidential. Subjects were also asked why they did not use certain tools and features, e.g., unlocking secrets or regular data backup.

All interviews were conducted by two interviewers in order to ensure that all important questions were asked. Interviews were audio-recorded and transcribed verbatim. Later, the transcriptions of the interviews were coded, analyzed and checked by both interviewers. To ensure sufficient numbers of participants in our study we used information saturation analysis. That is, after each interview we categorized all additional unique pieces of information that the interview revealed. Once information saturation was observed, the recruitment of new subjects was stopped.

2.3.2 Results

Demographics of Recruited Subjects

In total, 22 interviews were conducted during October, 2011. Half of them were conducted at the University of British Columbia (UBC) Point Grey campus, and

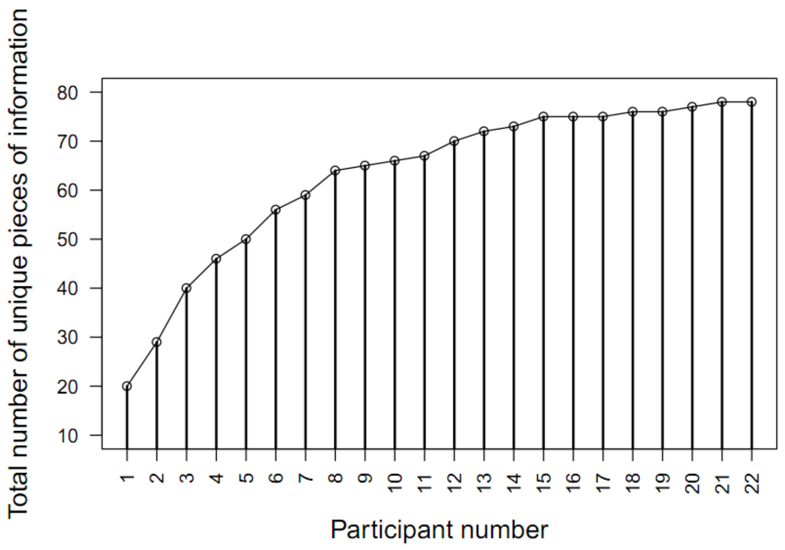

Figure 2.1: CDF of new collected information across the interviewed participants.

the rest at UBC's Robson Square campus in Vancouver. As shown in Table 2.1 the demographics of the recruited participants were diverse and included subjects from various occupations and age groups. Note, that some of the participants had more than one job, resulting in the total number of subjects not equaling the number of participants per occupation.

After the 18^{th} participant we observed that adding additional participants did not reveal new information. In accordance with the theoretical sampling approach, the recruitment of new subjects was stopped. The graph, shown in Figure 2.1, supports this decision and shows that saturation in data collection was reached.

RQ1 - Types of Data Stored on Smartphones

While analyzing the types of data users mentioned during the interviews, we observed that users refer to data from two different angles. First, they explicitly define that certain types of data are sensitive for them. Second, they defined some types of data as being valuable. The *sensitive* data included records that users wanted to keep to themselves, e.g., personal messages, while valuable data

Table 2.1: Demographics of 22 Interview Participants in Study 1.

Parameter	**Property**	**Participants**
Gender	Males	10
	Females	12
Age	under 18	1
	19-24	7
	25-30	5
	31-35	2
	36-40	3
	41-45	3
	46-50	1
Education	Still in High School	1
	High School	6
	Professional School or College Degree	4
	University (Bachelor's)	6
	Graduate School (Master's or PhD)	5
Household income	under 15K	6
	15K-30K	3
	30K-50K	3
	50K-80K	7
	more than 80K	3
Smartphone OS	iOS	11
	Android	4
	Symbian	2
	BlackBerry OS	4
	WebOS	1
Occupation	1 Caregiver, 1 Curator Assistant, 1 Entrepreneur, 2 Graduate Students, 1 High-school Student, 1 Language Teacher, 2 Marketing Specialists, 2 Municipal Workers, 1Network Administrator, 1 Nurse, 1 Librarian, 1 Pilot Instructor, 1 Proof-reader, 4 Sales Workers, 1 Security Guard, 1 Software Engineer, 1 Tailor, 1 Undergraduate Student, and 1 Unemployed	
Data Stored	Work Related	9
	Personal	22
Phone Ownership	Personal	19
	Company	3

Table 2.2: Types of Data and their Sensitivity and Value from Users' Perspectives.

Data Type	**Sensitive**	**Valuable**
SMS Messages	◒	○
Photos/Videos	◒	◒
Voice Recordings	◒	●
Notes	○	◒
Contacts	◒	◒
Music	○	○
Passwords	●	◒
Emails	◒	○
Documents	◒	○
Events in Calendar	○	○
Recorded GPS Tracks	●	○

included data that users were worried about loosing, e.g., memorable photos or videos. Considering users' needs, sensitive data require confidentiality protection, i.e., should be only accessible by the owners, while valuable data require availability protection, i.e., being available to the user, even if the device is lost or stolen. If, however, a data record is both sensitive and valuable, one should carefully design the availability of the system in order to avoid compromising confidentiality. Considering that most systems for availability protection rely on some sort of cloud storage, this creates another attack vector on users' sensitive data, and could have a devastating impact on users themselves, e.g., recent iCloud hacks that exposed personal images of celebrities [1].

A summary of data types and their sensitivity and value is provided in Table 2.2. Based on subjects' feedback, we mark sensitivity and value as none, partial or full. In Table 2.2 a fully filled circle in sensitive and valuable columns

[1] Due to the vulnerability of the iCloud's web interface, attackers were able to mount a password guessing attack on celebrity accounts and eventually gained access to the backup of their photos from their smartphoneshttp://www.bbc.com/news/entertainment-arts-39280844

means that most of the subjects agreed that the data type is sensitive or valuable. On the other hand, an empty circle means that a data type was not considered sensitive or valuable by any participant. Data types that were considered sensitive only by a minority, i.e. at least one participant, are shown as half filled circles.

Again, considering that these results were obtained in a qualitative study, one should treat them with caution. This is why I refrain from reporting any statistics from our observations, aside from the diversity of opinions. In fact, I do not provide any descriptive statistics about the data types or their classes as this was not the goal of Study 1. To provide such descriptive statistics one should use a different, quantitative approach. The following discusses the reasons subjects used to justify the sensitivity or value of data types.

Passwords: Some of our participants stored passwords on their smartphones using different means. One participant stored passwords for online banking as contact records in their address book. Another created notes for door PIN-codes for her workplace. A sub-group of the participants used special applications, such as password managers. Most participants opted to allow applications to save the associated credentials, so that they did not have to enter a password every time they opened the application (e.g., email clients, Facebook application, etc). All participants considered these passwords to be sensitive. It should be noted that the participants who used password managers were less worried, since such applications usually required an additional password. However, they did admit that the password they used was the name of a person or a simple word. Interestingly, some of the participants considered password lists highly valuable, and these lists were stored only on their smartphones where loss of the list would incur a significant amount of work to restore access to the corresponding accounts.

Music and Events in the Calendar: Music and events, on the other hand, were never mentioned as being sensitive or valuable. Most of the participants justified such judgment by the fact that they had a copy of such data on their computers, online, or that they could remember the information. In the case of losing appointment information or events, subjects reported that they also had a physical agenda book to keep the information.

Voice Recordings: Several subjects used their smartphones to record audio of conversations, which had the potential for being confidential and sensitive. For example, one of the subjects was a quality assurance specialist, and had recorded multiple conversations with employees that provided anonymous feedback on the internal affairs of the company. On the other end of the spectrum, subjects also used voice recordings for taking notes and memos. These types of recordings were not considered as sensitive. Most of the subjects that used voice recorders for note taking did consider these recordings as valuable, mainly because subjects were not sure if they would be able to recover them if lost.

Photos and Videos: Some of the participants defined photos and videos on their smartphone as both sensitive and valuable. Others considered their photos and videos sensitive for cultural reasons. For example, one of the participants stated that photos of his family were sensitive, as women in his culture wear a Hijab in public. Interestingly, most of the participants who took pictures and videos on their smartphones kept them there for some period of time in order to accumulate a considerable amount before transferring them to a PC. Several participants who recently lost or damaged their phones admitted that they had lost valuable pictures as well. Moreover, it was hard for participants to recall on the spot whether they had valuable or sensitive pictures, without first looking through their images and videos.

SMS Messages: The analysis of the interviews revealed that SMS messages have a short temporal value, which is lost once they have been read. Most of the participants stated that they do not use SMS for highly meaningful conversations, and rather use SMS messages for friendly chats and as a way to keep in touch with their friends. Certain subjects did, however, consider specific types of messages as being sensitive, but only if they were read by certain people, e.g., their parents *"I do not like the idea of someone, especially my parents, going through my messages..."*. At the same time, these subjects were comfortable sharing these SMS messages with their friends.

Contacts: The participants sometimes considered contact details as being sensitive, mostly because they were not comfortable sharing such data with others. Reasons varied from reputation consequences *"If someone got hold on of my contacts, I would feel uncomfortable, because I feel like those people trusted me to keep their phone numbers private"* to expected threats to people *"I am not sure what those who got my contacts numbers will do with them, they could call them or send them spam"*. The value of contact details was justified mainly by the lack of synchronization with a PC or an online account. Interestingly, some of the participants stated that they had a copy of their contact details in paper form, which they carried around with them as they had lost their smartphone previously or experienced other technical problems, such as their batteries dying.

Email Messages: Participants classified their emails as being not valuable, because all of them were able to access emails either online or on their personal computers. A majority of the participants had multiple email accounts configured on their smartphones. They classified their emails as "junk-collecting" or "sign-up" email, personal and work related. Nine of the participants used work email accounts on their phones and received confidential business emails which contained unreleased products details, marketing company budgets, business proposals, etc. The mix of unimportant and important emails defined email sensitivity as "could be sensitive".

Documents: Some participants uploaded work-related documents to their smartphones. These documents often contained confidential information, such as de-

Table 2.3: Security Practices and Experience of Interviewed Participants

Parameter	**Property**	**Participants**
Use pin-lock	Yes	7
	No	11
	No, but used to	4
Had experience of	Losing phone	5
	Breaking phone	4
	Losing and Breaking phone	1

scription of new products or sales figures. The necessity of having such data on their smartphones was justified by the need to access vital numbers during travels. These documents were not considered valuable, since they were also stored on the company's servers or work-related computers.

GPS Tracks: GPS tracks are usually recorded by training assistant applications, such as miCoach for iOS. Subjects used such applications to track their outdoor exercises and monitor performance. The tracking information that these applications collected was considered to be highly sensitive, mainly since most of these tracks lead to the subjects' homes.

RQ2 – RQ4 - Security Practices for Data Protection and Concerns

Subjects were asked whether or not they used any tools for data protection. In addition, the interview structure also included questions that focused on the reasons for and against using such tools. The results of the interviews on such practices are presented as follows. A short summary of the results is provided in Table 2.3.

Most of the participants, but not all of them, backed up valuable data whenever they "felt" that they needed to, which varied from *once a week* to *once in six months*. Those who lost their devices and valuable data, however, admitted that they started paying greater attention to this practice after the loss. The subjects often cited the following reasons for such infrequent and irregular backups (1) inconvenience of current systems, (2) lack of time, and (3) lack of information on what data needs to be backed up.

Several participants stated that they do not trust the security of their smartphones at all, and, thus, decided not to store any sensitive or valuable data on such devices. Their decisions were based on concerns they had with the security of smartphones, especially if they were lost, stolen or infected by malware. Interestingly, 20 participants stated that they considered smartphones to be less secure than PCs. The high mobility of smartphones was cited as the main justification, since this meant an increase in chance that the device could be lost or stolen.

The participants were then asked about what they would do if they had just lost their smartphone. A majority of the subjects told us that their first action would be to try to recover their device, by going through places they visited in the last few hours. Four participants said that they use special applications to track their devices, such as “Find my iPhone” [3] and would try to locate their smartphones through this approach first. The answers of those who had lost their mobile phones before did not show any differences.

In the case that subjects could not recover their smartphone within a couple of hours, all participants told us that they would call their service providers and block their line to avoid paying for someone else using their high cost services. All participants who stored phone passwords in any form said that they would change their passwords in a day or two after the loss. Not surprisingly, the phone itself was also mentioned as a financially valuable asset to lose.

In the scenario when their smartphone had been stolen or used by someone else, participants showed a different perception of risks, depending on who the attacker was. Threat expectations were higher for 17 participants when the attacker was someone who knows them. The participants also stated that when lending their smartphone to their friends, they would like to keep an eye on them because they had concerns about this person looking through their personal data, such as messages and pictures. Most of the time they did not care about showing some data, such as messages and emails, to complete strangers, but did care if such data were seen by someone within their social circle.

Not surprisingly, 21 participants stated that they would like to store backups of sensitive data at home on their PCs or external hard drives, rather than having them online. Two Android users decided to disable synchronization with their Gmail account completely because of privacy concerns. Moreover, half of the participants only used external hard drives as a backup solution for their home media files, such as videos, pictures and documents. Although most of them did use some form of “cloud” storage, such as Gmail, Facebook or Dropbox, they preferred to store only “shareable” content with these services. This is, also, consistent with the findings of Ion et al. [74], where they studied users’ attitudes towards adoption of cloud storage in general.

Out of 22 subjects, only seven participants used PIN-locks on their smartphones. Out of these seven, one subject used it only because of a company work policy, and told us that he would not use it otherwise. Another participant said that she only used PIN-lock to protect her SMS messages from her parents, and found it annoying that she was not able to isolate and protect these messages. All participants that used PIN-codes stated that they typed PIN-codes very often for data that is both not sensitive and not valuable to them, such as weather forecasts or games.

Those who previously used device locking with a PIN-code, but who switched it off at some point, justified this decision by needing quick access to specific data and functions of the device on the go or in specific circumstances. For instance, one of the participants said that she gave up on using PIN-codes to lock her device when she was at a party and needed constant access to the Internet to check certain information. She found it highly inconvenient to type her PIN-code each time, so she decided to switch off smartphone locking completely. Moreover, needing to type PIN-codes or passwords in on-the-go situations rendered some users’ devices unusable, especially when users were in a rush.

Similarly, users who did not use any type of PIN-code agreed that typing a

PIN-code for every application on their phone *"does not make any sense"*. For those who did not use such locks, the main reasons were (1) subjects did not have any sensitive data on their smartphones, (2) it was too inconvenient for them to type their PIN code or password, or (3) they felt "socially-awkward" to type a password in front of their friends or family members.

RQ5 - Experience with Unauthorized Access

Two subjects stated that they had been victims of unauthorized access to their smartphones in the past. One subject said that she had to lock her smartphone at home because her brother and parents tried to access her photos and messages *"I am enabling smartphone lock once I am home, to prevent my brother and my parents going through all my pictures and SMS messages."* Another subject, a female student who shared an apartment with other students, said that while she was asleep, her roommates used her phone without her permission.

One subject admitted that she looked through all messages and pictures on a phone that she found in a cinema theater *"I first called to the last number in call history [...] then I had to wait for them, so I decided to peek into photos and messages, just out of curiosity. Would you not do the same?"*.

2.3.3 Summary

The results of the qualitative study presented in this section provide a better understanding of users' perspective on threats to smartphones, especially those that are relevant to confidential data disclosure. In addition, the results of the study sheds light on (1) the data types that smartphone users store on their devices, (2) participants' opinions on data sensitivity, and (3) security practices users employ or do not employ and why.

Overall, the results of the qualitative study suggest that users store significant amounts of sensitive data on their smartphones and that they are concerned with the disclosure of such data if the device is lost or stolen. The majority of them, however, tend to not take any actions in order to ensure confidentiality protection. In particular, only a few subjects used secure passwords, while the rest used 4-digit PIN-codes. Even more, the PIN-codes were considered unusable by a majority of the subjects and were often avoided completely. Subjects justified such behavior by the need to have instant access to their data and applications, which often appeared to be not sensitive (e.g., games, weather forecast applications, and internet browsers).

Finally, this study provides evidence that users indeed experience unauthorized access to their devices and data. It is clear that these attacks are mounted by both strangers and insiders. In particular, several subjects admitted that they had been victims of such attacks, where one of them took proactive actions to defend herself, by locking her phone at home (i.e., an insider attacker). Another subject admitted that she accessed photos and messages on a phone she found in a cinema theater (a stranger attacker).

2.4 Study 2 – Online Survey

Although Study 1 provided us with rich qualitative data, it did not allow us to quantify different opinions, and hence, did not allow us to answer such quantitative questions as *"How many users use PIN-codes for device locking?"* or *"How many users encountered unauthorized access to their devices?"* To address this knowledge gap, a quantitative study, based on online surveys, was performed.

2.4.1 Methodology

The design of Study 2 is based on the results of Study 1. In particular, the questions and structure of the online survey was defined around the identified data types in Study 1. In addition, subjects were questioned about their perception of threats and risks with two different types of adversaries in mind, i.e., a *stranger* and an *insider*. In Study 2, we used an online survey, which allowed us to recruit a larger and more diverse participant pool and measure statistical prevalence of various opinions, practices and experiences. To ensure clarity of the questions and correctness of the data collection process, we conducted four pilot studies (between January and May 2012) with 60 subjects in total. Data from pilot studies are not included in the analysis presented in this chapter.

The online survey consisted of four parts. In the first part, general questions were asked about the use of smartphones. In particular, participants were asked when they locked their smartphones and if they also used a code (either PIN, Draw-a-Secret, or a password) to unlock it[2]. Next, the subjects were asked to visit a web page on their smartphones. Our data collection tool used this opportunity to record a *UserAgent*[3] string from their smartphone. This allowed us to eliminate subjects that did not provide evidence that they owned a smartphone.

The second part of the survey included questions about respondents' previous experience with their smartphones, e.g., loss or damage. Subjects were also questioned if they had previously accessed someone's smartphone without permission, and if someone had accessed their smartphone without their permission.

The third part of the survey contained questions about data types that subjects stored on their smartphones. Participants were provided with a pre-populated list of data types (compiled based on the results of Study 1) and were asked to select those that they stored. An option to add a new type was also available. These questions were asked twice, once for personal data and once for work-related data.

In the final part of the survey, subjects were required to rate their agreement with the following statement, "I would not have any concerns if *Personal/Work Data Type* could be viewed by such a thief" on a 5-point Likert scale for each data type. The following options were provided: Strongly Agree, Agree, Neutral, Disagree, Strongly Disagree. The rating task was performed twice, once for a stranger scenario and once for an insider scenario. The stranger scenario was

[2]Face and finger print recognition were not available at the time of this study.

[3]A *UserAgent* is a string that every browser sends to the web server. For instance, the following string is sent from an HTC Sensation 4G that runs Android 2.3.4 - *"Mozilla/5.0 (Linux; U; Android 2.3.4; en-us; HTC Sensation 4G Build/GRJ22) AppleWebKit/533.1 (KHTML, like Gecko) Version/4.0 Mobile Safari/533.1"*

presented as, “Assume your smartphone was just stolen by a person who does not know you [sic],” and the insider scenario as, “Assume your smartphone was just stolen by a person who knows you [sic].”

Finally, subjects ranked the importance of each data type they stored on their smartphones. Ranking was performed twice, once for the stranger scenario and once for the insider scenario. In each ranking task, subjects were asked to rank the data types by their level of concern, with most concerned at the top and least concerned at the bottom. A drag and drop user interface was provided for this task.

We instrumented a custom survey website with tools that allowed us to track the following: how much time each subject spent on each question; IP addresses of the PC and smartphone used for the survey; and UserAgent strings for the PC and smartphone. Later, these data were used to remove subjects that either skimmed through the survey (23 subjects), or did not use a smartphone (942 subjects). The UserAgent string was also used to measure the representativeness of our subjects in terms of mobile platforms and OS versions.

In our data analysis, we used the Fisher Exact Test (FET) or Chi-Squared Test (CHI) for tests on contingency tables. To analyze the differences between sensitivity rates for strangers and insiders, we used the U-test (Wilcoxon rank sum test). To analyze the differences between sensitivity ranks for strangers and insiders, we used the Wilcoxon signed-rank test (WSRT).

Study 2 was conducted between May 16 and June 23, 2012. The survey was available in the US, UK, Australia, New Zealand, and Canada on Amazon’s Mechanical Turk (MTurk); through other advertisement services, such as Kijiji and Craigslist; and through *“word of mouth”* approach. The study was approved by the UBC Behavioural Research Ethics Board (application H11-03512).

2.4.2 Results

In what follows we report the results of Study 2.

Demographics of Recruited Subjects

2,092 subjects were recruited for the online survey. 1,725 respondents successfully completed the survey, i.e., answered all required questions. Further investigation revealed that only 783 of the subjects used their smartphones as required in the smartphone ownership testing task. As was identified in the preliminary pilot studies, the survey required at least 10 minutes to finish. This is why 23 participants, who finished the study in less than 10 minutes, were excluded from the analyzed set. Finally, considering that the MTurk platform was the most successful recruitment tool, we decided to remove subjects recruited through other means, as to avoid having a user study that was difficult to re-produce.

The remaining 724 participants completed the survey in 25 minutes on average (std. dev., s=12.5). The majority of the participants were from the US (634); the rest were divided between Canada (50), the UK (29), Australia (9), and New Zealand (2). The majority of subjects used Android OS (391/51%) and iOS (278/37%). We did not find a statistically significant difference for our sample platform distributions and the distributions reported by Google and Kunzler [8, 68] (FET,$p>0.08$). Three hundred seventy of the subjects were male (51%) and 354 of the subjects were females. The average age for the subjects was 25.6 years (s=5.98). The average annual income was \$43k (s=\$19k).

The list of occupations reported by the participants was diverse and included more than 500 different titles in 16 various industry fields, such as agriculture, business, construction, education, etc.

We compared the demographics of our subjects with the results reported by Smith [105]. To the best of our knowledge, Smith's study is the only study that provides statistics on a representative sample of the US population of smartphone users (n=2,253), and the majority of our subjects were from the US. For this part only, all non-US subjects were removed (90). For the rest of the analysis, all 724 subjects were used. The analysis of differences between our US subjects and the ones reported by Smith's study [105] did not reveal a statistically significant difference in gender distribution. However, there was a statistically significant difference in age, income, and education. In particular, our participants appeared to be younger (29.6, $\sigma = 9.69, \chi = 361.6676, df = 3, p < 0.001$). This, however, is not surprising, as it was previously shown that MTurk subjects tend to be younger [88]. Although the differences in education and income distributions were statistically significant (FET, $p < 0.001$), we consider them practically insignificant due to small relative values. The average income in Smith's study was higher by 6% ($46k, sd = $20k), and the difference in education levels revealed that our sample had 9% more subjects with a high school diploma and 9% fewer subjects with a college or higher degree.

Demographic data analysis suggests that the recruited subject pool is a diverse and a representative sample, at least for the US, with a slight bias towards younger smartphone users.

RQ1 - Types of Data Stored on Smartphones

Subjects found the list of options for data types that we provided sufficient, since only three of them added new types. The 15 most used data types are provided in Table 2.4. Note that each data type name has a corresponding code, e.g., photos and videos are coded as *phv*, which is later used in discussion and figures for brevity.

RQ2 and RQ3 - Security Practices for Data Protection

The results of the online survey confirmed findings from the interview-based study, which suggested that most of the subjects did not take appropriate actions to protect their data on smartphones. In particular, only roughly half of the survey participants (379, 52%) locked smartphones with a code. These subjects are referred to as the lock-using group (LOCK). The remaining (345) participants did not use a locking system. We refer to this group as OPEN.

Subjects justified the necessity to lock their device by needing to limit access to data or functionality. In particular, 64% (243) of subjects in the LOCK group did so to prevent unauthorized access to their data, while 73% (278) of them did

Data Type (Label)	%
1 - Photos and videos (phv)	94
2 - SMS/MMS messages (sms)	93
3 - Call history (cah)	90
4 - Emails (eml)	87
5 - Contacts details (cod)	87
6 - Music (mus)	81
7 - Browser search history (bsh)	74
8 - Browsing history (bwh)	73
9 - Events in calendar (evt)	73
10 - Notes and memos (n&m)	72
11 - Data in social networking applications (osn)	68
12 - Progress in games (gam)	68
13 - Documents (doc)	64
14 - Voice recordings (voc)	42
15 - Passwords saved in applications or passwords managers) (pwd)	37

Table 2.4: The 15 most used data types by the subjects. All cases include only data types for personal use, since no work-related data types made it to the top 15.

so to avoid unauthorized use of the smartphone's functionality.

Similar to the results of Study 1, the OPEN group included 155 subjects that kept sensitive data on their smartphone and had used a locking system before, but had stopped due to various usability problems (too many authentication prompts, necessity to authenticate even if non-sensitive data was accessed, etc.). The other 190 subjects in the OPEN group did not have any sensitive data on smartphones.

Interestingly, most of the subjects in the LOCK group used either a PIN-code (206) or a Draw-a-Secret (DAS) (168) authentication method, whereas only 52 used alpha-numeric passwords. Note that subjects were able to select multiple types of authentication methods if they owned several smartphones, thus $\sum n \neq 379$. The participants from Study 1 justified the choice of PIN or DAS with ease of use, in comparison to fully-fledged alphanumeric passwords. The distribution of subjects' justifications for using a smartphone lock is presented in Table 2.5, and the distribution of reasons for not using a smartphone lock is shown in Table 2.6.

Reason	**n**	**%**	**CI ($\alpha = 0.05$, $z_{\alpha/2} = 1.96$)**
I feel comfortable having such protection	334	88	± 3.18
I do not want other people to use my phone services without my permission	284	75	± 4.28
I do not want other people sneaking into my smartphone, when I do not see it	246	65	± 4.7
I have confidential and sensitive data on my smartphone(s)	167	44	± 4.89
My employer requires that	25	7	± 2.39
I do not want my smartphone to "pocket dial"	6	2	± 1.2
My smartphone got stolen and i did not have a lock on it.	1	≈ 0	-
I lose (and later recover) my cell phone a lot.	1	≈ 0	-
It is a default on my phone	1	≈ 0	-

Table 2.5: Distribution of reasons for using a locking system (N=379). Note that $N \neq \sum n$, because the participants were able to provide multiple reasons. CI stands for confidence interval, given the number of subjects that were able to answer that question.

To summarize, a majority of subjects in the LOCK group used smartphone locking to feel comfortable, to avoid others using their device or looking through their data, or because they stored sensitive data (in some cases as required by their employer). On the other hand, 58% of OPEN group subjects did not lock their device because they did not store any data that required protection. Interestingly, 46% of subjects previously had locked their smartphones with a code, but stopped doing so due to usability issues. These results are consistent with the findings from the interview-based study, where subjects voiced concern about lack of granularity in the current locking mechanisms of smartphones.

Reason	**n**	**%**	**CI ($\alpha = 0.05$, $z_{\alpha/2} = 1.96$)**
I do not have any data that I want to hide on my phone	200	58	± 4.86
I tried locks before and found them very inconvenient	159	46	± 4.9
I often need instant access to applications that do not store any sensitive data (e.g. weather forecast, news, games)	145	41	± 4.84
It is not worth for me to use smartphone lock, because the amount of data and applications that are sensitive are very small compared to those non-sensitive	114	33	± 4.62
I do not save my passwords in applications and type it every time I use an application that stores sensitive data (e.g. email application, facebook application)	66	19	± 3.84
I do not care if my phone services will be used by someone	55	16	± 3.6
It's always with me or in my sight	16	5	± 2.08
I did not have time to setup it (new phone) or I am lazy	5	1.38	± 1.15
I did not know about this feature	3	1	± 0.89
Other	6	2	± 1.25

Table 2.6: Distribution of reasons for not using a locking system (N=345). Note that $N \neq \sum n$, because the participants were able to provide multiple reasons. CI stands for confidence interval given the number of subjects that were able to answer this question.

RQ4 - Security Concerns with Sensitive Data

To answer RQ4 the differences between users' concerns with confidentiality of their data were analyzed. First, we analyzed Likert scale ratings with U-test, since the collected data were ordinal, and, thus, parametric tests (such as t-test, ANOVA) were not applicable. The results of U-test revealed that out of the 32 data types subjects rated their concerns differently for only six types. In particular, subjects showed highest concern with an insider threat for SMS messages, call history, browsing history, and search history in the browser. Additionally, subjects were more concerned about strangers for contact details and progress in games.

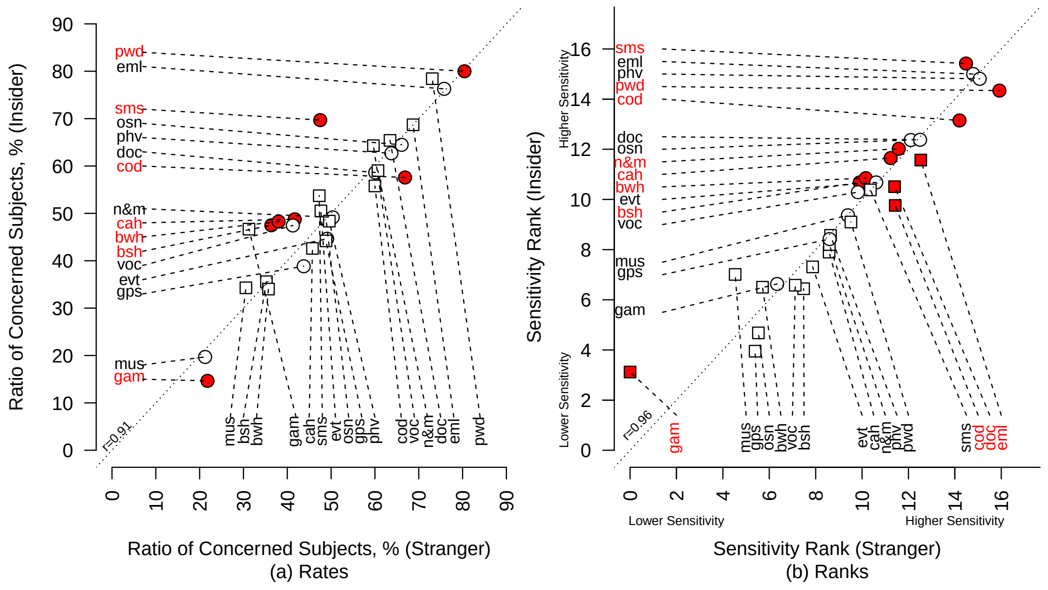

Figure 2.2: The proportion of concerned users with sensitivity in the presence of a stranger (horizontal axis) and in the presence of an insider (vertical axis). Data labels across the vertical axis and circles in the plots represent data types for personal use; data labels across the horizontal axis and squares in the plots represent data types for work related use. Filled shapes and red-colored data labels represent statistically significant differences between subjects' concerns with respect to a stranger and an insider (U-test for rates, WSRT for ranks, $p < 0.05$). The meanings for the abbreviated data type labels are in Table 2.4.

Figure 2.2a shows the proportions of subjects that were concerned with strangers (x axis) and insiders (y axis) for every data type. The proportion of concerned subjects for a data type was estimated as a fraction of the number of subjects that were either concerned or highly concerned with unauthorized access to the total number of subjects that stored such data. This plot shows that users' concerns with regards to both adversaries are highly correlated (r=0.91), which suggests that both types of adversaries are worth considering.

Statistical analysis of ranks for data types revealed 11 statistically significant differences (WSRT, $p < 0.05$). Most of the differences, however, had small absolute values, and could be ignored. For each of the data type we calculated an average value of the user provided rank and plotted results on Figure 2.2b. Simi-

Description of the experience	**n/%**
E1 - I have left my mobile phone at some place, but recovered it later (e.g., at my friends' place, in a restaurant, at parents' house, at school, etc.)	363/50
E2 - I have broken my mobile phone before, so that it was not usable	335/46
E3 - I have lost my mobile phone before and did not find it	165/23
E4 - Someone used my mobile phone without my permission with intention to use its functionality (phone call, browsing the Internet, etc.)	100/14
E5 - I used someone's mobile phone without owner's permission for some functions (phone call, browsing the Internet, etc.)	102/14
E6 - Someone used my mobile phone without my permission with intention to look at some of my data	89/12
E7 - I used someone's mobile phone without owner's permission to look into his/her data	66/9

Table 2.7: The distribution of "negative" experience of the participants ($N = 724$).

lar to the ratings, the correlation between ranks of user concerns for both types of attackers was high (r=0.96).

From these results, we can conclude that while users are concerned with unauthorized access to their data, these concerns are somewhat different for various data types. For example, users are more concerned with *insiders* gaining unauthorized access to their SMS messages, call history or browsing history. At the same time, users are more concerned with *strangers* if contact details are at stake.

RQ5 - Experience with Unauthorized Access To Smartphone

A summary of subjects' "negative" previous experiences is provided in Table 2.7. Half of the subjects had left their phones behind before. While such experience does not necessarily translate into theft or loss, it does make their device an easy target, since an attacker would have plenty of time to go through data. Almost a quarter of subjects (23%) that lost a device did not recover it.

Interestingly, 12% of the subjects had found that someone accessed sensitive data on their smartphones without their permission. Furthermore, 9% had admitted looking into someone else's smartphone without permission. These results provide empirical evidence that unauthorized access to data and functionality by insiders impacts about 10% of smartphone users. Subjects from Study 1 justified these invasions of privacy by simple curiosity (e.g., for partners or roommates) or by an urge to take care of and be informed about their children, i.e., parents "snooping".

We performed a logistic regression analysis in order to identify groups of smartphone users that had higher chances of being a victim of an authorized access. Logistic regression is best suited for models with binomial independent variables – in this case, those who have or do not have experience. In this analysis, we only analyzed the experience related to an unauthorized access (i.e., E4-E7). We built a model for each experience separately, four models in total. If a subject had such an experience, then we coded it as 1, otherwise 0. For independent variables, we considered the following values: *A -Age*, *G - Gender*, and *L - Lock Use*. For binomial independent variables (Gender, Lock Use), we used bipolar representation (-1,1). Equation 2.1 shows the form of the model we investigated, where E_x stands for one of the experiences from E4-E7.

$$E_x = \frac{e^{a_0+a_1G+a_2L+a_3A+a_4GL+a_5GA+a_6LA+a_7GLA}}{1+e^{a_0+a_1G+a_2L+a_3A+a_4GL+a_5GA+a_6LA+a_7GLA}} \tag{2.1}$$

The intuition of the model shown above is to assess if a given experience is correlated with subjects' attributes, such as age or gender [108]. The goal of logistic regression analysis is to eliminate attributes that do not have significant impact on an experience. In addition to the analysis of attributes, we also had to consider any interaction effect that might arise from a combination of variables, e.g., younger females or adult males who do not lock their device. These interaction effects are represented as GL, GA, LA and GLA variables in the equation above.

Logistic regression analysis revealed that, for all four models, all interaction effects were not statistically significant ($p > 0.174$), and thus could be removed from the model. Furthermore, Gender and Lock Use also showed statistically

Experience	a_0	a_1	p	**RD**	**AIC**	R^2
E4	-2.95	-0.53	< 0.001	546	550	0.09
E5	-2.90	-0.51	< 0.001	554	558	0.08
E6	-2.70	-0.36	< 0.001	521	525	0.05
E7	-3.13	-0.52	< 0.001	425	429	0.05

Table 2.8: Parameters of logistic regression models, where a_0 is intercept, a_1 is the coefficient in front of Age variable, p is the biggest p-value for both a_0 and a_1, RD is the residual deviance, AIC is Akaike Information Criterion, and R^2 is Nagelkerke R-squared.

insignificant prediction power on the experience ($p > 0.185$). That is why we simplified our models to the form shown in Equation 2.2. The parameters of the models are shown in Table 2.8.

$$E_x = \frac{e^{a_0+a_1A}}{1+e^{a_0+a_1A}} \tag{2.2}$$

First, the logistic regression analysis revealed that our models did not have strong predictive power since R^2 values were low. However, the coefficients of intercept and age showed a statistically significant difference from zero. Negative values of the intercept and the coefficient for age showed that the younger subjects had higher chances of experiencing unauthorized access. This is also depicted in Figure 2.3, where a larger ratio of younger subjects had experienced E4-E7. This might be attributed to various factors. For instance, younger smartphone users might tend to share their devices more frequently, or younger students often share accommodation with others while attending college or university.

2.4.3 Summary

The results of Study 2 confirmed the findings of Study 1. In particular, users were concerned with an unauthorized access to their devices for both data and functionality. Furthermore, the results provided evidence that threat of an insider attacker impacts about 10% of smartphone users. That is, 12% of smartphone

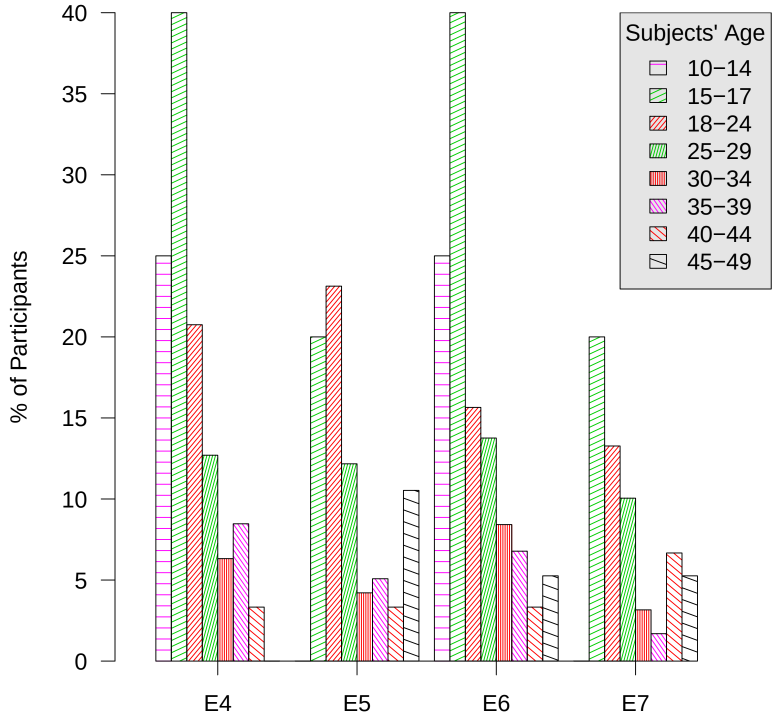

Figure 2.3: Distribution of the experiences E4-E7 (meaning for these labels are provided in table 2.7) over participants' age groups. We removed all the subjects that were younger than 10 and those that were 50 or older for clarity purposes.

users have experienced unauthorized access of their data or functionality, and 9% of the participants admitted that they had accessed someone else's smartphone without permission.

Study results suggest that most subjects (95%) who locked their smartphones (i.e., subjects in LOCK group, n=379), used PIN or DAS authentication methods. According to the recent research [55, 71], these methods are not resistant to eavesdropping, especially when users are distracted by other factors [48]. Even more, as was explained in Chapter 1, such unlocking secrets fall into *easy-to-guess* category. This is why we argue that the effectiveness of data protection systems against attackers that steal a victim's device is at least questionable and requires further research.

2.5 Limitations

The design of this study has several limitations. First, both Study 1 and Study 2 rely mainly on self-reporting, which is subjective, e.g., subjects might have not understood certain terminology. In order to reduce this risk to validity, we avoided security terminology and jargons in the questionnaire. In addition, a set of pilot studies for both Study 1 and Study 2 were carried out, in order to improve the clarity of the interview and survey questions.

Second, because the results of Study 2 are based solely on smartphone owners, the results in Table 2.7 should be treated as a lower bound. In addition, users might be reluctant to report on socially unacceptable behavior, such as snooping into someone's phone without permission. It is also possible that a user might not know if someone had accessed his or her smartphone without permission.

Finally, the participants of this study were recruited on the MTurk platform, which has been reported to differ [88] from the population of smartphone users [105]. Even though, alternative recruiting methods were used during subject recruitment, they unfortunately proved to be less effective. The comparison of demographics between recruited subjects and the previously reported population of smartphones users in the US [105] did reveal statistically significant differences. Most of them, however, were insignificant in practical terms.

2.6 Related Work

Several authors have investigated user concerns with the security and privacy of their smartphones. Chin et al. [46] conducted a user study to understand how concerned users are about their privacy when they use smartphone applications, especially for sensitive tasks like banking or shopping, etc. The authors found that users do tend to reduce the frequency of such activities because they are highly concerned with privacy. The participants attributed the cause of their behavior to fear; interestingly, theft and loss of a smartphone were among users' top five indicated fears. Unfortunately, none of these fears were investigated further. In particular, it is still unclear if these fears reflect real threats, and whether or not users had experienced such threats.

Dorflinger et al. [54] investigated users' attitudes towards gradual security levels and novel authentication methods. Although the authors provide a better understanding of user concern with novel authentication methods for smartphones, the question of how concerned users are with sensitive data in their smartphones remains unanswered. Even more, it is not clear what kinds of data users consider to be sensitive and if such sensitivity depends on who the attacker is. The studies presented in this chapter, on the other hand, fill this knowledge gap.

Similarly, Ben-Asher et al. [42] studied users' attitudes towards alternative authentication methods and the sensitivity of data and smartphone functionality. The authors, however, considered a limited set of data types, which only included seven different types of data. The authors also did not investigate how data sensitivity varied with different types of attackers, such as insiders or strangers. To improve on these results, subjects in both user studies presented in this chapter were allowed to provide their own data types. In addition, subjects assessed data sensitivity for two scenarios: (1) when data is leaked to a stranger, and (2) when data is leaked to an insider.

The research community has paid a lot of attention to evaluating novel authentication methods in recent years. For example, Shi et al. [102] and Riva et al. [94] evaluated implicit authentication methods for smartphones based on user behavior or context. Hayashi et al. [72] discusses possible extension to the whole approach of how we lock our mobile devices. In particular, the authors suggest that we could automatically detect specific environments and disable smartphone lock, if the environment is considered to be safe. In all the aforementioned papers, the authors made the following crucial assumption; that environments can be easily classified into *safe* environments, e.g., home or work, and *not safe*. By being able to detect *safe* environments, the authors were able to reduce the number of authentication attempts for a user, and thus, improve device locking usability.

In September 2013 Apple unveiled the Touch Id sensor in the new iPhone

5S smartphone. The finger-print sensor was designed to improve user experience with unlocking smartphones by simply pressing the home button. Such a design, however, raises an interesting research question “How does technology like Touch Id affect the strength of locking secrets that users choose?” Although Touch Id addresses certain usability problems (e.g., being able to unlock devices on the go), the strength of unlocking secrets is still crucial for confidentiality protection of data in smartphones. When a user fails to unlock his iPhone with a finger print, the device will ask for the unlocking secret. Furthermore, the underlying data encryption sub-system also uses unlocking secrets in the encryption key derivation process. Having a guessable unlocking secret increases the chance that an attacker will mount a successful password guessing attack, and sequentially gain unauthorized access to data-at-rest.

To understand how the use of Touch Id impacts users’ selection of unlocking secrets, Cherapau et al. [45] conducted three user studies based on interviews and surveys. The authors began with a study based on in-person surveys, which allowed us to measure the adoption of Touch Id and the strength of unlocking secrets. They then proceeded with interviews to understand the justification for the chosen unlocking secrets used with Touch Id. Finally, an online survey was conducted to measure the prevalence of the various adoption strategies established in the first two studies. Overall, the results of the study revealed that users do not take full advantage of the Touch Id sensor and still use guessable unlocking secrets. In particular, there was no statistically significant difference observed between the size of the search space of unlocking secrets chosen by users that used Touch Id and those that did not. That is, both groups relied mainly on 4-digit PINs. Similarly to Study 1, presented in Section 2.3, participants stated that they adopted short PIN codes instead of alphanumeric passwords because of the better usability of the former. These results suggest that the addition of biometric sensors, such as Touch Id, has not changed the current state of affairs on data security in lost or stolen smartphones. After we published our results, Apple introduced an option for using 6-digit PIN codes. It is still unclear, however, how many users would switch to this option, given that more than 30% of subjects in the Touch Id study were unaware that they could choose to use alphanumeric passwords.

Asking participants to admit socially undesirable behavior has its limitations, since users might be reluctant to share such information with others. To address this limitation Marques et al [83] used an anonymity preserving list survey experiment, which allowed users to share their experience without explicitly admitting their actions [84]. List experiments employ a between subjects design, i.e., two distinct groups of subjects. Both groups are presented with a list of activities or opinions, and the subjects are required to report the number of items that they have done or share. The lack of explicit selection provides subjects with a sense of anonymity. The control group has a list of four items, where two of the items chosen are highly likely to be selected by participants, e.g., brushing teeth or drinking water. The other two options chosen are highly unlikely to be selected by the participants, e.g., flying to the Moon. The treatment group also includes an extra option, which in our case was the act of snooping on someone else's smartphone.

For each of the groups the authors analyzed the reported numbers of items users selected. The difference between the averages corresponds to the ratio of subjects in the treatment group that selected the extra option. The results of the analysis revealed that the snooping rate was significantly higher, i.e., 31% in the recruited subjects pool, or 1 in 5 adults, if the results are prorated for the general population of smartphone users in the US. In addition, the results of this study confirmed the correlation between age and snooping behavior; younger smartphone users had a higher chance of being a victim of a snooping attack.

To gain a better understanding of how often users unlock their devices on a daily basis, Mahfouz et al. [82] conducted a field study with 41 smartphone users. Each participant installed a custom built monitoring application and ran it for at least 20 days. The application collected various data about smartphone usage, including the number of successful and unsuccessful unlocking attempts, the length of unlocking attempts, the number of failed unlocking attempts and other events. The results of the study revealed that users who used an unlocking secret unlocked their smartphone more frequently, i.e., 51 times a day on average against 41 times a day. Users who relied on PIN-codes were also less prone to make a mistake during the authentication process (0.5% error rate) in comparison to users who used Draw-A-Secret (3.5%) or an alpha-numeric password (4%). Although Draw-A-Secret (DAS) showed a significantly higher error rate than PIN-codes, it was used by 69% of subjects that used unlocking secrets, whereas PIN-codes were adopted by 22%. This suggests that users are willing to compromise, to some extent, the error rate of the authentication methods in smartphones for usability. Yet, it is still unclear which usability properties of an authentication method are the main factors for such decisions, e.g., ability to remember and recall secrets easily, ability to enter the secret easily, or ability to use the method while distracted and on the go, etc. It is also not clear to what extent users are willing to tolerate the error rate.

2.7 Discussion and Future Work

The results of Studies 1 and 2 revealed that users do store various types of data on their smartphones, including both relatively small data items, such as SMS messages, and large data items such as photos and videos. Sensitivity of data varied and also depended on the type of attacker. For instance, while contact details were considered sensitive if a stranger accessed them, users were not concerned with an insider reading them. On the other hand, users had higher concerns with an insider accessing certain personal data records, such as SMS messages. This suggests that a data protection system for smartphones should be both (1) efficient at supporting data types of various sizes, and (2) provide effective protection for data types against both insiders and strangers.

This chapter also presented analysis on the security practices users employ today, in order to protect confidential data in smartphones. In particular, while half of the recruited subjects used an unlocking secret, 95% of them chose either DAS or PIN-codes. These methods, however, allow the attacker to mount a relatively inexpensive password guessing attack. For example, it takes less than a second to go through all combinations of 4-digit PIN-codes for Android OS [104] and about 14 minutes for iOS [34, 37]. Even more, research by Raguram et al. [93] showed that these unlocking secrets can be reconstructed from recording reflections of a smartphone screen.

The results of the study on the impact of Touch ID on users' choice of unlocking secrets revealed that smartphone users still preferred weaker, easier to use authentication methods. Even more, about a quarter of the surveyed subjects stated that they previously used a locking secret, but decided to disable it due to various usability issues. Interestingly, a follow-up study revealed that a higher error rate of authentications does not necessarily correspond to a lower adoption rate (i.e., PIN-codes and DAS). This shows the importance of authentication method usability in smartphones for choosing which method to adopt. Considering that the same set of authentication methods is still being used today, and that the addition of finger print sensors, such as Touch ID, have not increased the entropy of unlocking secrets users choose, a data protection system should not assume that users will employ a hard to guess authentication secret to keep data in smartphones protected.

Another important result of the studies presented in this chapter is that the results of the studies provide evidence that smartphone users experience attacks by *insiders*. In particular, 89 subjects (or 12%) in Study 2 had caught someone from their social circle snooping through their smartphone without permission. Even more, 9% of surveyed subjects admitted that they had accessed data in someone else's smartphone without permission. The accuracy of this estimate was increased in the followup study [83], by revealing that 1 in 5 users in the US looked through someone's smartphone without permission.

These results suggest that novel authentication methods, especially those that are proposed to be used in smartphones, have to be evaluated against attackers that are as capable as *insiders*. In addition, the assumptions that the research community makes about safety of certain locations should be reevaluated. For example, both Riva et al. [94] and Hayashi et al. [72] proposed approaches to reduce the frequency of required authentications for smartphone unlocking based on location type. In particular, these authors assumed that home is safer and proposed disabling smartphone locking at home completely. The studies presented in this chapter, however, suggest the opposite, that the home can be full of insider attackers. This is especially true when a user has a roommate or a family member willing to peek into their smartphone.

Finally, the results presented in this chapter show that younger demographic groups have a higher risk of experiencing unauthorized access by *insiders*. It is still not clear, however, which factors increase or decrease this likelihood. The fact that younger users often share accommodation while in school could be one of such factors. Living with parents and siblings might also contribute to the increase of insider threat. Finally, more relaxed social norms, such as over-sharing, could also increase chances of unwanted access to data in a smartphone by an insider.

2.7.1 All-or-Nothing Locking Approach

These results presented in this chapter suggest that there is a gap in what the current smartphone locking systems provide and what smartphone users actually need, especially to protect themselves from insiders. Future research should focus on how to deter and prevent such attackers from unauthorized access. For example, in addition to smartphone locking, one can use facial recognition to detect when the current user is not the owner of the smartphone. The recently introduced Face ID unlocking mechanism in iPhone X could be the enabling technology for such an approach. In cases when a smartphone owner needs to share his device with someone, researchers might propose an easy-to-configure interface that unlocks certain parts of the smartphone that the owner considers public.

Recent research also studied contextual awareness for the unlocking process [73, 79, 92]. In such proposals, the context usually defines the complexity and usability of the unlocking process. In safe environments the user is required to go through a simple authentication mechanism, such as PIN-code or DAS. In untrusted environments the unlocking process relies on stronger authentication methods. Such approaches, however, might increase the mental load on users, and will have to be more carefully studied "in the wild".

Other researchers have tried to improve the current “all-or-nothing” model for smartphone locking. In such a model, the locking of a smartphone is either fully enabled or completely disabled. The all-or-nothing model, as the results from our studies showed, pushed 20% of smartphone users to disable smartphone locking. To address these limitations, Riva et al. [94] and Hayashi et al. [72] proposed automatic disabling of locking in certain assumed-to-be-safe locations, work or home. The results from our user studies, however, showed that these assumptions are questionable. In particular, the results revealed that these locations are full of insiders and users do experience invasion of privacy by such attackers. Thus, it is still not clear how to design a more granular access control system that takes context into account and provides greater flexibility to the users, while defending the users against potential insiders.

Finally, our studies also revealed that younger smartphone users have a higher risk of experiencing unauthorized access by *insiders*. It is still not clear which particular factors increase or decrease this likelihood. The fact that younger users often share accommodation while in school could be one of such factors. Living with parents and siblings might increase the *insider* threat. Further, more relaxed social norms, such as over-sharing, could also be a factor that increases the chances of unwanted *insiders* access. Future research needs to aim at improving our understanding of factors that impact such experiences.

2.7.2 Improving Security of Unlocking Methods

Increasing the complexity of unlocking secrets that smartphone users choose leads to stronger security of full-disk data encryption, since this derives its encryption key from the unlocking secret. This can be achieved by either nudging users to pick stronger unlocking secrets, or by improving authentication methods themselves in terms of security and usability. Persuading users to choose more complex secrets have been vastly studied in the usable security domain (e.g., see [58, 66, 77, 106]). For example, Egelman et al. [58] and Ur et al. [106] studied the effect on password strength meters on users' choice. While the results from both studies showed that password meters can be effective in improving users' choice, it is still unclear how these results would translate to mobile context, such as smartphone or typing on the go. Forget et al. [66] studied effectiveness of persuasive text passwords (PTP). In PTP, once a user chooses a password the system automatically adds a random character in a random place. User is allowed to shuffle the character and the position until he finds the combination that he accepts. The results of the user study showed this approach was mostly effective, with exception of password that were randomly chosen to begin with. Finally, Komanduri et al. [77] studied the effect password selection policies have on the actual security of passwords that users choose. Surprisingly, the study revealed that commonly adopted practices, such as requirement of having a special character in the password, lead to less secure passwords, while simple policies, such as lower-case 16 characters, allow users to choose less guessable passwords.

Other researchers have focused on improving the authentication methods themselves. For instance, both De Luca et al. [49, 50] and von Zezschwitz et al. [107] have proposed novel authentication methods that improved the usability of user authentication in smartphones, while addressing specific types of attacks, e.g., shoulder surfing [59, 99, 113]. These proposals, however, are still prone to password guessing attacks, due to a relatively small search space for authentication secrets, which is comparable to a commonly adopted 4-digit PIN-code.

Recent improvements in sensor capabilities provide additional opportunities for improving the usability of existing authentication methods that are believed to be unusable. For instance, a finger print scanner in iPhones (Touch ID sensor), significantly reduces the frequency of user authentication based on secrets. This makes it possible for a user to choose a more complex alpha-numeric password instead of relying on a 4-digit PIN-code. Unfortunately, one of the follow up studies [45] revealed that smartphone users still prefer easy-to-guess authentication secrets, such as 4-digit PIN-codes. The main reasons for such preferences were unawareness that a more complex option was available and the need to share unlocking secrets with others.

Finally, it is still unclear to what extent easy-to-guess unlocking secrets are being exploited. While there is some anecdotal evidence that attackers try to access private data on a stolen device (e.g., Honey Stick Project by Symantec [9]), such results were not obtained in a scientifically sound manner. In this work I assumed an opportunistic attacker – an attacker that aims to profit from the stolen smartphone itself, but who opportunistically tries to access data as well. There is no evidence to suggest that such attackers pose a real threat. This is why future research should also attempt to uncover the impact of data breaches that originate from stolen smartphones for which users choose easy-to-guess unlocking secrets.

2.8 Challenges

Conducting user research is challenging, but doing research on sensitive matters, such as private data in smartphones, adds another dimension of complexity - *ethics*. Researchers are bound to high ethics standards that often make certain kinds of research impossible. For example, when researchers attack existing systems, they often are limited to using the researchers themselves as subjects, e.g., evaluation of a shoulder surfing attack [93], or must clearly state their intent before the attack. Both of these approaches impact outcomes through bias.

In particular, in late 2012 I tried to evaluate the assumption that users are vulnerable to eavesdropping attacks in public places, such as coffee shops and while using public transit. The study design was based on observing users in real settings to assess (in)security of their unlocking secrets. To limit subject bias, we planned to approach subjects for debriefing and consent for including their data in the analysis after the observations were made. While initially our study was approved by the ethical board, the application was rejected two months later[4]. This made it impossible for me to assess how easy it was for an attacker to

[4]Application H12-02254, titled "Smartphone Unlock in a Wild". Approved on December 19, 2012. Rejected on February 25, 2013

eavesdrop a user's unlocking secret before stealing their smartphone.

Similarly, it is challenging to assess how often user data is being accessed by insiders, since it is challenging to obtain consent from the *insider* attackers without impacting their behavior. Furthermore, when it comes to strangers, it is unclear how often they actually try to guess unlocking secrets and decrypt data. All we have at the moment is the anecdotal evidence that attackers are interested in accessing private data and that users employ easy-to-guess unlocking secrets.

2.9 Conclusion

This chapter presents the results of user studies that focused on understanding users' concerns with sensitive data in smartphones in the presence of two different types of attackers, insiders and strangers. It provides evidence that an insider threat is real and that 1 in 5 users in the US had peeked into someone else's smartphone without permission. In addition, the studies revealed that the vast majority of smartphone users employ unlocking secrets that can be guessed within minutes. It also appears that the introduction of novel authentication mechanisms, such as Touch ID, did not have a considerable effect on password complexity.

These results suggest that research on novel authentication methods for smartphones needs to account for an attacker as capable as an insider. An insider is an attacker who aims to eavesdrop an unlocking secret and mount a so-called lunchtime attack, where a victim leaves her device unattended for a brief span of time, sufficient for the attacker to unlock it and gain unauthorized access to sensitive data. In addition, researchers should not assume that smartphone users will choose an unlocking secret that is complex enough to keep their sensitive data protected. As the results of the conducted studies suggest, the state is the opposite, users prefer choosing unlocking secrets that are easy to memorize and type, which, unfortunately, are also *easy-to-guess* for attackers.

Chapter 3

Analyzing Cryptographic API use in Android Applications

While smartphone users can protect of their data by choosing an unlocking secret that is hard to guess for an attacker in a reasonable amount of time (e.g., tens or hundreds of years), developers control secrecy of their applications' data by using encryption algorithms and protocols in a secure fashion. In this chapter I present the results of the analysis of how application developers use and misuse a set of cryptographic functions that are commonly employed for encryption key derivation, secure random number generation and symmetric ciphers.

3.1 Motivation and related work

The research community has paid a lot of attention to the (mis)use of cryptography in smartphone applications. For instance, Lazar et al. [78] studied Common Vulnerabilities and Exposures (CVE) that were reported between January of 2011 and May of 2014 that were related to cryptography. The results of their analysis showed that 83% of the CVEs were introduced by application developers that incorrectly used Crypto API. To understand how this issue can be addressed, Acar et al. [36] studied the usability of several cryptographic libraries. The results of their user study suggested that while making the Crypto API simpler had its benefits, application developers still required proper documentation, code samples and certain features to be available for the library to be used properly.

Several researchers used static analysis methods and tools to analyze Crypto API misuse in Android application binaries. For example, Fahl et al. [62] studied the misuse of asymmetric cryptography for SSL/TLS protocols, and certificate validation in particular. The analysis of 13,500 top free Android applications revealed that 8% of the analyzed applications misused SSL/TLS API, which made these applications potentially exploitable.

Egele et al. [57] designed and implemented the CryptoLint system based on AndroGuard [23] framework. CryptoLint used static analysis to identify misuses of Crypto API in Android applications. The authors defined six rules for correct use of Crypto API, which were either based on formal definitions, such as *Indistinguishability under chosen-plaintext attack* (IND-CPA) notion of security [52], or recently published reports. For instance, the use of the Electronic Code Book (ECB) mode of operation for symmetric ciphers is considered to be insecure under IND-CPA, since symmetric ciphers in the ECB mode produce exactly the same ciphertext for two identical plaintexts, allowing attackers to identify plaintext by the corresponding ciphertext.

Other rules, e.g., the use of SecureRandom class, require that developers do not use static seed values, since using static values makes a random number generator (RNG) predictable. If a predictable RNG is used for encryption key generation, then an attacker can seed his RNG with the same static value, generate the same encryption key, and, eventually, decrypt data.

When it comes to password-based encryption key derivation functions (PBKDF), one should be careful with two parameters: (a) the salt value, and (b) the number of iterations. The use of a static salt value allows attackers to employ a rainbow table approach [87], which can significantly reduce the computational efforts required to mount a successful password guessing attack. The number of iterations defines how much computational work an attacker needs to do for a single password candidate. Choosing the number below the recommended value of 1,000 iterations, results in faster computation for attackers.

The results of the analysis based on the CryptoLint system revealed that 88% of Android applications that used Crypto API violated at least one rule. Similarly to the CryptoLint study, we focus on the same set of rules (replicated in Section 3.2), while introducing source attribution to the analysis pipeline. In addition, we extended the original dataset of the CryptoLint study by adding newly collected applications from 2015 and 2016. To make reproduction of similar studies easier in the future, we made the BinSight tool available as open source.

Other researchers focused on libraries or source of the information the developers used. For instance, Derr et al. [39] studied how promptly application developers adopt new versions of libraries, especially when there is a known vulnerability in the library. While doing so, the authors also evaluated the six rules defined in the CryptoLint study for the identified libraries. Unsurprisingly, the results of the analysis revealed that libraries violated these rules too. In comparison, we studied violations that originated from either of the sources, i.e., a library or an application itself. Acar et al. [35] studied how the source of information that applications developers used during implementation of the applications impacted code security. The results of the studies showed that developers with no security background often use sources, such as Stack Overflow, that frequently contain insecure snippets. Furthermore, majority of the applications on Google Play contained security related errors that were common in the wild, including discussion threads on Stack Overflow. Similarly, Fisher et al. [64] showed that 15.4% of applications on Google Play contained security-related code snippets from Stack Overflow, 97.9% of which were insecure.

To summarize, while previous research has looked into either libraries or Android applications as a whole, there are still several open research questions on misuse of Crypto API. That is, it is unclear how similar or different the misuse of Crypto API in applications themselves and libraries. In addition, it is not clear if either of these source has changed since 2012 and how. Finally, it is unclear if libraries or applications contribute the most of Crypto API misuse cases. To answer these questions we ought to be able to attribute calls to Crypto API to libraries or to applications. From practical perspective, attributing a Crypto API misuse to its source has several important implications. First, one needs to clearly identify the responsible party for fixing the bug. Second, identifying the source of a misuse allows researchers to reduce over-counting of bugs, by identifying ones that originate from libraries. In addition, by being able to analyze binaries, the BinSight tool allows applications developers to obtain an insight into how a library (mis)uses Crypto API. This allows them to make an informed decision on whether or not they want to use this library in their application.

3.2 Common rules in cryptography

Similarly to the CryptoLint report [57], we analyzed the same set of rules for secure use of Crypto API. Throughout the rest of this chapter I use the term *APK file* to refer to an Android Application binary as a whole, i.e., when the origin of a call to Crypto API is not taken into account. Such treatment of Android applications as a whole resembles the reporting approach from the CryptoLint study [57]. I use the term *applications* to refer to the cases when Crypto API calls originate from Android applications themselves. Finally, I use the term *libraries* to refer to the cases where Crypto API are called from libraries.

An APK file was flagged as misusing Crypto API if it contained a violation of any of the rules from any source. While one can declare these APK files as insecure, we note misuse of Crypto API does not imply an exploitable vulnerability. The main reason for this argument is that developers might be using cryptography for purposes other than data confidentiality or integrity. For example, one might use Crypto API for obfuscation. In the rest of this section, the Crypto API use rules are explained in more detail. For a complete description of rules and their in-depth justification, please refer to the CryptoLint study report [57].

3.2.1 Symmetric key encryption

A block cipher is a deterministic algorithm that operates on fixed-length groups of bits, called blocks, with an unvarying transformation specified by a symmetric key. A stream cipher, on the other hand, is a stateful algorithm that combines plaintext digits with a pseudo-random keystream, which is generated from a symmetric key. Block and stream ciphers are used in symmetric key encryption to encrypt messages of arbitrary length. It is important to know that a symmetric key encryption scheme must be either probabilistic or stateful to be IND-CPA secure [40].

In block ciphers, a mode of operation defines security properties the cipher would provide, such as confidentiality. A popular mode is electronic codebook (ECB), which is a stateless, deterministic algorithm defined over a block cipher. As such, the ECB mode is not IND-CPA secure. The major problem with ECB mode is that identical messages encrypt to identical ciphertexts, which represents an information leak that is often intolerable. Still, ECB mode is commonly considered secure if the message is smaller than the size of the block in the underlying cipher, and all messages are unique. Therefore,

Rule 1: *Do not use ECB mode for encryption.*

Another popular mode of operation is ciphertext block chaining (CBC), which is an encryption algorithm that is built from a block cipher, where each block of plaintext is XORed with the previous block of ciphertext before being encrypted with the block cipher. The first block of plaintext is XORed with an initialization vector (IV). Using a constant IV will result in a deterministic, stateless cipher, which is not IND-CPA secure. Thus,

Rule 2: *Do not use a constant IV for CBC mode.*

Any symmetric encryption scheme, defined using a block or a stream cipher, should not reveal its key. If the key is hard-coded into a publicly-available application as a constant, then the key is not private, and so the resulting encryption does not provide confidentiality. Symmetric encryption schemes commonly assume a randomized key generation algorithm that should be used instead. Accordingly,

Rule 3: *Do not use constant encryption keys.*

3.2.2 Password-based encryption

User-created passwords are often predictable and prone to offline password guessing attacks [101]. For technical details on how an offline password guessing attack is usually mounted please refer to Chapter 1. To address this issue, password-based encryption (PBE) schemes try to increase the costs of such attacks by requiring significant amounts of computation in order to derive an encryption key. For example, in iOS the key derivation process is calibrated to take about 80 milliseconds [37]. This results in a significant increase of efforts for attackers, since they need to try thousands of different password candidates, while virtually having no effect on end-user experience. PBE schemes achieve this by using random salt value and applying multiple iterations of a cryptographic hash function, typically using a key derivation algorithm.

The salt and the iteration count entail a multiplicative increase in the work required for a password guessing attack [41]. Using constant salt values enables attackers to use the rainbow table approach [87], which significantly reduces computational burden in cases when passwords for multiple accounts are being guessed. Choosing low iteration counts results in less work per guess for attackers, which, again, speeds up the password guessing attack. For this reason, we chose to use 1,000 iterations as a minimum value, as suggested by RFC 2898 [76]. Hence,

Rule 4: *Do not use constant salts for PBE, and*

Rule 5: *Do not use fewer than 1,000 iterations for PBE.*

Note, that several recent publications proposed different approaches on increasing computational cost of key derivation process, e.g., scrypt [89] or using

10,000 iterations for PBKDF2 [70], in this work we decided to use 1,000 as our bare minimum in order to be able to compare with the results from 2012. Future research, however, should strongly consider increasing the number of iterations.

3.2.3 Random number generation

Android provides an API to a seeded, cryptographically-strong pseudo-random number generator (PRNG) via the SecureRandom class [31]. This PRNG is designed to produce non-deterministic output, but if seeded using a constant value, it will produce a constant, known output across all implementations. If such a PRNG is used to derive keys, the resulting keys would not be random, making the encryption insecure. As such,

Rule 6: *Do not use a constant to seed SecureRandom.*

3.3 Cryptography in Android

As discussed in Section 3.1, there are various reasons to use cryptography in Android by both applications and third-party libraries. In the following, a brief introduction is provided for the application ecosystem in Android, focusing on packaging and Java runtime, and for the use of cryptography in Java.

3.3.1 Android applications ecosystem

Android applications are authored as either native C/C++ or Java source code. In this study, only applications written in Java are considered, because Java has had a stable Crypto API interface since the release of Java 1.4 in 2002. An Android Java application is compiled to Dalvik executable (DEX) bytecode. This bytecode is packaged with additional resources, such as images, third party libraries, and configuration files, into an application package (APK) file. The APK file is then uploaded to the Google Play Store, and when a user installs the application, the APK file is downloaded and installed on their device.

Even though DEX bytecode is compiled from Java, the Dalvik virtual machine (DVM) is considerably different from the Java virtual machine. For example, while the Oracle Java virtual machine is stack-based, DVM is register-based, with a dedicated assembly language called Smali. However, it is possible to convert DEX bytecode to Oracle's Java bytecode with the Dex2Jar tool [19], albeit with some limitations, such as the inability to decode specific classes. We note that DVM was recently replaced by Android runtime (ART), which translates the DEX bytecode into the CPU's native instructions for faster execution.

3.3.2 Java cryptography

Android provides a rich execution framework that offers access to various subsystems, including Java cryptography architecture (JCA). The JCA standardizes how developers make use of many cryptographic algorithms by defining a stable API. Accordingly, a cryptographic service provider (CSP) is required to register with the JCA in order to provide the actual implementation of these algorithms. This abstraction allows developers to replace the default CSP, which is BouncyCastle [18] in Android, with a custom CSP that satisfies their demands. For example, SpongyCastle [22] is a popular third-party CSP that supports a wider range of cryptographic algorithms.

Symmetric and asymmetric encryption schemes are accessible to developers through the Cipher class, as described in Listing 3.1. To use a specific encryption scheme, the developer calls the Cipher.getInstance factory method and provides a transformation as an argument. A transformation is a string that specifies the name of an algorithm, a cipher mode, and a padding scheme to use. In Listing 3.1, the returned cipher instance uses AES in CBC mode with PKCS#5 padding. Only the algorithm name is mandatory, while cipher mode as well as the padding scheme are optional. Unfortunately, all CSPs default to the ECB mode of operation if only the cipher name is specified, which is insecure [30].

Listing 3.1: Simplified symmetric key encryption in Java

```java
// values of iv and key should be randomly generated
public byte[] encrypt(byte[] iv, byte[] key, byte[] data) {
  IvParameterSpec iv_spec = new IvParameterSpec(iv);
  SecretKeySpec key_spec = new SecretKeySpec(key, "AES");

  Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5PADDING");
  cipher.init(Cipher.ENCRYPT_MODE, key_spec, iv_spec);

  return cipher.doFinal(data);
}
```

3.4 Datasets

As summarized in Table 3.1, three datasets were used for the analysis, 132,590 APK files in total. R12 is a subset of the CryptoLint dataset with 10,990 APK files. The original dataset had 145,095 APK files and was collected between May and July of 2012 [57] by crawling the Google Play marketplace. First, the authors of CryptoLint excluded all APK files that did not use Crypto API. Second, the authors also excluded all APK files that had all Crypto API calls originating from 11 white-listed libraries. This resulted in a sub-set with 15,134 APK files. The CryptoLint tool, however, failed to analyze 3,386 files from this sub-set and 758 files were lost since 2012, resulting in 10,990 APK files in the R12 dataset. Considering that the 758 lost files are a random sample of the set that was presented in the CryptoLint report [57], such loss does not have a significant impact on our results.

R16 dataset was collected in May of 2016 with the help of Sophos. To select APK files in the R16 dataset we first generated a random sample of 120,000 APK files that were available on Google Play market at that time and then downloaded that set from Sophos servers. Unfortunately, some of the files were corrupted during the downloading process, leaving us with 117,320 APK files.

For trend analysis we focused only on the R12 and R16 sets. In addition, we considered two versions of the R12 and R16 datasets. First, we analyzed subsets of R12 and R16 from which we removed all APK files that had all calls to Crypto

Name	Number of APKs	Sampling	Year
R16	117,320	Random	2016
R12	10,990	Random	2012
T15	4,280	Top-100	2015

Table 3.1: Summary of used datasets

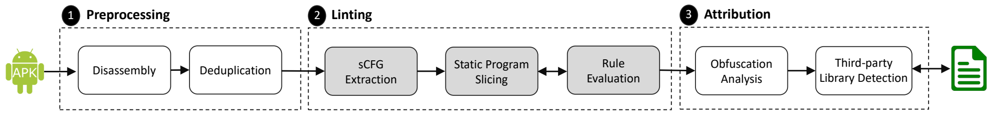

Figure 3.1: Cryptographic API linting for Android applications using BinSight. Gray components represent parts that were reimplemented from CryptoLint [57], and white components represent the extensions that we added.

API originating from 11 libraries selected by the authors of CryptoLint. We denote these sub-sets as R12* and R16* respectively. Second, we analyze both R12 and R16 *as-is*.

Finally, the T15 dataset includes the Top 100 Android applications in each category from June 2015. For this dataset a list of the Top 100 applications was first obtained through Google Play store API. Then each APK file was separately downloaded through the ApkDownloader tool [25]. The downloading process was completed between June 13–28, 2016. As 20 applications were removed from the Google Play Store before we were able to download them, the final size of the dataset is 4,280[1]. We compared T15 to R16 only for additional insight into differences between average and top Android applications.

3.5 Crypto API linting with BinSight

At a high level, the rules defined in §3.2 represent temporal properties that can be validated using automated program analysis in a task known as linting [61]. Lint-

[1]Due to large size of T15 and R16 datasets we cannot make them available online, but will share upon request. For the R12 dataset we refer readers to the authors of the CryptoLint study.

ing is a process of validating certain formal conditions on source code through static analysis of the code or binary. Usually, it implies that one converts an application into a super control flow graph (sCFG) representation and then analyzes the structure of the graph to validate defined conditions. For example, one can trace all the inputs for a variable that holds an encryption key and check whether or not that variable holds static values, i.e., values that are hard-coded, or dynamically generated values, e.g., through a random number generator.

While previous research has proposed various linters for Android Crypto API [57, 103], they suffer from various limitations. In particular, the state-of-the-art linter CryptoLint is not available as open source and is unable to analyze over 23% of APK files [57] . In addition, none of these tools provide any code navigation, which is valuable for manual in-depth analysis. Finally, existing tools do not support attribution of the source of misuse, i.e., by using these tools one cannot tell whether a misuse is due to an application code or a third-party library.

To overcome these limitations, we developed BinSight framework based on technical description of the CryptoLint [57]. Although the BinSight tool does not introduce any novel ideas to the field of static analysis, it nevertheless improves analysis stability over CryptoLint, provides a rich, graphical UI for manual inspection of an APK file, and attributes a Crypto API call to an application or a library. To improve accuracy of the analysis conducted in the CryptoLint study, we introduced two additional stages to the APK analysis pipeline, as illustrated in Figure 3.1. We released BinSight as an open source project and included implementation details as comments in the source code[2].

In what follows, each stage of the analysis pipeline is discussed.

3.5.1 Preprocessing

Each downloaded APK file undergoes a two-step pre-processing stage before it is linted. The goal of this stage is to filter out all applications that do not use Crypto API and remove all duplicate APK files, as described below.

[2]The project can be located at the following URL: https://github.com/iim/binsight

Disassembly

Similar to CryptoLint, our analysis operates on a higher-level representation of the Dalvik bytecode. In particular, we use ApkTool [17] to decode an APK file and disassemble it into a set of Smali files. Each Smali file represents a class definition, and uses DEX operation codes to represent instructions [20]. We picked ApkTool over AndroGuard [23], which was used by CryptoLint, to improve analysis reliability. As shown in §3.6, BinSight was unable to analyze six applications out of 95K, while CryptoLint failed to analyze 23% of applications out of 15K.

After an application was disassembled, we searched all of its generated Smali files to locate entry points to Crypto API. If such entry points were not found, the application was disregarded from further analysis. Otherwise, we proceeded to the de-duplication step.

Deduplication

Downloading thousands of APK files from Google Play is technically challenging. First, it has to span over weeks or months in order to avoid account blocking. Second, an application might be listed in multiple categories. These challenges lead to duplicates in a dataset, and thus, removing duplicates is important for validity of the results. For de-duplication we relied on application Id (stored in the manifest file).

For each dataset we separately generated a list of all APK filenames, corresponding application Id and its download time (for T15 and R16 sets), or, when available, application version (R12). We then identified all duplicates within a dataset by grouping files with the same application Id. For identified duplicates within a dataset we kept the latest version of the application, based on its download date or version.

3.5.2 Linting

Once the interesting pool of APK files was identified, BinSight evaluated the common cryptographic rules defined in Section 3.2. In particular, it computed static program slices that terminate in calls to Java Crypto API, and then extracted the necessary information from these slices to evaluate the rules.

In what follows I provide a brief overview of the three main steps involved in this stage. I refer the reader to related work for further details [57, 103].

Super Control Flow Graph extraction

It is typical for an application to use Crypto API in multiple methods. For example, a cipher object could be instantiated in an object constructor and then used in two different methods to encrypt and decrypt the data. If the two methods were analyzed in isolation, we would not be able to extract the encryption scheme that was used when the cipher object was instantiated. Fortunately, the super control-flow graph (sCFG) of an application allows us to perform inter-procedural analysis, which is required to correlate the use of a cipher object for encryption and decryption with its instantiation.

BinSight constructs the sCFG of a preprocessed application as follows. First, it extracts the intra-procedural CFGs of all methods from the decoded Smali class files. This task also involves translating all methods into single static assignment (SSA) form [47], and extracting the class hierarchy of all classes in the application. Next, BinSight superimposes a control-flow graph over the CFGs of the individual methods, resulting in the sCFG. In this sCFG, call edges are added between call instructions and method entry points, and method exit points are connected with exit edges back to the call site.

One one hand, similar to CryptoLint, BinSight constructs an over-approximated sCFG of the application. That is, BinSight extracts calls that might never be executed when the application is running. On the other hand, BinSight does not analyze dynamic edges, which could be created through Java Reflection API [29]. This creates a risk of missing crucial calls to Crypto API, hence the resulting

Endpoint signature	Rule
Cipher.getInstance()	1
cipher.init()	2
secureRandom.setSeed()	6
new SecretKeySpec()	3
new PBEKeySpec()	4
new PBEParameterSpec()	5
new SecureRandom()	6

Table 3.2: Cryptographic API endpoints and related rules.

sCFG would become under-approximated.

Static program slicing

Static program slicing is the computation of a set of program statements, called slices, that may affect the values of certain variables at a particular program point of interest, referred to as a slicing criterion [44]. BinSight applies static program slicing on the sCFG to identify if the analyzed application uses any of the Crypto API. In particular, BinSight searches the sCFG for nodes that belong to Java's Crypto API endpoints. If these nodes are found, it uses their incoming edges to locate all call sites in the application. Note that this search depends on the type of the Crypto API endpoint in the sCFG. Table 3.2 shows the relevant API endpoints and their corresponding cryptographic rules.

Rule evaluation

Rule evaluation depends on the values assigned to the parameters of a Crypto API, where value assignment can be either local or external to the containing method. For the earlier case, BinSight computes a backward slice of the program to all possible locations where the involved parameter is set, after which we apply validation logic on its value. As for the latter case, the evaluation depends on the origin of value assignment outside the method. As such, BinSight computes backward slices to all locations where this value can be assigned. BinSight stops the computation if it reaches a dead-end, where a node does not have any incoming edge or it reaches an assignment to a static value.

3.5.3 Attribution

After the linting stage, every call site that terminates in a Crypto API is attributed to its source, which could be the application code itself or a third-party library. Our attribution approach relies on package names of classes that have API call sites, and cross-references them with an exhaustive list of third-party libraries. The attribution has to overcome obfuscated package names in order to correctly map call sites to libraries. This is done in the following two steps.

Obfuscation analysis

Although de-obfuscating Android applications has been recently studied [43, 81], the underlying techniques, while effective for manual forensics, are inefficient for analyzing applications on a large scale. Moreover, it is unclear how prominent the use of obfuscation is in the real-world, especially in the classes that use Crypto API. To automatically detect the level of obfuscation a rule-based classifier was developed that identifies whether a given package name is fully obfuscated or not. The manual analysis of the results of the classifier revealed that even if the package was partially obfuscated, one can still use it to identify the library it belongs to. Section 3.6 shows that less than 4% of package names were fully obfuscated, requiring sophisticated de-obfuscation techniques.

The following rules were defined as a result of several manual iterations over the results the classifier provided. Our main objective at this point was to find patterns that one can use in the classifier. Eventually, seven rules (listed below) were assembled, which allowed automatic assignment of the level of class identifier renaming (CIR), i.e., none, class, partial, and full CIR obfuscation.

1. If all parts of the identifier are of length one, then it is a case of full obfuscation. An identifier of a class is the combination of the class name and

its package name, e.g., `com.google.ads.ShowAd` defines a class `ShowAd` in `com.google.ads` package.

2. If all but the first part of the identifier are of length 1 and the first part is in the set {com, ch, org, io, jp, net}, then it is a case of full obfuscation. For instance, the example provided in step 1 would translate to `com.a.a.B`, where `com.a.a` would be renamed package name and `B` would be renamed class name.

3. If none of the package name parts in the identifier are of length one, then it is a case of either none or class-level obfuscation. The intuition behind this rule was based on the observation that, obfuscating software tends to rename parts to a single character.

4. If at least one part but not all of the identifier are of length one, then it is a case of partial obfuscation. The intuition behind this rule was the same as for the step 3.

5. If class name is longer than 3 chars then it is a case of no obfuscation. Similarly to rules 3 and 4, we observed that if a class names were renamed to names with one or two characters.

6. If class name length is 1 character, then it is a case of class name obfuscation.

7. If class name length is 2 or 3 characters and the first character is in lower case, then it is a case class name obfuscation. We observed that naming convention for Java classes uses an upper-case letter for the first character, thus, the cases where the first character was in lower-case and the length of the name was 2 or 3 characters it meant that the name is obfuscated.

Third-party library detection

As mentioned above, almost all call sites that terminate in cryptographic APIs correspond to package names that were identifiable. The fully obfuscated pack-

	Number of APKs			
Name	Total	Unique	Dups	Crypto?
R12	10,990	10,222	768 (7%)	10,222 (100%)
R16	117,320	115,683	1,637 (1.4%)	95,775 (82.8%)
R16*	117,320	115,683	1,637 (1.4%)	93,994 (81.3%)
T15	4,280	4,067	213 (5%)	3,645 (89.6%)
Total	132,590	129,972	2,618	109,642

Table 3.3: Summary of duplicates and Crypto API use in all three datasets

age names were labeled as "obfuscated", meaning that BinSight was unable to attribute them to a library or the application.

For the remaining majority of the package names a frequency analysis was conducted. All non-obfuscated package names that were seen in a single APK file were assigned to the application category. For the remaining set of package names a manual analysis was conducted, starting with the most frequently encountered package names first. The manual analysis was finished once it covered 95% of the call-sites. The remaining package names, which were not covered by the manual analysis, were labeled as "possibly library". As a result of this analysis, a list of package names was compiled that allowed BinSight to identify libraries and possible libraries. Note that this approach complements a recent proposal that relies on a list of library signatures generated from a large database of third-party SDKs [39].

3.6 Measuring Crypto API misuse

This section presents the result of the analysis of 109,642 APK files (out of 132,590) that had at least one call to Crypto API. To the best of our knowledge, this is the largest dataset analyzed for Crypto API misuse (e.g., the CryptoLint study is based on the analysis of 11,748 APK files only). First, we discuss duplicates, obfuscation detection and source attribution for each of the datasets. Then we present the overall statistics on Crypto API misuse, and proceed with the analysis of each rule separately. In our analysis we compare R12 to R16 in order to understand what may have changed between 2012 and 2016, and T15 to R16 in order to understand how a top application differs from an average one.

For each comparison we conducted a statistical significance test (Chi-square) to test whether the found difference was statistically significant with 99% confidence. In what follows we discuss only statistically significant results, and all figures show 99% confidence interval whiskers.

3.6.1 Preprocessing

Unsurprisingly, every application in R12 made at least one Crypto API call, confirming the analysis and the white-listing performed by Egele et al. [57]. Interestingly, while Egele et al. found that only 10.4% of the applications in their original dataset with 145K APK files made a call to Crypto API, this ratio has significantly changed for R16 and T15. In particular, we found that 83% and 90% of the applications in R16 and T15, respectively, made at least one call to Crypto API. Such a significant increase in use of Crypto API in Android applications can be attributed to many reasons, including white-listing that authors of CryptoLint applied or increased necessity to protect user data.

Our analysis revealed that while all datasets contained duplicates, R12 had the largest ratio, of 7%. We removed all duplicates from the analyzed datasets. The summary of the datasets after de-duplication is shown in Table 3.3.

Unlike CryptoLint, BinSight was able to disassemble and analyze all but six of the 132,590 APKs, which represents a significant improvement over the CryptoLint tool, and which failed to analyze 3,386 APK files (23% of the analyzed set) due to technical problems[3]. BinSight completed analysis in about 14 days on a dual Xeon CPU computer with 128GB RAM, i.e., processing about 7500 APK files a day, which suggests that BinSight is not only robust, but also scalable. We

[3]According to CryptoLint authors, there were two major problems: (a) the tool did not finish analysis within 30 minutes, and (b) the analysis infrastructure ran out of memory.

	Class identifier renaming level				
	None	Class	Partial	Full	Total
R16	509,643 60.64%	203,447 24.21%	106,091 12.62%	21,279 2.53%	840,460 100%
R12	78,883 77.12%	14,513 14.19%	6,882 6.73%	2,002 1.96%	102,280 100%
T15	26,821 59.40%	12,907 28.59%	3,620 8.02%	1,804 3.99%	45,152 100%

Table 3.4: Obfuscation analysis of class identifiers.

made BinSight available as an open source project[4].

3.6.2 Linting and attribution

Obfuscation analysis

As noted in Section 3.5, it is unclear how prominent obfuscation is, in particular in class identifier renaming (CIR) [39]. To understand this, the reliability of using package names for source attribution was analyzed, by quantifying CIR in each dataset. The analysis was limited to only those classes that made at least one call to Crypto API. While this served the needs of this study, these results on the prevalence of obfuscation should not be considered as a generalization to all Android applications.

There are different levels at which CIR can be applied by an obfuscator like DexGuard. For instance, for class `com.domain.package.Class`, an obfuscator might not change the identifier at all. It might rename the class name only, parts of the package name, or sometimes the whole class identifier. For the first three levels, we can map the class to a library or an application if the package name has an identifiable prefix. As for the fourth level, we cannot use the package name for source attribution.

[4]https://github.com/iim/binsight

		Source (%)			
	Call sites	Libs	Apps	Libs?	?
R16	840,460	90.7	4.9	1.9	2.5
R12	102,280	79.5	14.5	4.0	2.0
T15	45,152	80.6	10.7	4.7	4.0

Table 3.5: Attribution of cryptographic API call sites.

The goal of this analysis was to quantify how many class identifiers fall into each of the four CIR levels, as follows. First, a list was automatically compiled that contained all unique class identifiers that made at least one call to Crypto API. Then the list was manually inspected, in order to derive patterns for full and partial obfuscations. After the set of identified rules was updated, the list of package names that made at least one call to Crypto API was recompiled. At this point, the category was also added to each package name, so that the further analysis was focused on package names without category. The process was completed in four runs, which resulted in the list of seven classification rules presented in Section 3.5.3. While the presented rules were simple to evaluate, they suffered from false positive and negative cases. The analysis of those case showed that the number of such cases was below 1% out of total number of cases (i.e., false positive and negative rates combined).

The results of the analysis revealed that using package names for source attribution is still reliable. In particular, for applications in R16 we were able to identify the source for 97.5% of classes that made calls to Crypto API. The results of the analysis for all datasets are provided in Table 3.4.

Third-party library detection

We classified package names into one of four categories: applications (apps), libraries (libs), possible libraries (libs?), and obfuscated (?). We now describe how we performed this classification. First, we assigned all package names that have been fully obfuscated to the *obfuscated* category. We then assigned all package names that were found in a single application into the *applications* category. For the remaining packages, which were found in two or more applications, we ranked them based on how many applications used them in each dataset, and then performed manual inspection in a decreasing order of rank. In particular, for each package name we labeled as a library if we were able to find the library's source or website, or if our manual inspection revealed that it was indeed a library. We stopped manual analysis once we identified enough package names to cover 95% of the call sites. We assigned the remaining unclassified package names to the *possible libraries* category.

In total, we manually analyzed 12,165 package names from the three datasets, out of which 3,622 (29.7%) belonged to libraries. Overall, we identified 638, 260, and 265 libraries in R16, R12 and T15, respectively. The top-2 libraries and their package names are shown in Table 3.7. The fact that our analysis revealed 260 libraries in the R12 dataset suggests that the CryptoLint study suffers from over-counting misuse cases.

Source attribution based analysis revealed that the libraries were responsible for the majority of calls to Crypto API in all three datasets, as summarized in Table 3.5. Furthermore, 79.5% of all calls to Crypto API in the R12 dataset originated from 260 libraries. While the authors of the CryptoLint study did white-list 11 libraries, our analysis shows that the white-listing approach was inadequate. In particular, the authors missed 249 libraries, which accounted for 79.5% of the calls in their dataset. This suggests that the reported results in CryptoLint study suffer from over-counting, due including same library in their statistics multiple time. Overall, these results with missed libraries and the ratio of calls that originate from libraries suggests that researches that analyze Android applications for Crypto API misuse one should be careful misuse inflation, due to counting the same misuse cases from a library multiple times.

To this end, we showed that (a) one can reliably use package name for source attribution, since this covers 97.5% of the calls to Crypto API, and (b) libraries are the major contributor to Crypto API calls and should be properly identified.

3.6.3 Crypto API misuse in Android Applications

In what follows we present the main findings on Crypto API misuse rates across all source categories (applications or libraries) for all three datasets. We begin with the results of the analysis on overall misuse rates across all rules, i.e., at least one rule is violated. Then we proceed with an in depth analysis of misuse rates for each rule separately.

To understand the extent of which the white-listing approach used in the CryptoLint study has impacted results, we additionally analyzed two sub-sets R12* and R16*, which were generated by applying the same white-listing approach to the R12 and R16 datasets respectively.

We measured misuse rates from two complimentary perspectives. First, similar to the CryptoLint study, we assessed the ratio of APK files that contained a misuse. Second, we assessed the ratio of Crypto API call-sites that made a mistake. While the APK files ratio provides intuition into how many APK files contain at least one misuse of Crypto API, such an approach is biased towards libraries, especially popular ones. Call-site ratio provides an assessment of the likelihood that a call from an application or a library will make a mistake. And since we separate calls from libraries and applications, this measure provides a stronger intuition into how misuse rates have changed within libraries and applications.

The following reports the ratio of APK files with misuse for each category separately (named accordingly) and overall ("Any" category). The ratio for each category is computed for the total number of APK files in the dataset. This, however, does not imply that the sum of ratios for all four categories will be equal to the "Any" category, since an APK file might have misuses contributed by various sources. Similarly, the ratio of call-sites with misuse contains the "Any" category, which shows the ratio for all call-sites. Unlike the ratio of APK files, the ratio of call-sites with misuse is assessed separately for calls within each category.

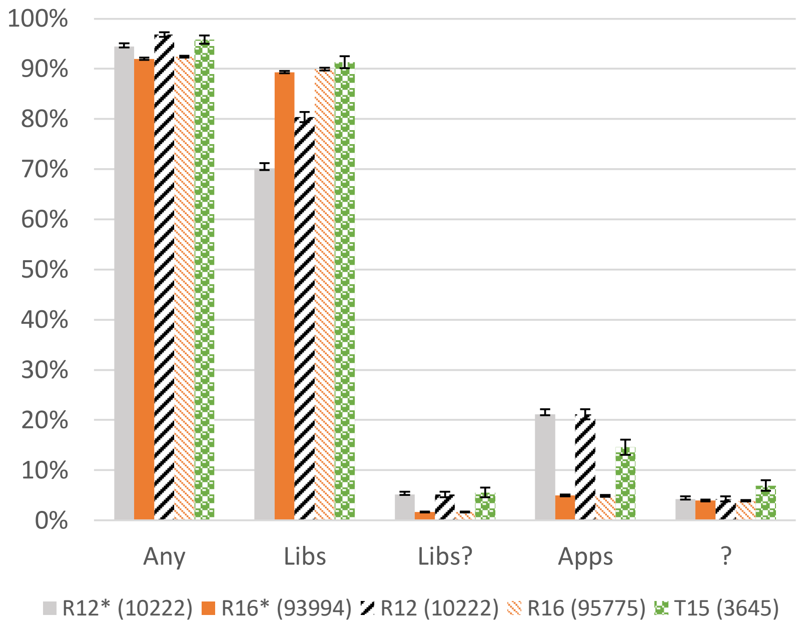

Figure 3.2: Ratio of APK files that violated at least one Crypto API use rule per dataset. "Any" category includes all call-sites for the analysis, without considering the source (i.e., library or an application). This approach was used in the CryptoLint study. The remaining categories (Libs, Libs?, Apps and ?) include call-sites that belong to the corresponding source only (i.e., a library, a possible library, an application or a fully obfuscated case). The proportions are calculated as the ratio of APK files that contained at least one misuse from specific category (or, any category for "Any") against the total number of APK files that used Crypto API in the dataset. The total number of APK files that made at least one call to Crypto API for each dataset is provided in the legend.

Overall Crypto API misuse rate

The ratios of APK files with at least one violation of the rules per category are shown in Figure 3.2. Unsurprisingly, our results for the R12* subset were in-line with previously reported results, i.e., 94.5% in our study and 88% in CryptoLint study [57]. We attribute the difference here to two factors: (a) we removed 7% of duplicates, and (b) 768 APK files from the original R12 dataset were lost. As

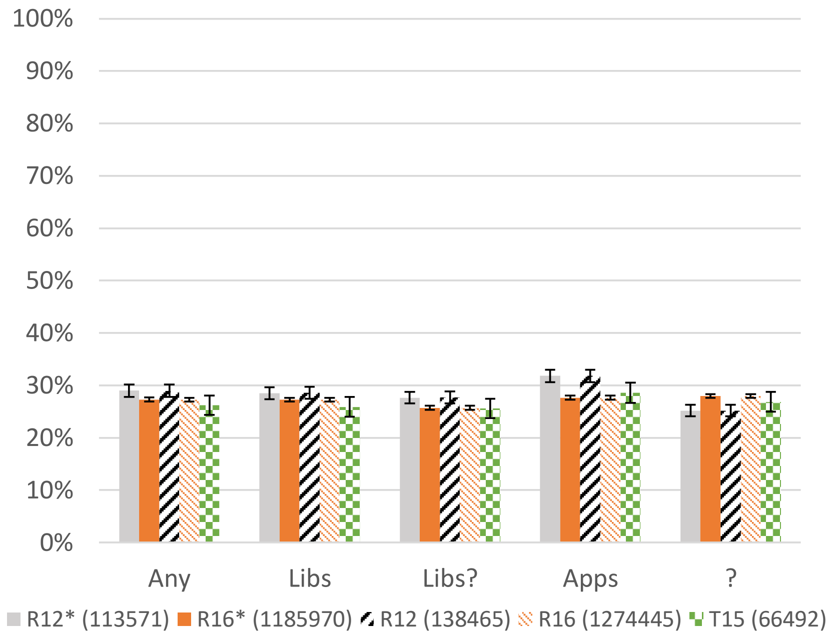

Figure 3.3: Ratio of call-sites that violated one Crypto API use rule per dataset. The total number of call-sites to Crypto API for each dataset is provided in the legend.

expected, we found that the white-listing approach used in the CryptoLint study reduced the ratio of APK files to include libraries that had introduced misuses. However, this did not have any impact on the call-sites ratio, as shown in Figure 3.3.

Overall, we found that since 2012 the ratio of APK files with at least one misuse has decreased from 94.5% to 92.4%. At the same time, the overall likelihood of a call-site to Crypto API to make a mistake remained around 28%, that is, *each fourth call to Crypto API makes a mistake*. Per category analysis, however, showed that while libraries have increased the ratio of APK files they introduced to Crypto API misuses to 90% (from 80%), the likelihood of a call-site from a library to make a mistake did not change significantly. The increase in the ratio of APK file libraries introduced to misuses of Crypto API can be explained by the overall increase in the number of libraries. In particular, while the R12 dataset contained only 260 libraries, R16 had 638 identified libraries.

Unlike libraries, applications have improved in both the ratio of APK files and the likelihood of a call-site that makes a mistake. In particular, the ratio of APK files decreased from 21% to 5% and the ratio of call-sites from 31.8% to 27.7%. Although, the increase in the total number of libraries might have also impacted the ratio of APK files for applications.

Comparing T15 with R16 revealed a single statistically significant difference, namely, the ratio of APK files to which applications introduced misuses. While the ratio for R16 was 5%, for T15 it was 14.6%. This difference can be attributed to various factors, such as the difference in the total number of libraries (265 in T15 against 638 in R16).

Symmetric key encryption

Data analysis revealed that the overall use of ECB mode for symmetric ciphers has significantly decreased since 2012, as shown in Figures 3.4 and 3.5 . That is, the number of APK files that were flagged as using ECB mode has dropped from 77% in R12 to 30% in R16. Similarly, the ratio of relevant call-sites that use ECB mode has dropped from 53% to 29%. Source attribution, however, revealed that this decrease can be mainly attributed to improvements in libraries. In particular, while applications decreased the ratio of relevant call-sites that use ECB mode from 63% to 47%, libraries have reduced this ratio from 52% to 26%, i.e., a two fold improvement. Comparison of the T15 and R16 datasets revealed that an average application is less likely to use the ECB mode than the top application. The white-listing approach used by the CryptoLint study had a negligible impact on the results, i.e., most of the introduced differences were either not statistically significant or practically negligible.

Despite the positive outlook on the use of ECB mode, we found that there was a statistically significant increase in the use of static IVs, as shown in Figures 3.6 and 3.7. In particular, since 2012 the ratio of APK files that use a symmetric

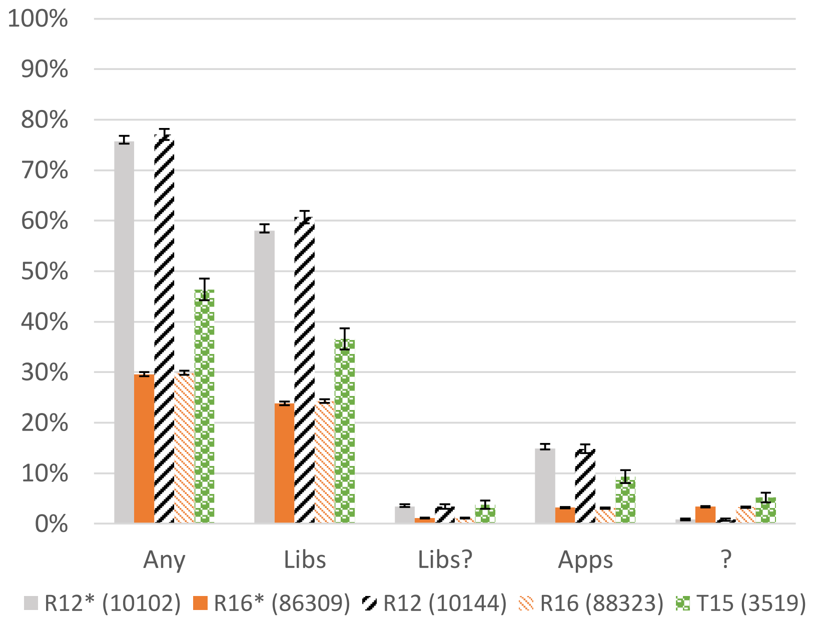

Figure 3.4: Ratio of APK files that violated Rule 1 - "Do not use ECB mode for symmetric cipher." The total number of APK files that used symmetric cipher per dataset is provided in the legend.

cipher in CBC mode to static IVs increased from 32% to 96% by 2016. The ratio of relevant call-sites to symmetric cipher APIs has increased from 31% to 71%. The increases were mainly due to libraries rather than applications, since the later actually improved and decreased their use of static IVs. In particular, applications reduced the ratio of call-sites that violate Rule 2 (use of static IV with CBC mode) from 64% to 52%, while libraries have increased that ratio from 26% to 72%. Comparison of the T15 and R16 datasets did not reveal practically significant results, i.e., the T15 dataset was comparable to the R16 dataset. While the white-listing approach did have a statistically significant impact on the R12* sub-set, it did not have similar effect on the R16* sub-set and it did not have an effect on the overall results, since all differences were still statistically significant.

The decrease in use of ECB mode and the increase of static IV use with CBC

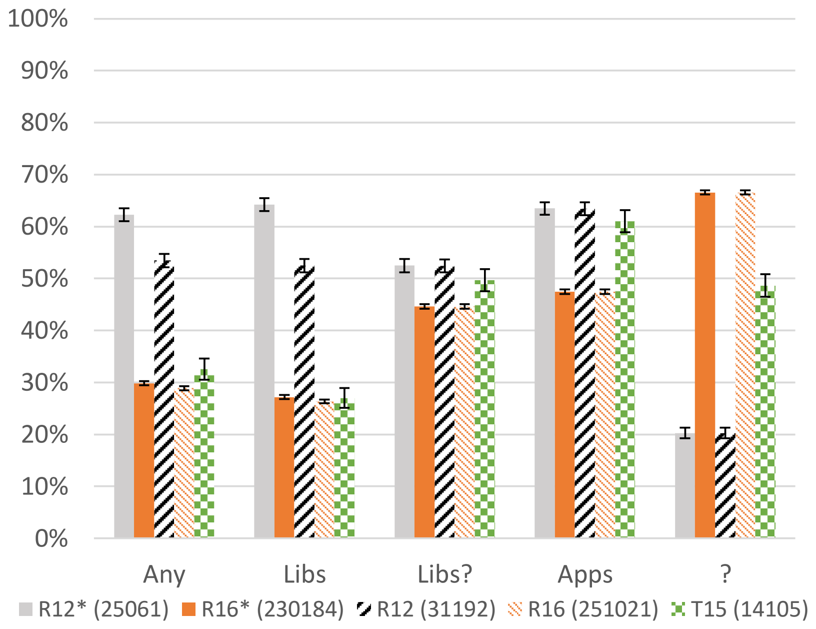

Figure 3.5: Ratio of call-sites that used ECB mode for symmetric cipher. The total number of call-sites that created symmetric Cipher objects in Java per dataset is provided in the legend.

mode, suggest that while developers tried to move away from insecure ECB mode, they failed to adopt a secure mode for symmetric ciphers. This failure might be explained by a lack of understanding. Another factor that may have impacted the shift is the warning message that the Android Lint tool began to show after the CryptoLint study was conducted. The warning message highlights that ECB mode is default and is insecure (*"...because the default mode on android is ECB, which is insecure."*). In fact, Crypto Stack-Exchange[5] is full of questions and suggestions on how to "fix" this warning message by replacing "ECB" mode with "CBC".

The analysis of Rule 3 violations (not using static encryption keys) revealed that, again, the overall rates of static key usage have increased (see Figures 3.8

[5]http://crypto.stackexchange.com/

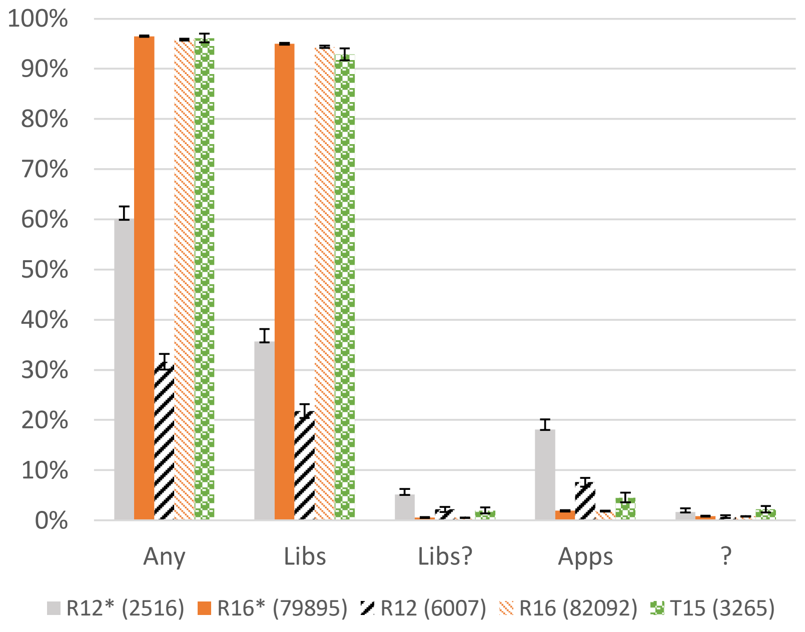

Figure 3.6: Ratio of APK files that violated Rule 2 - "Do not use static IV for CBC mode in symmetric cipher." The total number of APK files that used symmetric cipher in CBC mode per dataset is provided in the legend.

and 3.9). In particular, since 2012 the ratio of APK files that use symmetric cipher with a static key increased from 70% to 93%. The ratio of call-sites that use symmetric cipher with static key increased from 45% to 57%. Unlike with the use of static IV, both applications and libraries were to blame. Although applications did decrease the ratio of APK files that contribute to a violation of rule 3, the ratio of call-sites that violate rule 3 and originate from applications showed the opposite, i.e., the likelihood that a call from applications would use a static encryption key increased from 40% to 44%. This highlights that relying solely on the ratio of APK files with misuses might be misleading due to the impact libraries have on this measurement.

In addition to validating the formal rules of using cryptography, we extracted the top-6 most-used symmetric ciphers from each dataset, as summarized in Ta-

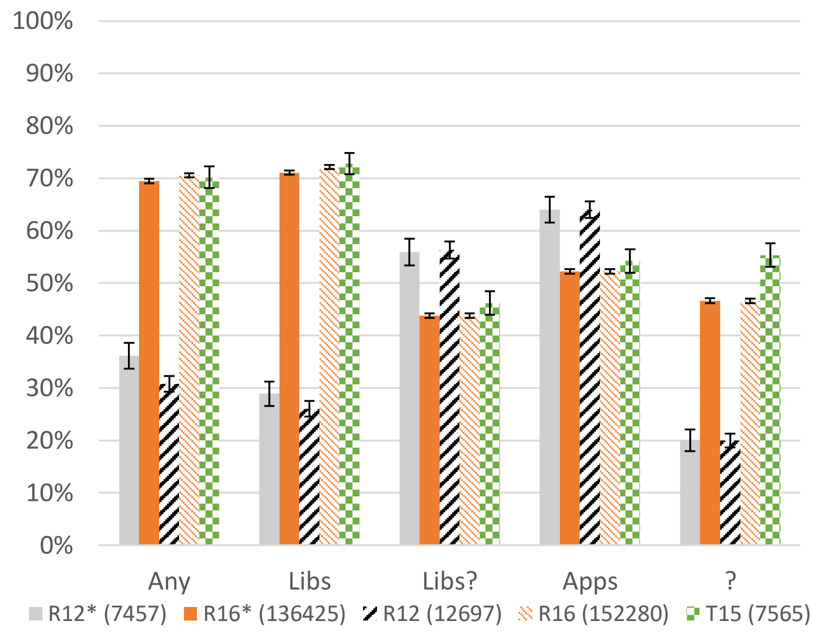

Figure 3.7: Ratio of call-sites that used static IV with CBC mode for symmetric cipher. The total number of call-sites that used Cipher objects in CBC mode per dataset is provided in the legend.

		Cipher (%)						
	Call sites	AES	DES	3DES	PDE*	RC4	Blowfish	Others
R16	251,021	64.4	13.6	1.1	0.7	2.1	0.9	17.2
R12	31,192	58.9	12.5	8.8	6.5	0.4	1.9	10.9
T15	14,105	67.8	8.9	0.8	0.9	1.1	0.8	19.7

Table 3.6: The top-6 ciphers used in Android applications. PDE was used with MD5 and 3DES.

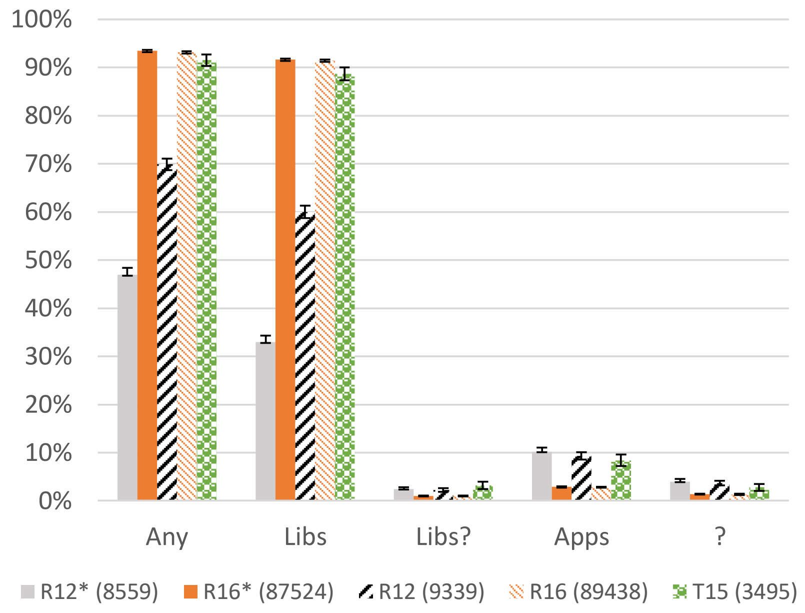

Figure 3.8: Ratio of APK files that violated Rule 3 - "Do not use static encryption key for a symmetric cipher." The total number of APK files that used symmetric cipher per dataset is provided in the legend.

ble 3.6. While the use of AES has increased and DES cipher has barely changed, triple DES, which is a more secure version of DES, has significantly decreased, from 9% in R12 down to about 1% in both R16 and T15. Surprisingly, we found that the RC4 cipher has made it to the top-3 used ciphers in both T15 and R16, even though it is considered insecure [65] and the security community has suggested removing it from cryptography libraries [90].

Assuming that the warning message might have been the root cause in the drastic decrease of ECB mode use, it is worth investigating in future research if adding similar messages for static IVs and encryption keys will have a similar effect. In addition, to supplement these warning messages, Google can provide "ready-to-use" code snippets to application developers in Android Studio IDE. This can eliminate the necessity for the developers to search online for code ex-

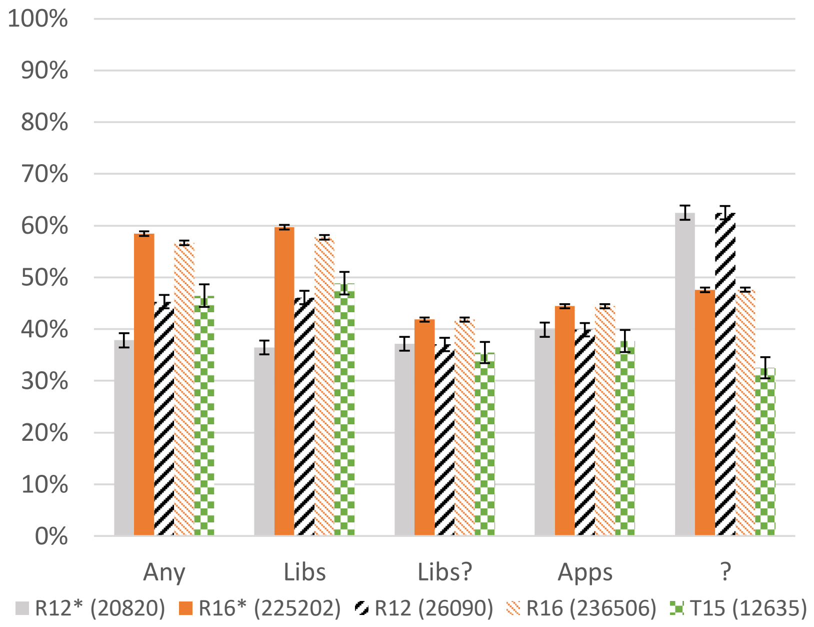

Figure 3.9: Ratio of call-sites that used static encryption key for a symmetric cipher. The total number of call-sites that set an encryption key for a symmetric cipher per dataset is provided in the legend.

amples that might potentially have implementation issues.

Password-based encryption

Since 2012, the rates of misuse of Password-based encryption (PBE) has overall decreased for both static salts (Rule 4) and number of iterations (Rule 5), as shown in Figures 3.10- 3.13. In particular, the ratio of APK files that provided static salts for PBKDF has decreased from 81% to 74%. The ratio of APK files that used less than 1,000 iterations decreased from 58% to 51%. The ratio of calls to relevant Crypto API that violate either Rule 4 or 5 has also decreased (as shown on Figures 3.11 and 3.13). Source attribution-based analysis showed that both libraries and applications have improved.

Comparison of the T15 and R16 datasets showed that, on average, 19% and

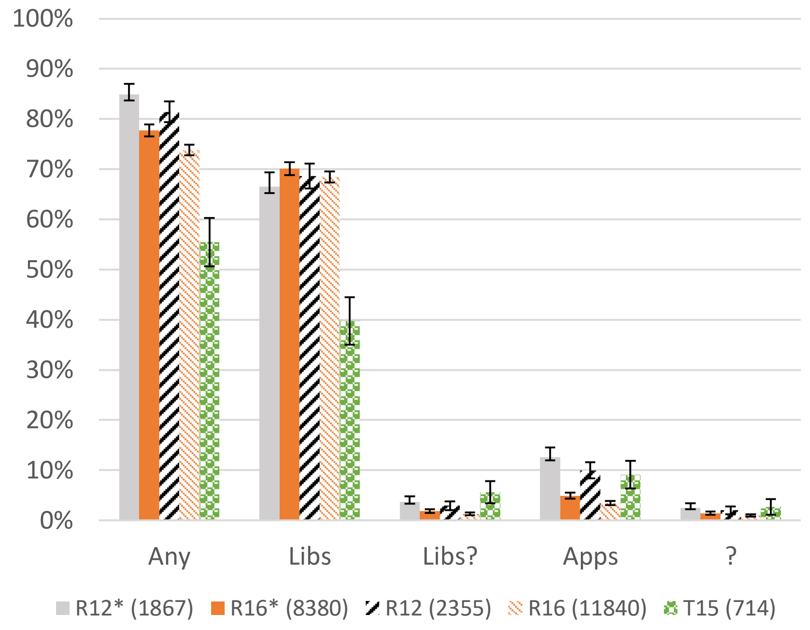

Figure 3.10: Ratio of APK files that violated Rule 4 - "Do not use static salt for PBKDF." The total number of APK files that used PBKDF per dataset is provided in the legend.

24% less APK files from T15 violated rules 4 and 5 respectively. Further, per source-category analysis revealed that this was mainly due to improvements in libraries used by the applications in the T15 dataset. Applications, however, violated rule 4 more frequently (10% more). We omit discussion of R12* and R16* sub-sets since our analysis did not reveal any statistically and practically significant results.

While these results suggest that there is a negative trend in the misuse of Crypto API, future research should focus on how to improve these results even further. For example, one might consider showing a message to application developers with implications of using static salts or fewer than 1,000 iterations. Such a warning message might include time estimates of how long a password guessing attack would take on today's hardware to go through the entire password space.

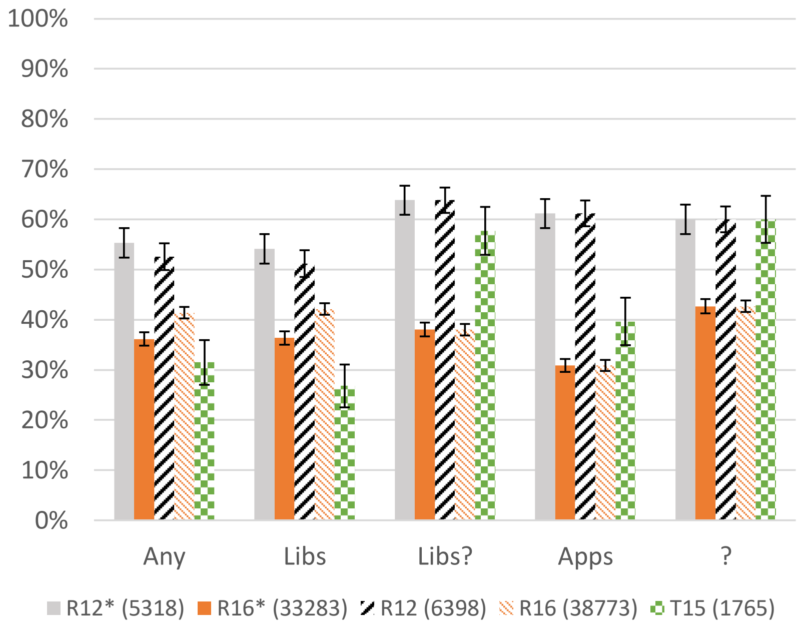

Figure 3.11: Ratio of call-sites that used static static salt for PBKDF. The total number of call-sites that provided a salt value for PBKDF per dataset is provided in the legend.

Random number generation

The use of static seed values for SecureRandom has significantly decreased since 2012 (Figures 3.14 and 3.15). In particular, while the ratio of APK files that provide static seed to SecureRandom has dropped from 73% to 67%, the ratio of relevant call-sites that use static seed value decreased to 43% from 69%. Although the ratio of APK files where libraries have introduced a violation of rule 6 has grown by 3%, call-site analysis has revealed that the libraries have significantly reduced the likelihood that a call to SecureRandom will provide a static value (from 72% in 2012 to 42% in 2016). This, again, shows that relying on the APK files ratio as the only way of measuring Crypto API misuse might convey an incorrect message.

Comparison of the T15 and R16 datasets revealed that libraries used by the top

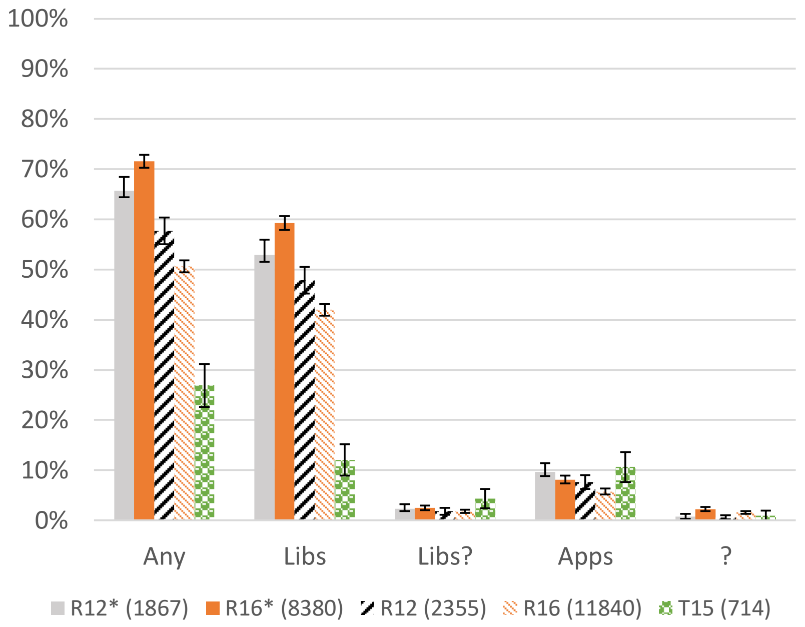

Figure 3.12: Ratio of APK files that violated Rule 5 - "Do not use less than 1,000 iterations for PBKDF." The total number of APK files that used PBKDF is provided in the legend.

applications had much lower impact on the APK files with violation of rule 6 and had a lower rate of calls to Crypto API that provided static values. I omit discussion of R12* and R16* sub-sets, since the analysis did not reveal any statistically or practically significant results.

Considering that the SecureRandom class can seed itself and that re-seeding does not decrease its entropy, we would suggest that this class always seeds itself, even if an application developer provides a static seed value for the constructor.

3.6.4 The impact of third-party libraries

Another important factor to consider for libraries is popularity. That is, a popular library with misuse will impact a significantly larger set of APK files. To understand how popularity impacts misuse rates we proceed with the following

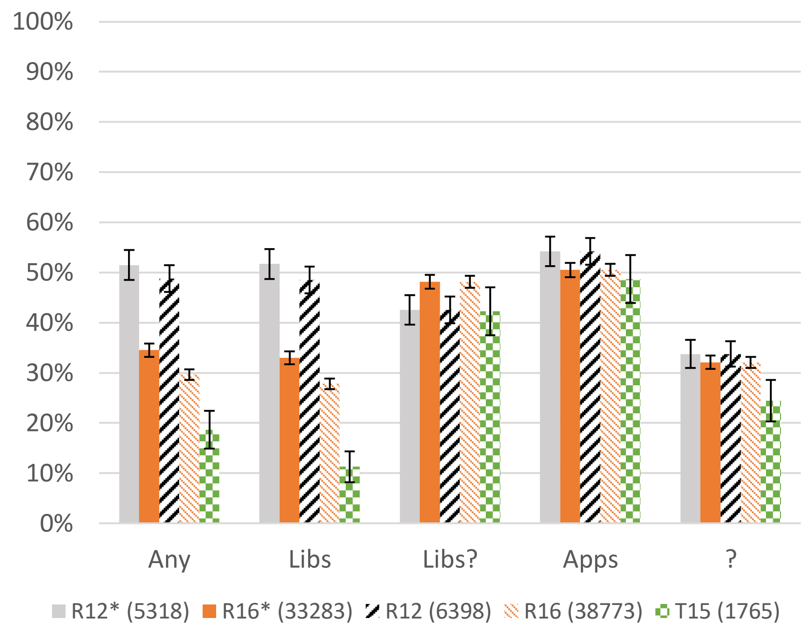

Figure 3.13: Ratio of call-sites that used 1,000 or less iterations for PBKDF. The total number of call-sites that used PBKDF per dataset is provided in the legend.

				Rank in dataset		
Company	Library	Package	Violated rules	R16	R12	T15
Google	Play SDK	com.google.android.gms.internal	2, 3	1	–	1
Apache	HTTP Auth	org.apache.http.impl.auth	1	2	3	5
InMobi	Advertising	com.inmobi.commons.core.utilities.a	–	3	–	2
Google	Advertising	com.google.ads.util	3	38	1	36
VPon	Advertising	com.vpon.adon.android.utils	1, 3	–	2	–

Table 3.7: Summary of Top-2 libraries from each dataset that made use of Crypto API. Empty values imply that the library was not found in the dataset.

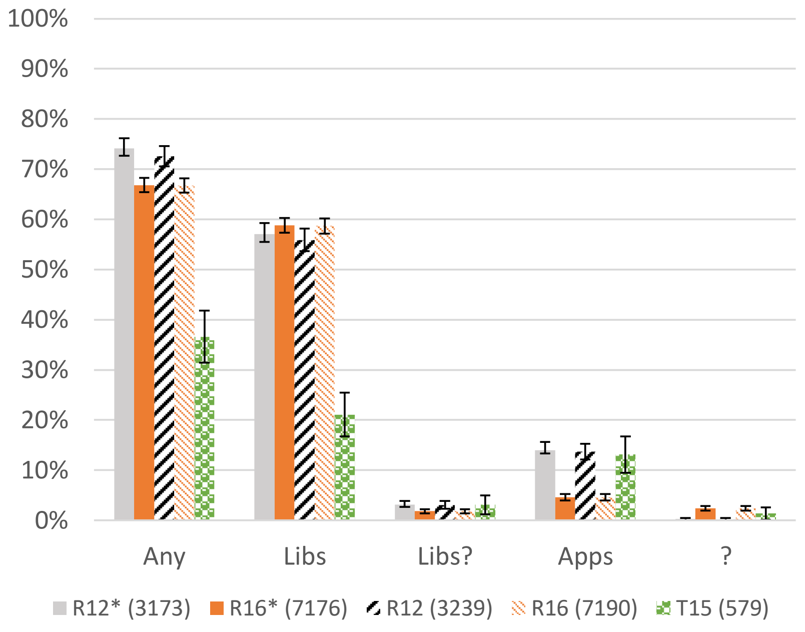

Figure 3.14: Ratio of APK files that violated Rule 6 - "Do not use static seed for SecureRandom." The total number of APK files that used SecureRandom per dataset is provided in the legend.

analysis. We measured the number of APK files that would be misuse-free if one began fixing misuses in libraries, starting with the most popular libraries. Figure 3.16 shows this impact for each dataset. In particular, by fixing the top most library in R16, 50,015 APK files would become misuse-free (or 56% of all APK files with misuses), and by fixing all libraries 79,207 APK files would be fixed (or 89.5% of APK files with misuses).

3.6.5 In-depth analysis of top libraries

Considering that libraries made 90% of all calls to cryptographic APIs and, if fixed, can potentially reduce the number of APK files with API misuse by a factor of 10 (see Figure 3.16), an in depth manual analysis on the top 2 libraries from each dataset was performed. After selecting the top 2 libraries from each dataset,

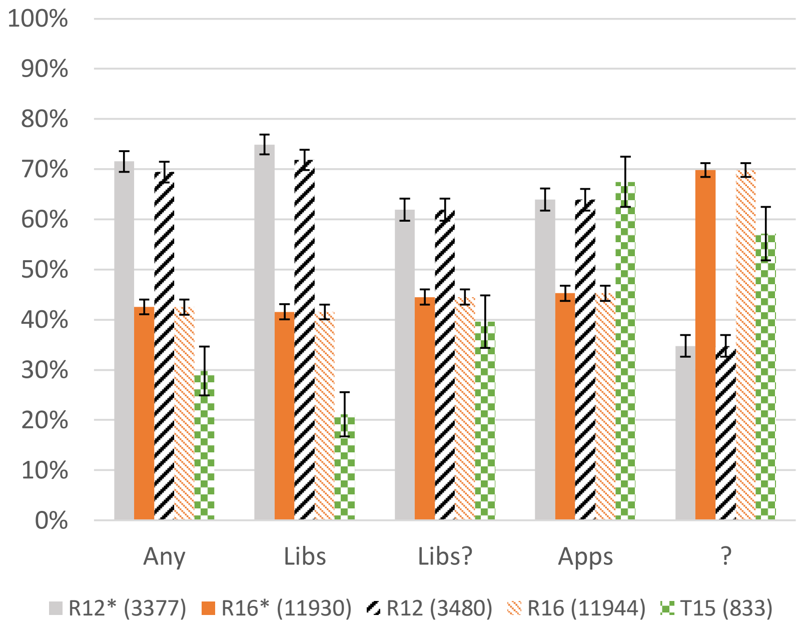

Figure 3.15: Ratio of call-sites that used static seed for SecureRandom. The total number of call-sites that seed SecureRandom per dataset is provided in the legend.

we found that one library was in the top 2 in T15 and R16, resulting in 5 libraries in total.

In what follows we provide the details of the results of our manual in-depth analysis of five libraries. This analysis was mainly focused on the reasons for the use of Crypto API and the security impact of the identified misuses, if any. Overall, the results of the analysis revealed that four out of five top libraries violated some of the rules. Three of them were identified as false positives, i.e., formal violation of Crypto API use which did not introduce a security vulnerability. Most of the analyzed misuse cases originated from obfuscation. In addition, three libraries implemented their own data encryption layer over HTTP or HTTPS protocols. This issue, however, can be trivially addressed by switching to HTTPS for all communications and dropping the libraries' implementations of data en-

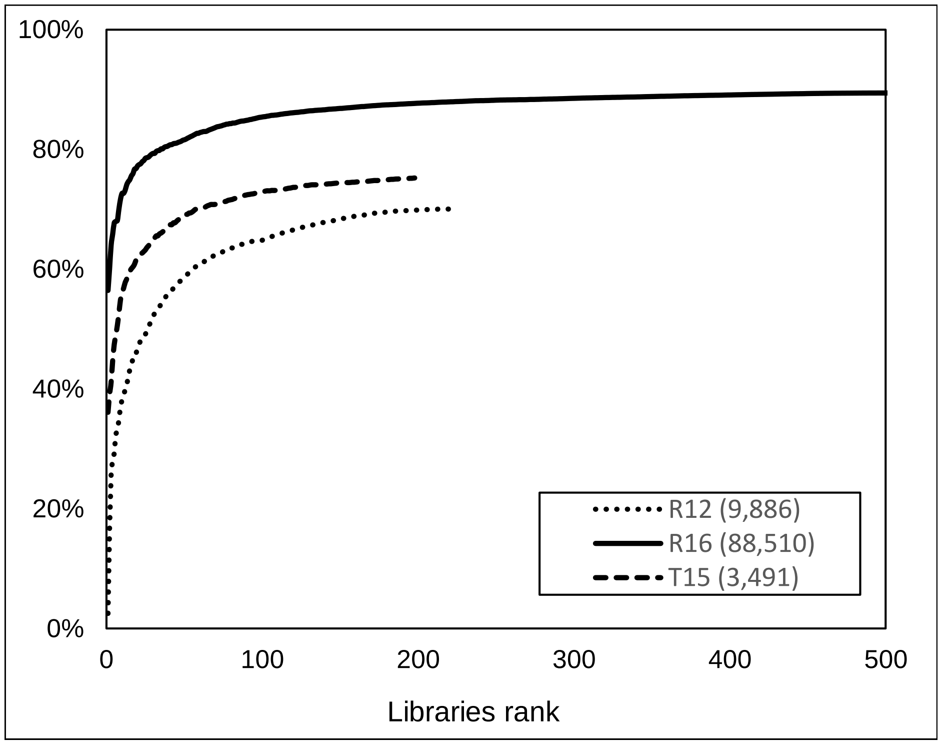

Figure 3.16: Proportion of APK files that would become Crypto API misuse-free depending on the number of fixed top ranked libraries. The legend shows the total number of applications that had at least one misuse in the corresponding dataset. We identified 222, 507 and 198 libraries with misuse in R12, R16 and T15 datasets, hence, the end of the corresponding lines.

cryption.

Google advertisement

This library was the top library in the R12 dataset, and made the top 40 in both the T15 and R16 datasets. This library provides advertisement services to applications. It makes use of data encryption API in the `AdUtil` class, located in `com.google.ads.util` package. The implementation uses a static key for encryption (i.e., violating Rule 3), which is hard-coded in the `AdUtil` class. The encryption function receives plaintext as a string and returns cipher text, also as a string. The encryption uses AES cipher in CBC mode with PKCS5Padding. The encryption function is later used to encrypt a string representation of a user's location, before being sent back to Google's servers. Considering that the communication happens over HTTPS protocol, the use of static key does not impact confidentiality in the presence of a network attacker. However, we would recommend fixing this to avoid exploitability if for some reason the HTTPS protocol was not available.

In addition, we found that in R16 and T15, this library has significantly changed. In particular, the newer version no longer used encryption. The structure of the library was significantly simplified as well. There were, however, several applications in both T15 and R16 datasets where the old version of the library was used and had `AdUtil` class with the same key and same misuse. Interestingly, these applications were updated relatively recently (2 – 3 months prior to the data collection of T15 and R16). This observation confirms findings from a recent report [39] which showed that application developers are slow to adopt new versions of libraries.

VPon advertisement

This library is also an advertisement library, present only in the R12 dataset. Similarly, it uses a cipher to encrypt and decrypt data. All identified call-sites were located in the `CryptUtils` class in the `com.vpon.android.utils` package. This library violates two rules: the use of ECB mode (rule 1) and a static encryption key (rule 3). `CryptUtils` class exposes two types of encryption functions, one that uses `javax.crypto.SealedObject` as an input for encryption, and one that accepts key and data as a string and returns a string as a result. The functions that work with `SealedObject` are used to encrypt requests that are sent back to the server and decrypt responses from it. This suggests that the static key is shared between the library and VPon's servers. The requests are sent over both HTTP and HTTPS. Unfortunately, we were unable to understand exactly what data are sent over what protocol. The second function, based on strings as input and output, is only used to decrypt obfuscated string literals. Decrypting string literals in Android applications is a common obfuscation approach, which implies that data confidentiality is not the primary objective. To summarize, this library violates two rules (use of ECB mode and use of static key) to communicate with the advertisement server and to obfuscate data.

Apache library

This library provides an ability for applications to communicate over HTTP and HTTPS protocols. It was the only library in the top 5 in all three datasets. The library uses Crypto API in many locations, but one specific call site, which used ECB mode, drew our attention. In particular, this library implements a suit of NT Lan Manager (NTLM) authentication protocols, which are commonly used to authenticate over HTTP(s). This protocol, by design, uses a DES cipher in ECB mode (i.e., violating rule 1), to implement challenge response validation. However, it only encrypts a single cipher block (i.e., 16 bytes). Considering such use, this misuse was classified as a false positive, since the encrypted content is random and fits into a single block.

Google Play SDK

This library provides services of the Google Play platform, such as In-App purchases or authentication with Google accounts. This library was the top most library in both T15 and R16, and was absent from the R12 dataset. It violated two rules, the use of static IV and static keys (rules 2 and 3). Interestingly, this library implemented only a decryption function that accepts a byte array as a key and a string as cipher-text. It outputs plain-text as a byte array. The key is Base64 encoded and hard-coded as a property in a static class, which is located in the *com.google.android.gms.internal* package. The same static class contains all the cipher-texts that get decrypted. All cipher texts are hard-coded in Base64 format. Further analysis revealed that one of the cipher texts is actually an encrypted DEX file, which upon decryption (about 3K in size) loaded into application space through Java Reflection API [29]. The remaining cipher texts are properties of the class (such as name of class and name of its fields or functions). This case falls into the obfuscation category, and was thus considered as a false positive.

InMobi advertisement

This library (the second-most popular library in the T15 dataset) provides in app advertisement capabilities to the applications. The call-sites to Cipher facilities were found in `InternlSDKUtil`, located in the `com.inmobi.commons.internal` package. This class uses an AES cipher in CBC mode with PKCS7 padding. It also uses an RSA cipher, to exchange a symmetric key with the server. We found that this library, similarly to VPon, uses encryption facilities to encrypt communications with their advertisement server. Interestingly, we saw the use of both HTTP and HTTPS protocols for communication to the same domain address, thus, it is unclear why the library developers had not switched all communications to HTTPS. This library generates an encryption key once, stores it in `SharedPreferences` and then reuses it on all subsequent communications. Formally, InMobi's implementation did not violate any of the evaluated rules of Crypto API use.

3.6.6 The impact of third-party libraries revisited

The results of the in-depth analysis of the top libraries revealed that the current approach used for identification of Crypto API misuses in APK files suffers from a significant ratio of false positives. We classify a misuse case as a false positive if the actual use of the Crypto API was not meant to provide integrity or confidentiality protection, i.e., not a concern for IND-CPA. For example, while Google Play SDK violated Rules 2 and 3 (did not use static IV for CBC mode and static encryption key), it did so for obfuscation purposes only. Another limitation of the current approach is that it misses certain edge cases, e.g., encryption of a single block of random data in ECB mode. Such cases significantly inflate misuse rates, and thus, convey a wrong state of actual misuse of cryptography in Android applications. Future research should focus on expanding BinSight's ability to classify if cryptographic APIs are used for obfuscation purposes.

3.7 Discussion and Future Work

The results of the analysis of more than 132K Android applications revealed that 9 in 10 calls to Crypto API originate from third party libraries. Libraries are also the main source of misuse cases, where 89.5% of the APK files collected in 2016 were flagged only due the libraries they used. By re-analyzing the dataset from the CryptoLint study we found that the authors have missed 249 out of 260 libraries in their dataset, which significantly contributed to over-counting in their results. In particular, 222 of the missed libraries were responsible for 70% of the flagged APK files in their dataset. These results suggest that future research sources on Crypto API (mis)use must use source attribution and analyze libraries and applications separately.

Our implementation of source attribution relies on package names. For this approach to work, the classes with calls to Crypto API should not be fully obfuscated, i.e., full renaming of class identifiers should not be used. Although our analysis revealed that only 2.5% of the classes were fully obfuscated among the applications in 2016, future research should focus on improving the ability of analysis tools to identify libraries and applications. One can achieve this objective by exploring methods for de-obfuscation proposed by Bichsel et al. [43] or Backes et al. [39].

To help developers choose secure libraries, the research community should also invest time in establishing a centralized repository to share identification data for libraries. Application developers would be able to consult such a repository to make informed decisions on which libraries to use, while library developers would also be able to respond to the discovered issues and explain misuse cases.

By using static analysis we inherited all limitations that come with such an approach. In particular, our sCFG is both an over and under estimation of the actual sCFG. One one hand, we overestimated sCFGs by including all detected edges. Such an approach might include edges that will never be executed during an application's run-time. On the other hand, in our analysis we did not include edges that were dynamically created. Such edges are usually created through Java Reflection API [29]. These limitations create risks for validity of the presented results. Future research should consider complementing BinSight with dynamic analysis capabilities and include analysis of reflection API into the sCFG construction process.

Although one cannot obfuscate calls to platform APIs, such as Crypto API, it is still possible to hide them through late binding. In particular, one can use Java Reflection API to side-load a binary that would make the actual call to Crypto API. This, as mentioned above, can be addressed by augmenting BinSight's analysis pipeline with the analysis of calls to Java Reflection API, which can be based on the same static analysis approach we used for Crypto API calls.

Even though the ratio of fully obfuscated classes in our datasets was negligible (2.5% in R16), understanding how obfuscated applications differ from non-obfuscated ones is still an important and interesting research question to investigate. Such low adoption of full obfuscation, on the other hand, allowed us to use trivial yet efficient and effective source attribution based on package names.

While looking into the top libraries we found that not all misuses of Crypto API necessarily have security implications. However, our analysis was exploratory in nature and does not provide precise assessment of the ratio of all identified Crypto API misuses that are false positives. Considering that the top library from R16 was responsible for 56% of flagged APK files and that it used Crypto API for obfuscation, we suspect that a significant portion of misuse cases are indeed false positive. This suggests that the current approach to analyzing Crypto API misuse is inadequate in several aspects.

First, as we showed, currently defined rules miss certain edge cases, which inflate the number of misuse cases. Second, without understanding the purpose of why the Crypto API is used in the first place, it is impossible to say when we actually encounter a misuse, given that Crypto API might be used for other reasons than confidentiality or integrity protection. Finally, without understanding the types of data that are being handled in the Crypto API calls, it is impossible to say to what extent a misuse might correspond to a security vulnerability.

To this end, our analysis showed that there are still plenty of open research questions that need to be addressed by future research. Given the identified limitations of the current approach to the analysis of Crypto API misuse, it is impossible to say if the current misuses actually results in security issues. In what follows, I discuss two research areas for future work.

3.7.1 Extending the Crypto API analysis

In our analysis we used the same six rules for secure use of Crypto API that the authors of the CryptoLint study defined. This set, however, is far from complete. First, future research should consider adding new rules, based on the recently reported attacks on the use of cryptographic APIs or modes of operation [95, 112]. Second, extending BinSight's coverage to asymmetric ciphers, similarly to what Shaui et al. [103] proposed, would provide the research community and application developers with a single tool that can provide an overall evaluation of how an application uses symmetric and asymmetric cryptography. Such an evaluation would have to be able to separate public keys from private ones, since having a static public key does not pose the same security threat as using static encryption keys for symmetric ciphers. Third, considering that recent research has showed that applications incorrectly validate SSL certificates [62], one can also include analysis for the implementation of secure protocols, such as TLS or SSL.

While identifying misuse cases is important, it is also highly important to understand what happens with data once it is encrypted. In addition, an insight into what kind of data is being encrypted would allow ranking the uncovered issues based on severity level. That is, if highly sensitive data are being handled by code that misuses Crypto API, one should try to fix that issue first, comparing to a benign misuse such as obfuscation.

One can achieve such a goal by analyzing program slices that gather data before submitting them for encryption. Furthermore, these slices should also include the flows where cipher-text is passed through an input/output (IO) API. Including IO flows is important for a proper assessment of security, since the attack surface is significantly different for network and on-host attackers. Finally, one can use dynamic analysis methods and data tainting to track all sensitive data records. For example, see [60] and [38].

Although one cannot obfuscate calls to platform APIs, such as Crypto API, it is still technically possible to hide such calls from static analysis. For instance, an application can use Java Reflection API to side-load a binary that would contain a class definition that makes the actual calls to Crypto API. To uncover such cases, future research should introduce analysis of the Reflection API into the BinSight pipeline. Once such call sites are found and the corresponding binaries are downloaded, one can feed these binaries to the existing BinSight pipeline, and proceed with the analysis as usual.

More importantly, our manual in-depth study of top libraries revealed that the identified misuse cases did not impact the actual security of the applications. In particular, while static linting was efficient in detecting Crypto API misuses, it failed to capture actual security implications, since most of the manually analyzed libraries used cryptography for other reasons than confidentiality or integrity protection. Reporting too many false positives would make it hard to convince application and library developers to change their coding practices. Future research should first focus on identifying these false positives. For instance, one could white-list known benign misuse cases and share them with the community. An example of this is the use of ECB mode in the Apache library for NTLM protocol implementation, which is secure since it encrypts a single block of random data. Another approach is to employ dynamic analysis and data tainting, in order to uncover the kinds of data involved. This would allow ranking of all misuses based on how sensitive the involved data are, and thus, would allow the research community to focus on what is most important.

3.7.2 How Crypto API Misuse Rates Have Changed

By using the original dataset from the CryptoLint study [57], we were able to (a) replicate the original study and (b) compare how misuse rates have changed between 2012 and 2016. The analysis of applications from the CryptoLint study showed comparable results, i.e., about 90% of Android applications contained at least one misuse of Crypto API. The analysis of applications collected in 2016 revealed similar results, i.e., around 90% also had at least one case of Crypto API misuse.

In contrast to the CryptoLint study itself, we analyzed calls from libraries and applications separately. Source attribution analysis revealed that 9 out of 10 calls to Crypto API originated from libraries. Such library domination makes the misuse rate measured by the CryptoLint study highly biased towards libraries. In particular, we showed that 507 libraries in the 2016 dataset were the only reason why 80.5% of APK files were flagged as misusing Crypto API. To provide better understanding of trends, we used the ratio of call-sites with mistakes to all call-sites as a complementary metric. While the ratio of APK files with Crypto API misuse provides insight into the impact of libraries and applications on the overall number of APK files with misuses, the call-site ratio provides intuitive probability for how often a call from libraries or applications will make a mistake.

Trend analysis showed that while both applications and libraries have improved in certain aspects, e.g., the use of ECB mode for symmetric ciphers, they have significantly worsened in others areas, such as in the use of static encryption keys. As there has been a decrease in the use of the ECB mode, a reason for this is possibly the introduction of a warning message in Google's Android Application Integrated Development Environment (IDE), which highlights the insecurity of the ECB mode. Interestingly, there are plenty of "suggestions" online for how to "fix" this warning message by replacing the ECB mode with CBC.

In addition, our analysis has revealed that RC4, a symmetric cipher with known vulnerabilities [65], has become the third most-used cipher among applications collected in 2016. Future research should focus on studying if warning messages, similar to the "insecure ECB mode" message, would change how developers use Crypto API. Further, recent research showed that application developers also need code samples to make sure they use API properly [36]. Thus, one approach would be to incorporate samples into the IDE as ready-to-use code snippets.

3.8 Conclusion

This chapter presents the results of a study on the misuse of cryptography APIs in Android applications. Although other researchers had previously measured the spread of misuse, this study differs in that it focused on source attribution, i.e., understanding whether a misuse originates from a library or an application. Such focus revealed that 9 out of 10 calls to Crypto APIs originate from libraries and that libraries are the major contributor of misuse, both in terms of APK files and the ratio of call-sites. In particular, third-party libraries were the only source of Crypto API misuses for 89.5% of flagged APK files.

While replicating the CryptoLint study and confirming its results, we showed that the study's analysis missed most of the libraries (249 out of 260). This led to over-counting, i.e., counting the misuse multiple times, since 70% of the identified APK files by the CryptoLint had misuses that originated only from 222 libraries. That is, 222 libraries were solely responsible for the flagging of 6932 APK files in the R12 dataset (out of 9886).

This chapter also provides insights on how Crypto API misuse rates have changed between 2012 and 2016. We found that the trends were mixed. While the overall ratio of APK files with misuse attributable to libraries had significantly worsened, libraries managed to improve in some areas (decreasing the use of ECB mode and static seed for SecureRandom). This chapter also demonstrates that using the ratio of APK files is biased towards libraries, especially the popular ones. To address this limitation we proposed to use the ratio of call-sites that made a mistake. We demonstrated that in certain cases the ratio of APK files provides a misleading message.

Finally, this study also provides insights into misuse cases by presenting results of a manual in-depth analysis of the top-2 libraries from each dataset. The results showed that while static linting is efficient at detecting Crypto API misuses, it fails to capture actual security implications. In particular, manual analysis revealed that the investigated libraries used cryptography for reasons other than confidentiality or integrity protection. Another observation was the edge case for Rule 1, i.e., encrypting a single cipher block of random data in ECB mode. This is why future research on Crypto API misuse should focus on improving the ability of analysis tools to identify use cases, their ability to detect edge cases, and, more importantly, their ability to identify types of data being processed by calls to Crypo API. Understanding the all this allows assessing the severity of a misuse cases, i.e., does it potentially leads to a data breach of sensitive data, or it is a mere functional false positive, e.g., obfuscation.

Chapter 4

Storing Encryption Keys on Wearable Devices

In this chapter we present the results of feasibility evaluation of a system that uses wearable devices to manage encryption key. The main intuition on why such a system might help address the issues presented in previous chapters is the proliferation of various wearable technologies (e.g., smart watches or fitness trackers).

4.1 Introduction

Public and private organizations see plenty of benefits in the adoption of smartphones or tablets for their businesses. For example, the *bring your own device* (BYOD) policy has become a norm [5]. While some businesses have less constraints on how well company's data needs to be protected, certain types of data are under stricter requirements. For example, health related data in the US is required to follow the Health Insurance Portability and Accountability Act of 1996 (HIPAA) [28] and protection of any personally identifiable information in Canada has to follow the Personal Information Protection and Electronic Documents Act (PIPEDA).

Adoption of the BYOD policy creates certain challenges in following HIPAA or PEPIDA. In particular, as shown in Chapter 2, 90% of users tend to rely on easy-to-guess unlocking secrets, which results in practical password guessing attacks and enables data decryption. Furthermore, 1 in 3 users do not lock their devices at all, which makes any data stored on the device immediately readable.

Although it is hard to measure how often an attacker actually tries to access confidential data on users' smartphones, there is some anecdotal evidence to consider. In the US alone every tenth smartphone owner has experienced theft at least once [4]. More than 30% of all street robberies involve smartphone theft [2]. Finally, 46% of companies from North America and Europe stated that theft of a data bearing device, such as a smartphone, was the key factor in the data breaches they experienced [5].

All of the above makes it challenging, if not impossible, for certain organizations to adopt smartphones and tablets in their businesses. In the following we design and evaluate a system, called Sidekick, that aims to address two issues. First, it aims to make unlocking secrets optional for data-at-rest security, by implementing wearable devices as key storage devices. Second, it aims to give full control to organizations over how their data are being encrypted, while still allowing the BYOD policy.

Evaluation results revealed that this proposal is practical from a technical point of view. That is, one can use Sidekick on all existing platforms and on devices that have Bluetooth Low Energy (BLE) stack. In addition, the system imposes negligible latency and power consumption overhead. There are still, however, plenty of open research questions, especially on usability of this proposal.

4.2 Threat Model

This section provides a description of threats, risks and attackers' capabilities considered during the design and development of the Sidekick system.

4.2.1 Threats and Risks

The Sidekick system was designed with a focus on smartphone loss and theft threats, or theft, for brevity. In an adversarial model we consider an opportunistic attacker, i.e., an attacker who's main objective is to profit from selling the device itself. As an additional source of revenue, such an attacker might attempt accessing data. Such an opportunistic attacker would only spend sufficient amount of time to find an unlocking secret which is *easy-to-guess*, as defined in Chapter 1. To achieve this, the attacker would use available tools for offline password guessing attack, e.g., HashCat [33] in order to find the unlocking secret. Attackers who aims accessing data as a primary objective are beyond our scope, since they can coerce smartphone owners to give up their unlocking secret through physical threats.

Once the device is in the hands of an attacker, there is a *risk* of confidential data disclosure. If an attacker gains access to confidential data, the owner of the data might suffer losses, such as reputation and/or financial damages. For example, the victim might have to pay fees for leaking private customer information. Our main focus is on organizations that adopted the BYOD policy but need control over protection of their data confidentiality. To achieve this, such organizations are willing to require their employees to carry a wearable device. This is why in our evaluation we focused only on technical aspects and not on usability of the proposal. The main reason for us to focus on the technical aspects, rather than usability or security, is three fold. First, we aimed to use the BLE communication stack which is available on all mobile platforms, but was deliberately compromised for energy efficiency [10]. Second, we evaluated a well-known and researched mutual authentication protocol on top of the BLE stack, with the Elliptic Curve Diffie Hellman protocol for establishing session keys. Third, considering that we envisioned Sidekick to be implemented as a background service on the wearable devices that are already in use, thus, having minimal impact on the user interactions with the wearable device.

4.2.2 Attack

Accessing confidential data that are not encrypted by either PBE or the application itself is trivial. If, however, PBE is enabled, the attacker would need to extract a *bit-by-bit* image of the internal storage first. This can be achieved through existing tools (e.g., [7]). The main reason for extracting the image first is to bypass the limitation on the number of failed unlocking attempts. This limitation is enforced on the OS level and often leads to complete data wipe-out. For example, in iOS a user can enable device wipe-out after 10 unsuccessful unlocking attempts. With the storage image in hands, the attacker mounts a password guessing attack in order to recover the key encryption key (KEK), which is used to protect the actual data encryption key (DEK).

The stolen device might use specialized hardware for PBE. For instance, iPhones and iPads use cryptographic chip for key derivation process. This chip has an embedded hardware key, which is used in key stretching. If such specialized hardware is used, the attacker will need to run certain computations on that hardware during the password guessing attack. Such attacks are called *on-device* brute-force attacks and, in general, are significantly slower than *off-device* attacks since massive parallelization becomes unavailable. Both on-device and off-device attacks are considered offline, as they bypass enforced limitations on the number of allowed failed unlocking attempts. In off-device scenario, the storage image is process off device, hence none of the restrictions can be applied. In on-device scenario, an attacker uses specialized hardware directly, which allows him to circumvent limitations enforced by Operating System or drivers. On-device attacks, however, are significantly harder to mount and require certain types of vulnerabilities in the booting sequence. Once the attacker recovers the KEK, it becomes trivial to recover the DEK and obtain access to encrypted data.

If an attacker needs to bypass data encryption implemented by an application, a binary file of the application needs to be first obtained and reversed engineered. For example, Android applications are distributed as APK files, which can be downloaded from the Google Play store with existing tools such as APKDownloader [25]. If a misuse of cryptographic API is found, the attacker then uses this knowledge to decrypt the data.

Because Sidekick relies on wearable devices, one must evaluate attacks on the used wireless communication stack and its implications on the overall security of the encrypted data. An attacker might attempt to obtain KEKs transferred over the BLE channel. If such an attack is successful, then one can decrypt DEK, and then decrypt encrypted data. An attacker might aim to corrupt the KEK during transmission over BLE, which, if successful, would make the corresponding DEK encryption cryptographically inaccessible. This capability might be exploited by ransomware – a malware that encrypts data and requires a payment to be made for the victim to get his or her data back.

4.2.3 General Assumptions

The design of Sidekick makes several general assumptions about the capabilities of an attacker. First of all, a perfect cryptography is assumed, i.e., attackers cannot differentiate the used cipher (AES) from a random permutation function. Considering that non-generic attacks have yet to be found for the AES cipher, this assumption is sound. It is also assumed that there are no security bugs in the implementation of the data encryption system that would introduce a shortcut for encryption and decryption (e.g., by using a hard-coded KEK or DEK, or by using a biased and predictable random number generator).

In addition, confidential data disclosures through a compromised OS kernel are not considered for the following reasons. First, having a secure OS kernel does not prevent the password guessing attack and attacks on misused cryptographic APIs. Second, unless a trusted secure platform is used, a compromised OS kernel renders any data encryption ineffective. That is, by virtue of controlling the OS kernel, the attacker can read and write any memory page, and thus can extract DEKs and KEKs from RAM directly.

Use of wireless communication stacks enables attackers to track users based on device addresses (i.e., Media Access Code). It is assumed, however, that tracking users is not one of the objectives of the attackers. Such an objective is not only unrelated to data-at-rest security, but can also be trivially addressed with existing methods and tools, e.g., by enabling the BLE privacy feature [10]. Finally, the ability to deny the existence of data is not considered either, since (a) it is complimentary to confidentiality protection, and (b) can be addressed using one of the available systems (e.g., [104]).

Finally, we assume that an attacker's main focus is the smartphone itself and the data stored on the device is secondary. That is, an attacker would attempt accessing data only if data protection is inadequate, e.g., the device is unlocked or unlocking secret can be guessed within time frame that the attacker considers reasonable. Considering that the main objective of the Sidekick is increasing the search space for the encryption keys, we focus the evaluation on costs associated with the use of such external device. We do not focus on security of the wearable device itself, both in terms of physical and operating systems, since we assume that the attacker will either not have access to this wearable device or a secure and tamper resistant device is being used.

4.2.4 Crypto-Attacker

The *crypto-attacker* aims to obtain confidential data by recovering the KEK through password search. It is assumed that the *crypto-attacker* has the following capabilities. First, he has physical access to the victim's smartphone. Second, the *crypto-attacker* knows the design of the data encryption system and knows how the KEK is generated. Third, he can obtain a *bit-by-bit* image of the internal storage on the stolen smartphone, which allows bypassing the file system access control. Techniques that allow acquiring the raw storage image have been widely discussed in the last few years [104, 114]. Fourth, if the stolen device uses special hardware for data encryption, e.g., crypto-chip in iPhones, he knows how to mount an *on-device* password guessing attack. Finally, it is assumed that the *crypto-attacker* has limited time during the attack, and that the attacker is not capable of mounting a successful guessing attack on a pseudo-randomly generated 128-bit KEK.

4.2.5 Network-Attacker

A *network-attacker* might have several objectives. First, he might be interested in data-at-rest stored on the smartphone. In this case the attacker plans to steal the smartphone later, but first aims to obtain all KEKs transmitted over BLE in order to eliminate the necessity to later perform a KEK search. Second, the *network-attacker* might be interested in corrupting KEKs to cryptographically lock the user's valuable data. The attacker can then use his knowledge of how he corrupted the KEK and request a ransom payment from the victim.

To compromise the wireless channel, the *network-attacker* can use one of two approaches. First, the attacker can focus on the wireless messages themselves by exploiting insecure protocols and gaining the ability to recover or corrupt KEKs. In particular, we assume that the *network-attacker* is able to exploit vulnerabilities reported in BLE stack thus far [96–98]. In particular, these attacks showed that recovering the established pairing keys for the BLE stack is practical. This gives the attacker ability to decrypt and modify any message transmitted over the BLE stack.

Second, if the attacker has control over an application on the victim's smartphone with access to the Bluetooth stack, he can communicate with the wearable device and retrieve required KEKs before stealing the device. This type of an attack is called a *misbonding* attack [86]. In order to mount a misbonding attack an attacker needs to obtain access to the Bluetooth stack, by requesting *android.permission.BLUETOOTH* permission for an Android application. This allows the application to communicate with all Bluetooth devices that are paired and connected, including the wearable device used for KEKs storage. From the wearable device perspective, all applications that communicate with it have the same identity, that of the paired smartphone. That is why wearable device with Sidekick must be able to identify each application in order to enforce access control on the stored KEKs.

4.3 Sidekick Design

In this section we present the design of the Sidekick system. We begin with a high-level overview of the system, then proceed with a discussion of used countermeasures to mitigate attacks by a *network-attacker*. We conclude with a security analysis of the proposal in the presence of *network* and *crypto* attackers.

4.3.1 High Level Overview

The Sidekick system relies on a wearable device in the encryption keys management task. This allows the decoupling of user authentication and data encryption by making the dependency on the unlocking secret optional in the Data Encryption Key derivation process. Instead, Sidekick generates all KEKs randomly and stores them on an external device. Such systems that separate storage have been already proposed and deployed for personal computers and laptops, e.g., TrueCrypt and BitLocker [12, 14]. With the evaluation of the Sidekick system we aim to evaluate if it is practically feasible to use an insecure BLE stack while achieving both (a) keeping the latency and power consumption reasonable, and (b) mitigating the misbonding attack by introducing mutual authentication between applications on the smartphone side and Sidekick service on the wearable device side.

The key technical difference in the design of the Sidekick system from all existing proposals is the process of fetching the KEK from a wireless wearable device, shown in Figure 4.1. Thus, our technical evaluation of Sidekick is focused on the performance of the KEK fetching process.

There are several reasons to choose a wireless stack and a wearable device over existing physical connections. Physically attached external devices, such as memory cards or USB flash drives, are not well suited for smartphones, since not all modern smartphones have a USB port or allow using external memory cards. Physically attached external devices also require constant user attention, since forgetting to unplug the external device from the smartphone destroys all security properties that a data encryption system provides. The proliferation of

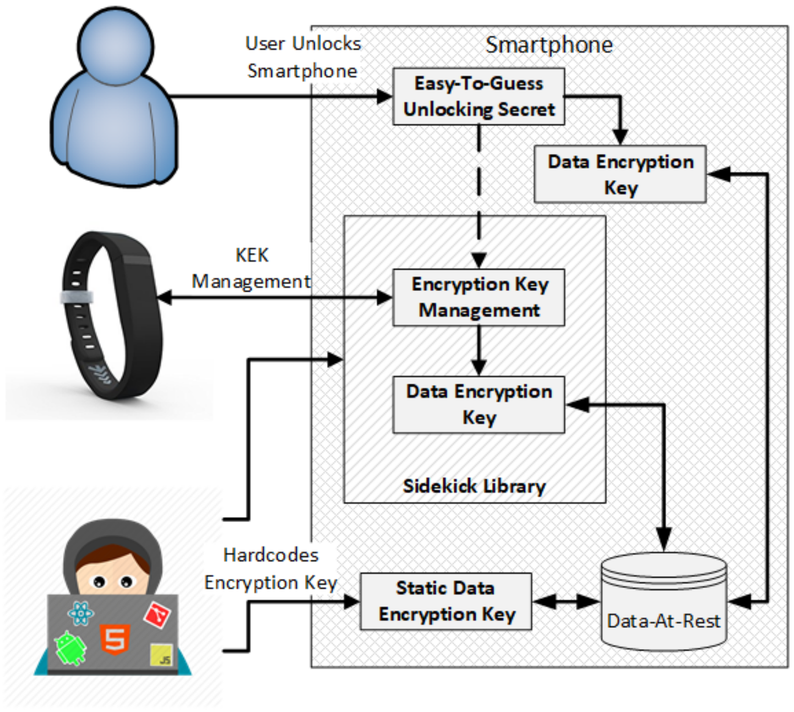

Figure 4.1: In currently deployed systems, a user needs to provide an unlocking secret to unlock his or her device. The unlocking secret, most probably, is an easy-to-guess one. That secret is then used to derive a Data Encryption Key (DEK), which is then used for data encryption/decryption. When application developers need to encrypt data in smartphones, they usually use a static data encryption key, i.e., hard code it into their application, and then also roll out their own implementation of the data encryption. Sidekick addresses both issues by randomly generating key encryption keys (KEK) and then storing them on a wearable device. Sidekick makes data encryption independent from the unlocking secret, by mainly relying on KEKs while making the use of unlocking secrets optional (showed as a dashed line). It also provides a simpler API to application developers so that they do not need to roll out their own implementation of data encryption and a encryption key management system.

wearable devices, such as smartwatches and fitness trackers, suggest that there will be plenty of options for users to choose from for storing KEKs.

Sidekick uses the BLE stack for the following reasons, which are mostly prac-

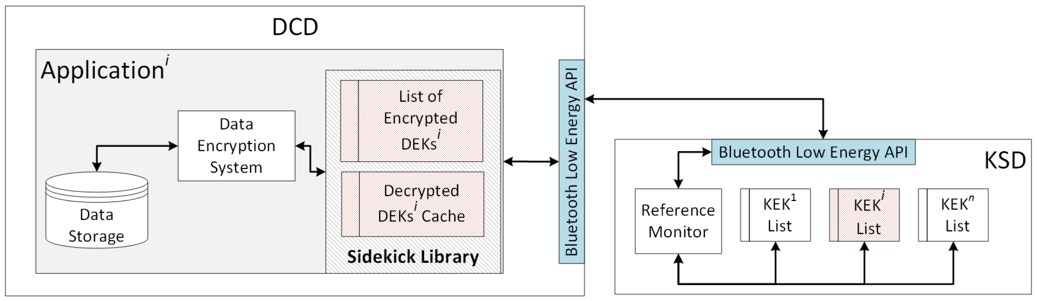

Figure 4.2: High-level design of the Sidekick System. A data containing device (DCD) runs applications that link the Sidekick library. The library takes care of all communications with the KSD, e.g., storing or retrieving a KEK. Once a required KEK is retrieved, a corresponding DEK is decrypted and stored in the Decrypted DEKs Cache by the Sidekick Library. The DEK is then passed to the Data Encryption System in order to encrypt/decrypt data. Each application has a separate KEK List. The Reference Monitor on the KSD mitigates a misbonding attack by ensuring that each application has access only to its own KEK List.

tical. First, BLE hardware and BLE APIs are available on all platforms today, while other Personal Area Network (PAN) stacks, notably Near Field Communication (NFC), have limited support. Second, the use of BLE in wearable devices is energy efficient, since recent research has showed that a BLE-based System-On-Chip (SoC) can work for months off a single coin-cell battery [6, 69]. Finally, most of the released wearable devices already rely on the BLE stack for their communications with smartphones (e.g., Nike+ [16]). Sidekick was prototyped on the CC2540 SoC developed by Texas Instruments. Although several other, more capable BLE-enabled SoCs were available at the time, CC2540 was chosen since it was the most popular and least capable BLE-enabled SoC at the time of the experiments [13].

The overall design of Sidekick is shown in Figure 4.2. A smartphone, that stores sensitive data is called a data containing device (DCD). A wearable device that stores KEKs is called a key storing device (KSD). When a users tries to access sensitive data on the DCD, Sidekick fetches the corresponding KEK from the KSD and recovers the DEK. Once the data is decrypted with the recovered DEK, a user can read or modify that data on the DCD.

There are four requests that Sidekick can send to a KSD, namely *get*, *store*, *update*, and *delete* on a KEK. Each of the four requests (Req) has a corresponding response (Resp) from the KSD. For instance, when the DCD needs to store a new KEK on the KSD, it sends a ***StoreReq*** to the KSD with the new KEK in the payload. Once the KSD has processed that request, it responds with a ***StoreResp***, which contains a KEK_{ID}, a unique KEK identifier, in the payload. Later, when the DCD needs to fetch that KEK, it needs to provide the KEK_{ID} it received in the ***StoreResp*** payload.

4.3.2 Securing Communications over BLE

While the main objective of Sidekick is to mitigate attacks by the*crypto-attacker*, we have to address the risks that arise from the use of the BLE stack. In particular, a *network-attacker* can exploit one of the previously reported vulnerabilities [96–98][1]. The existence of such attacks is not surprising, since the security of BLE was compromised on purpose to make BLE-based SoCs power-efficient [10]. In addition, Sidekick needs to overcome the limitations in access control in mobile operating systems for access to the BLE stack. In particular, Sidekick needs to control access to each KEK so that two different applications cannot fetch each other's KEKs.

Transport Layer Security. To ensure integrity and confidentiality protection for all of Sidekick's communications over the BLE (to protect against the *network-attacker* who aims to corrupt KEKs), we used the Counter with CBC-MAC mode (CCM) [109], since (a) all BLE-enabled SoCs have this implemented in hardware [10], and (b) it has been proven to be secure [75].

Mutual Authentication. On top of the transport layer, which is the Attribute Protocol (or ATT) in the BLE stack, a well-known and studied mutual authentication protocol based on a shared secret was used [63]. The mutual authentication

[1]As of this writing all these attacks are still practical.

protocol was used for two reasons: (a) to establish a pairing key during the initial pairing between the KSD and an application on the DCD, and (b) since each application established its own key, the pairing key was used to authenticate applications to properly enforce access control to KEKs.

Pairing KSD and DCD. For mutual authentication to work, one should first establish a bootstrapping shared secret. Unfortunately, wearable devices are often limited in their Input/Output capabilities. For instance, the CC2540 SoC only has two LEDs in the default circuit design, which is why a blinking LED (BLED) approach, proposed by Saxena et al. [100], was adopted in Sidekick to establish the initial shared secret. In BLED, one device generates a pseudo random key and shares it by controlling how an LED blinks, while the other, significantly more capable device, uses a camera and converts a blinking LED into a bit stream. The security of a secret established in this way is only important while a new, significantly stronger shared key is being established through a key establishment protocol, such as Elliptic Curve Diffie-Hellman (ECDH). The results of benchmarking experiments with the Samsung S3 and the iPhone 4S revealed that both devices can reliably handle a bit-stream of 3 bits/s. To make the wait time shorter for end users, while maintaining a sufficient level of security, Sidekick uses BLED to established a 32-bit secret in about 10 seconds, which corresponds, approximately, to a six-digit PIN-code. Limited amount of RAM and processing power available in modern wearable devices make it impossible to use the original DH protocol [51]. To overcome this limitation, Sidekick uses the ECDH protocol, based on the P128 curve.

Other Considerations. Sidekick needs to mitigate *Replay* and *Retry* attacks as well. To mitigate *Replay* attacks Sidekick uses a *Nonce* in each message and verifies that *Nonce* on the recipient side. To mitigate *Retry* attacks, Sidekick uses a monotonically increasing number as a *Nonce*, which allows a recipient to detect retried old messages by comparing the message number of the message in question with the last message sent/received thus far.

4.4 System Evaluation

4.4.1 Experimental Setup.

Sidekick was evaluated with an iPhone 4S and a Samsung S3 smartphone as the DCDs, and a CC2540 SoC [11] as the KSD. The KSD was implemented in C as a firmware for the CC2540. The DCD side was implemented in native languages for the given platforms, i.e., Objective C for iOS, Java for Android. Sidekick had low memory requirements on the KSD side (20Kb or ROM and 4Kb or RAM), and negligible impact on smartphones. The evaluation of the system was conducted under the assumption that KEKs and DEKs are 256-bit long.

4.4.2 Latency

The *overall latency* is defined as the time span from the moment an application on the DCD submits a request to the moment the application receives a corresponding response from the KSD. The *overall latency* consists of two parts (a) *communication latency* – the time spent on completing a request over BLE, and (b) *computation latency* – the time spent on all required calculations, such as message encryption and decryption. Considering that the benchmarking experiments revealed that *communication latency* is in the order of several magnitudes higher, the *computation latency* is omitted from further discussion.

There are several fundamental factors in BLE that impact *communication latency*. First, maximum transmission unit size at the ATT layer, referred to as MTU_ATT in the BLE specification [10], limits the number of bytes one can fit into a single ATT packet. Second, the BLE specification defines a connection interval (CInterval), i.e., a time window for a single packet. Finally, BLE defines a connection interval latency (CLatency), which defines the maximum number of allowed connection intervals without a message before the connection is considered closed. Note that, to maintain connection, the BLE stack sends so called *empty-PDU*s in unused connection intervals. CLatency allows devices to skip sending *empty-PDU* to conserve energy, by allowing the wearable device to stay longer in the most power-efficient modes.

The results of sniffing on the BLE connection setup process revealed that the CC2540 SoC and Android OS smartphones supported up to 23 bytes in each packet, while the iPhone 4S allowed up to 132 bytes in a single packet. The default values for CInterval were $30ms$ and $48.5ms$ for Android and iOS respectively. Finally, both platforms used zero as the default value for CLatency, i.e., skipping connection intervals was not allowed by default.

Table 4.1 provides a summary of latency for each of the four supported requests. These results suggest that with the default configuration of BLE stack in both iOS and Android, Sidekick introduces less than a second delay in the DEK recovery process. Considering that this delay is significantly smaller than the process of unlocking a smartphone with a secret, as was shown in Chapter 2, one can hide this delay by fetching KEK during the unlocking process.

Table 4.1: Overall Latency for each four request/response message pairs for the default values for CInterval and CLatency.

Req/Resp ,	Overall Latency, ms	
(Payload Length, bytes)	Android OS	iOS
Store (32/4)	873	540
Retrieve (4/32)	873	540
Update (32/0)	873	540
Delete (4/0)	776	480

4.4.3 Power Consumption

Smartphones. Power consumption is a crucial property of a system that is meant to be used in smartphones. Draining too much power would make the proposal less appealing to end-users. The results of the laboratory experiment on power consumption revealed that retrieving a single 256-bit KEK consumes approximately $5.7\mu Ah$ on smartphones, or about 0.0004% their battery capacity. That is, if a user unlocks their smartphones 100 times a day and a KEK is fetched during each unlock, that would consume 0.04% of battery capacity. Considering that users tend to charge their smartphones on a daily basis, one can completely ignore the power consumption overheads that Sidekick introduces.

Wearables. The CC2540 SoC is based on the 8-bit 8051 CPU, which provides great flexibility in power consumption through four power modes: Active Mode, and Modes 1-3. In theory, CC2540 can run for 9 hours in Active Mode and up to 30 years in Mode 3 on a single CR2032 battery [69]. Of course, in practice the battery lifespan depends on specific firmware. Power-consumption experiments revealed that retrieving a single KEK consumes approximately $0.12\mu Ah$ of battery capacity on CC2540, which corresponds approximately to 0.00005% of CR2032 capacity. That is, a single CR2032 battery allows the KSD to receive and process about 2 million requests from the DCD.

To assess battery lifespan more precisely, one needs to define a daily workload, i.e., the number of requests sent per day to the KSD. For example, if we consider that the KSD stores a single KEK for the entire smartphone, then we can set the expected daily workload to approximately 100 requests a day [110]. On the other hand, a banking or a business application might store multiple keys on the KSD. Four workloads were used, with 1, 10, 100, and 1,000 requests a day, to cover various Sidekick usage scenarios.

In addition, several various values for CInterval were used to show how a slight increase in the overall latency impacts battery lifespan. In particular, the following four values {0, 15, 32, 48} for CInterval parameter were evaluated. The values correspond to 540, 1000, 1500, 2000 ms of the overall latency for a KEK fetching request.

Table 4.2: CR2032 battery life in days, depending on the acceptable overall latency for a request and on the number of requests per day.

	Maximum Latency, ms			
Requests per Day	540	1,000	1,500	2,000
1	14	217	443	652
10	14	217	442	651
100	14	215	434	633
1,000	14	197	366	496

The results shown in Table 4.2 suggest that the default configuration for the BLE channel (CInterval = 30*ms* and CLatency = 0) for the CC2540-based KSD allows it to run for only two weeks. If, however, we increase maximum allowed latency up to 2,000 ms, then KSD can last more than 600 days on a single battery, assuming a workload of 100 requests per day.

4.4.4 Session Key Renewal

Secrecy of the session keys is crucial for the overall security of Sidekick. Power consumption experiments on wearable devices revealed that establishing a single 128-bit session key consumes 0.044% (or 0.1 mAh) of battery. That is, a single CR2032 battery allows at most 2,272 session keys to be established with ECDH-P128. This is why it is also important to factor in the energy consumption of session key establishment and renewals in the battery lifespan assessment for wearable devices.

For demonstration purposes let us consider the following example:[2] a user unlocks his smartphone 100 times a day, and, in parallel, Sidekick makes a request to the KSD to fetch the KEK. Considering that the fastest unlocking secret [82] requires about 2 seconds, we would also assume that having a two second overall latency is acceptable. With these parameters set, the results shown in Table 4.2 suggest that the CR2032 battery will last for 633 days. Now, if we renew the session key twice a day, that would correspond to 406 days of battery lifespan, i.e., 36% of battery capacity will be spent on key establishment.

4.4.5 Summary

Overall, the results of the benchmark, latency and power consumption experiments revealed that the use of wearable devices to decouple user authentication and data encryption in smartphones is a practical proposal. In particular, the latency of retrieving a KEK can be completely hidden behind the smartphone un-

[2]This example closely matches requirements of the current data protection systems in iOS and Android OS.

locking process. The evaluated proposal had insignificant impact on battery life and the current implementation allows a KSD device to run for more than a year on a single coin-cell battery.

4.5 Related Work

There are two complimentary ways of improving security of password-based encryption. One can increase the cost of password guessing attacks either by increasing the costs of the key derivation process or by nudging users to choose passwords that are harder to guess.

The research community has also proposed other KDFs such as ones that focus on substantially increasing the cost of each step for attackers. For example, Boyen [111] developed a halting key derivation function, which forces an attacker to perform substantial amounts of additional computations for each guess, and thus significantly increases the overall cost of the attack, while keeping users' costs relatively low. Other proposals, e.g., [70, 104], have suggested to increase the number of PBKDF iterations, to keep up with the recent improvements in computational capabilities of modern processors. This, however, is a never-ending arms race.

Others proposals focused on designing and evaluating novel authentication methods for smartphones (e.g., [49]) that are usable, yet, secure. While the presented evaluation results suggest that the proposed authentication methods are usable and resilient against specific attacks, such as shoulder-surfing attacks, users still choose easy-to-guess unlocking secrets, which are comparable with 4-digit PIN-codes in complexity. In contrast, Sidekick is a KDF agnostic system, that is, it eliminates the dependency of data encryption security on the unlocking secret. Although, one can tangle a randomly generated KEK with an unlocking secret, that would still provide at least the same amount of entropy that the randomly generated KEK does.

Finally, researchers proposed protection techniques for data-at-rest on mobile devices. For instance, DOrazio et al. [53] proposed an approach to conceal or delete unprotected data in iPhones. In particular, the proposal generates a new key **C** and then uses this key to encrypt the per-file key (stored in file's metadata block on iOS). This solution renders the file cryptographically unreadable without **C** key. If the user does not store that key anywhere, then the data practically becomes cryptographically deleted. Their solution, however, requires substantial expertise from end-users in order to setup the concealment. For instance, users are required to jailbreak their devices. Sidekick, on the other hand, was designed to work transparently with minimal user involvement. If a user can use a fitness tracker, she should be able to use Sidekick as well.

4.6 Discussion and Future Work

Overall, the evaluation results suggest that the Sidekick system is practical in added latency and power consumption. Low power consumption makes the proposal attractive due to minimal maintenance efforts, i.e., frequency of battery replacement or not needing to change how often one charges a smartphone. Considering that the current implementation does not require any changes in the mobile OS, one can use the proposal right away. In fact, the implementation of secure communication from Sidekick is being used by the company FusionPipe[3].

The Sidekick system was designed for organizations that want to take control over protecting the confidentiality of the data in their employees' smartphones. Hence we have focused our evaluation on primary technical aspects – low cost and low maintenance solutions. There are, however, plenty of open research questions and challenges remaining.

First, having yet another device might push users away and it is not clear if users are embracing existing wearable devices or keeping them closeby at all times. Thus far it is not clear if a system design based on a wearable device offers better usability than strict requirements to pick a hard-to-guess unlocking secret. Second, if such a system is adopted by end users for themselves, one should extend Sidekick with a usable fall-back mechanism that would allow user recovery in case

[3]http://www.fusionpipe.com/

the wearable device becomes unavailable. Finally, physical and system security of the wearable devices might need to be properly evaluated, especially when such protection is required.

While technical evaluation of the Sidekick system showed promising results, there are still plenty of remaining research questions. For instance, it is unclear if users are willing to trust such a system in the first place, considering that a user will not be able to access data if something happens with the wearable device. Also, it is still unknown how often users actually use wearable devices and wear them. Finally, the usability of all user interactions involved in the configuration process of Sidekick is not studied, and thus, might potentially hinder adoption.

Fortunately, there are ways to make the proposal simpler, at least for application developers. Existing cryptographic libraries and APIs are often hard to use [36], so to address issues related to the misuse of cryptographic APIs, such as the use of ECB mode, static IVs and static encryption keys, future research should consider integrating Sidekick with the recent EXT4 file system driver [27]. This file system has recently[4] received an update adding support of per-file encryption [80]. To encrypt a file in EXT4, an application developer or a user would need to declare the file as encrypted. This can be achieved by either setting file attributes through system calls or by using the `e4crypto` command line tool[5]. The driver takes care of properly using cryptographic primitives, thus simplifying data encryption for developers.

While declaring a file as encrypted in EXT4, one needs to provide a so-called "encryption policy". This policy can define the process of how file-specific key encryption key (KEK) is derived in the Linux kernel. The actual data encryption key is then wrapped with the KEK and stored in the file's *inode*. Before proceeding with the requested IO operation, the driver attempts to reconstruct the KEK, and, subsequently, recover the DEK. As of this writing, the EXT4 driver supports

[4]The set of patches that enabled encryption in the EXT4 driver were released in 2015 with Linux kernel 4.1. The author of this thesis worked on its implementation during his internship at Google in 2015.

[5]http://man7.org/linux/man-pages/man8/e4crypt.8.html

kernel keyrings[6] as the only source of data chucks for the KEK derivation process. As a result, both the driver and the `e4crypto` tool need to be extended to support Sidekick's wearables.

Integration of the Sidekick system with EXT4 driver would provide several advantages. First, application developers will not be required to implement the actual implementation of the data encryption, since it will be provided by the EXT4 driver. Second, security research community would be able to focus on smaller code based, i.e., the file system driver. Finally, considering that drivers often have direct access to hardware, drivers are able to take advantage of hardware acceleration, such as AES-NI[7].

4.7 Conclusion

This chapter presented the design and evaluation of Sidekick – a system that decouples data encryption and user authentication in smartphones by using wearable devices for encryption key management. The evaluation results showed that the proposed system is effective from a technical standpoint; it can be deployed across all mobile platforms right away, works on new and old devices, and does not noticeably increase power consumption and latency. However, there are still plenty of open research questions remaining, especially with regards to usability.

Sidekick was evaluated on iOS and Android platforms and a commonly used BLE-enabled SoC (CC2540 [11]). The results of experiments on latency and power consumption revealed that with the session key renewed twice a day, a Key Storage Device based on CC2540 can work more than 400 days on a single coin-cell battery.

[6]https://lwn.net/Articles/639523/

[7]AES-NI is hardware implementation of AES by Intel CPUs.

Chapter 5

Discussion and Conclusion

In this thesis we looked at data-at-rest security from various angles. In particular, we studied how end users use and misuse smartphone locking systems. We then studied how application developers employ Crypto API in their applications and what kind of mistakes they make. We then looked into practical the use of wearable devices is for sensitive data protection in smartphones. In what follows we summarize the implications of the results presented in this dissertation.

A set of users studies presented in this work provide a deeper insight into how and why users use (or do not) smartphone locking systems (see Chapter 2 for more details). The results of these studies revealed that there is a gap in the design of smartphone locking systems and users needs. First, the authentication methods are still far from being able to provide both usability and security, since almost all subjects chose an easy-to-guess unlocking secret. Second, 20% of the subjects found it cumbersome to unlock their device when all they needed was weather forecast. Although recent updates to both iOS and Android allowed certain features of the phone to be used while in locked state (e.g., Camera), there are still plenty of other services and applications that need to move in the same accessibility domain, e.g., games, anonymous browser. Finally, results of the studies revealed that users are targeted by attackers from their social circle, thus, home might not be as safe as previously thought.

These results grant further research into usable authentication methods and more flexible application access control systems for smartphones. While new authentication methods should be usable they also need to be secure in the environment where shoulder surfing attacks are highly probable.

Another vector of attack on users' smartphones is through smartphone applications that victims use. For example, if an application that stores and handles sensitive data misuses Crypto API, data might be decrypted while in transit or while stored, if a bit-by-bit image is obtained. To understand the extent of exposure we conducted an analysis study on 132K of Android applications. The results of the analysis (see Chapter 3 for more details) revealed that the current data protection systems in smartphones are inadequate in the presence of an attacker with physical access to the device. In both cases, if a full-disk encryption is used, or if an application itself encrypts data, then an attacker has higher chances of being able to recover the encryption key, and thus, decrypt the data.

An attackers' ability to recover an encryption key arises mainly from two limitations. First, as we showed in Chapter 2, users tend to choose unlocking secrets that are easy-to-guess within minutes [34, 37, 104]. Second, more than half of calls to symmetric cipher API by developers relies on a static encryption key, which can be trivially extracted with such tools as Dex2Jar and Java Decompiler [19, 21].

Addressing both of these limitations is challenging. On the user side, one needs to understand how to improve complexity of unlocking secrets without degrading usability. Unfortunately, while new technologies have promised to address certain limitations, they appear to be not as effective [45]. On the applications developers side of the issue, not much progress has been made. To complicate the matters, the problem of Crypto API misuse is still far from being fully understood by research community.

While our studies present interesting results, e.g., almost all calls to Crypto API are made by a small set of libraries, there is still gap in our understand of what does that mean from practical perspective. Furthermore, manual in-depth analysis showed that the existing analysis approach based on static analysis suffers from a new type of false positive – functional false positive. A functional false positive is a case where a misuse has no security implications. For example, in the recent years binary code obfuscation became a standard, and while it encrypts certain bits of binary with a static key (i.e., formally misuses Crypto API) the goal is not to protect confidentiality, but rather make the reverse engineering task more complex. Future research should not only expand the set of the rules for Crypto API misuse, but should also look into techniques that can be used to detect functional false positives.

While application developers used static encryption keys, end-users struggled with creating hard-to-guess passwords. Both of these issues rendered sensitive data-at-rest insecure. With the design of Sidekick system (presented in Chapter 4) we aimed to evaluate if using wearable devices for encryption key management is vital from practical point of view. We define practical as (a) a solution that would be unnoticeable from latency point of view, and (b) would not drain significant amount of power from both a smartphone and wearable. We envisioned Sidekick to be deployed as a service into already used wearable devices, such as FitBit or Apple Watch. While results of the experiments showed that wearable devices could provide a practical solution, it is still unknown if smartphone users would like to adopt such an approach.

Bibliography

[1] Smart phone thefts rose to 3.1 million last year, consumer reports finds. http://www.consumerreports.org/cro/news/2014/04/smart-phone-thefts-rose-to-3-1-million-last-year/index.htm. last accessed April 22, 2015. → pages 1

[2] Announcement of new initiatives to combat smartphone and data theft. https://www.fcc.gov/document/announcement-new-initiatives-combat-smartphone-and-data-theft. last accessed May 12, 2015. → pages 1, 101

[3] Find My iPhone. http://itunes.apple.com/ca/app/find-my-iphone/id376101648. last accessed February 4, 2012. → pages 25

[4] Phone Theft In America. https://www.lookout.com/resources/reports/phone-theft-in-america. last accessed April 22, 2015. → pages 1, 101

[5] 2014 Cost of Data Breach Study. http://www-935.ibm.com/services/us/en/it-services/security-services/cost-of-data-breach/. last accessed April 22, 2015. → pages 1, 100, 101

[6] Bluetooth SIG Analyst Digest Q4 2012. https://www.bluetooth.org/library/userfiles/file/Bluetooth_Analyst2012. → pages 109

[7] Elcomsoft iOS Forensic Toolkit. http://www.elcomsoft.com/eift.html, 2012. Accessed February 15, 2013. → pages 103

[8] Dashboards | Android Developers. http://developer.android.com/about/dashboards/index.html, 2012. Accessed July 18, 2012. → pages 31

[9] Symantec Smartphone Honey Stick Project. http://www.symantec.com/en/ca/about/news/resources/press_kits/detail.jsp?pkid=symantec-smartphone-honey-stick-project, 2012. → pages 1, 50

[10] Specification Of The Bluetooth System 4.1. https://www.bluetooth.org/docman/handlers/downloaddoc.ashx?doc_id=229737, 2013. Accessed Feb 08, 2013. → pages 102, 105, 110, 112

[11] Bluetooth Low Energy System on Chip CC2540. http://www.ti.com/product/CC2540, 2013. Accessed February 15, 2013. → pages 112, 119

[12] TrueCrypt - Free Open-Source On-The-Fly Disk Encryption Software for Windows 7/Vista/XP, Mac OS X and Linux. http://www.truecrypt.org/, 2013. Accessed February 15, 2013. Version 7.1a. → pages 107

[13] Bluetooth Smart CC2541 SensorTag. http://www.ti.com/ww/en/wireless_connectivity/sensortag/index.shtml?DCMP=sensortag&HQS=sensortag-bn, 2014. → pages 109

[14] BitLocker Drive Encryption. http://windows.microsoft.com/en-CA/windows7/products/features/bitlocker, 2014. Accessed February 26, 2014. → pages 107

[15] 2 Billion Consumers Worldwide to Get Smart(phones) by 2016, 2014. URL http://www.emarketer.com/Article/2-Billion-Consumers-Worldwide-Smartphones-by-2016/1011694. → pages 1

[16] Nike+. https://secure-nikeplus.nike.com/plus/, 2014. Accessed February 26, 2014. → pages 109

[17] Apktool - A tool for reverse engineering Android apk files (Version 2.01). http://ibotpeaches.github.io/Apktool/, July 2015. last accessed June 29, 2015. → pages 4, 63

[18] The Legion of the Bouncy Castle. https://www.bouncycastle.org/, July 2015. last accessed June 29, 2015. → pages 59

[19] Tools to work with Android .dex and Java .class files. https://github.com/pxb1988/dex2jar, July 2015. last accessed June 29, 2015. → pages 59, 121

[20] Dalvik bytecode. https://source.android.com/devices/tech/dalvik/dalvik-bytecode.html, July 2015. last accessed October 29, 2016. → pages 63

[21] Yet another fast Java Decompiler. http://jd.benow.ca/, July 2015. last accessed June 29, 2015. → pages 121

[22] Spongy Castle - repackage of Bouncy Castle for Android. https://rtyley.github.io/spongycastle/, July 2015. last accessed June 29, 2015. → pages 59

[23] Reverse engineering, Malware and goodware analysis of Android applications ... and more (ninja !)). https://github.com/androguard/androguard, November 2016. last accessed November 16, 2016. → pages 53, 63

[24] Android now has 1.4 billion 30-day active users globally. https://techcrunch.com/2015/09/29/android-now-has-1-4bn-30-day-active-devices-globally/, 2016. Accessed August 2, 2016. → pages 1

[25] Direct apk downloader. Direct APK Downloader, 2017. URL https://androidappsapk.co/apkdownloader/. → pages 61, 104

[26] CUDA | GeForce, 2017. URL https://www.geforce.com/hardware/technology/cuda. → pages 2

[27] Ext4 file system. https://www.kernel.org/doc/Documentation/filesystems/ext4.txt, 2017. → pages 118

[28] Summary of the HIPAA Security Rule, 2017. URL https://www.hhs.gov/hipaa/for-professionals/security/laws-regulations/index.html. → pages 100

[29] java.lang.reflect (Java Platform SE 8) - Provides classes and interfaces for obtaining reflective information about classes and objects., 2017. URL https://docs.oracle.com/javase/8/docs/api/java/lang/reflect/package-summary.html. → pages 64, 92, 94

[30] Java Cryptography Architecture Oracle Providers Documentation for Java Platform Standard Edition 7. http://docs.oracle.com/javase/7/docs/technotes/guides/security/SunProviders.html, May 2017. last accessed May 15, 2017. → pages 59

[31] SecureRandom (Java Platform SE 7), 2017. URL https://docs.oracle.com/javase/7/docs/api/java/security/SecureRandom.html. → pages 58

[32] Soot - a framework for analyzing and transforming java and android applications, 2017. URL https://sable.github.io/soot/. → pages 9

[33] hashcat - Advanced Password Recovery. https://hashcat.net/hashcat/, 07 2018. → pages 2, 102

[34] D. Abalenkovs, P. Bondarenko, V. K. Pathapati, A. Nordbø, D. Piatkivskyi, J. E. Rekdal, and P. B. Ruthven. Mobile Forensics: Comparison of extraction and analyzing methods of iOS and Android. *Master Thesis, Gjøvik University College*, 2012. → pages 46, 121

[35] Y. Acar, M. Backes, S. Fahl, D. Kim, M. L. Mazurek, and C. Stransky. You get where you're looking for: The impact of information sources on code security. In *Security and Privacy (SP), 2016 IEEE Symposium on*, pages 289–305. IEEE, 2016. → pages 54

[36] Y. Acar, M. Backes, S. Fahl, S. Garfinkel, D. Kim, M. L. Mazurek, and C. Stransky. Comparing the usability of cryptographic APIs. In *Proceedings of the 38th IEEE Symposium on Security and Privacy*, 2017. → pages 52, 98, 118

[37] Apple. iOS Security, 8.1 and up. http://www.apple.com/business/docs/iOS_Security_Guide.pdf, 2014. Accessed April 26, 2015. → pages 2, 46, 57, 121

[38] S. Arzt, S. Rasthofer, C. Fritz, E. Bodden, A. Bartel, J. Klein, Y. Le Traon, D. Octeau, and P. McDaniel. Flowdroid: Precise context, flow, field,

object-sensitive and lifecycle-aware taint analysis for android apps. In *Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '14, pages 259–269, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-2784-8. doi:10.1145/2594291.2594299. URL http://doi.acm.org/10.1145/2594291.2594299. → pages 96

[39] M. Backes, S. Bugiel, and E. Derr. Reliable third-party library detection in android and its security applications. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, pages 356–367, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-4139-4. doi:10.1145/2976749.2978333. URL http://doi.acm.org/10.1145/2976749.2978333. → pages 3, 54, 68, 70, 90, 93

[40] M. Bellare and P. Rogaway. Introduction to modern cryptography, 2017. URL https://cseweb.ucsd.edu/~mihir/cse207/classnotes.html. → pages 56

[41] M. Bellare, T. Ristenpart, and S. Tessaro. Multi-instance security and its application to password-based cryptography. In *Advances in Cryptology–CRYPTO 2012*, pages 312–329. Springer, 2012. → pages 57

[42] N. Ben-Asher, N. Kirschnick, H. Sieger, J. Meyer, A. Ben-Oved, and S. Möller. On the need for different security methods on mobile phones. In *Proceedings of the 13th International Conference on Human Computer Interaction with Mobile Devices and Services*, MobileHCI '11, pages 465–473, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0541-9. doi:10.1145/2037373.2037442. URL http://doi.acm.org/10.1145/2037373.2037442. → pages 42

[43] B. Bichsel, V. Raychev, P. Tsankov, and M. Vechev. Statistical deobfuscation of android applications. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, pages 343–355, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-4139-4. doi:10.1145/2976749.2978422. URL http://doi.acm.org/10.1145/2976749.2978422. → pages 66, 93

[44] D. W. Binkley and K. B. Gallagher. Program slicing. *Advances in Computers*, 43:1–50, 1996. → pages 65

[45] I. Cherapau, I. Muslukhov, N. Asanka, and K. Beznosov. On the impact of touch id on iphone passcodes. In *Proceedings of the Symposium on Usable Privacy and Security*, SOUPS '15, page 20, July 22-24 2015. → pages 3, 6, 43, 49, 121

[46] E. Chin, A. P. Felt, V. Sekar, and D. Wagner. Measuring user confidence in smartphone security and privacy. In *Proceedings of the Eighth Symposium on Usable Privacy and Security*, SOUPS '12, pages 1:1–1:16, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1532-6. doi:10.1145/2335356.2335358. URL http://doi.acm.org/10.1145/2335356.2335358. → pages 41

[47] R. Cytron, J. Ferrante, B. K. Rosen, M. N. Wegman, and F. K. Zadeck. Efficiently computing static single assignment form and the control dependence graph. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 13(4):451–490, 1991. → pages 64

[48] A. De Luca, M. Langheinrich, and H. Hussmann. Towards understanding atm security: a field study of real world atm use. In *Proceedings of the Sixth Symposium on Usable Privacy and Security*, SOUPS '10, pages 16:1–16:10, New York, NY, USA, 2010. ACM. ISBN 978-1-4503-0264-7. doi:http://doi.acm.org/10.1145/1837110.1837131. URL http://doi.acm.org/10.1145/1837110.1837131. → pages 40

[49] A. De Luca, A. Hang, F. Brudy, C. Lindner, and H. Hussmann. Touch me once and i know it's you!: implicit authentication based on touch screen patterns. In *Proceedings of the 2012 ACM annual conference on Human Factors in Computing Systems*, CHI '12, pages 987–996, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1015-4. doi:10.1145/2208516.2208544. → pages 49, 116

[50] A. De Luca, M. Harbach, E. von Zezschwitz, M.-E. Maurer, B. E. Slawik, H. Hussmann, and M. Smith. Now you see me, now you don't: Protecting smartphone authentication from shoulder surfers. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '14, pages 2937–2946, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-2473-1. doi:10.1145/2556288.2557097. URL http://doi.acm.org/10.1145/2556288.2557097. → pages 49

[51] W. Diffie and M. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22:644–654, 1976. URL http://citeseer.nj.nec.com/diffie76new.html. → pages 111

[52] D. Dolev, C. Dwork, and M. Naor. Non-malleable cryptography. In *SIAM Journal on Computing*. Citeseer, 1998. → pages 3, 53

[53] C. DOrazio, A. Ariffin, and K. K. R. Choo. ios anti-forensics: How can we securely conceal, delete and insert data? In *System Sciences (HICSS), 2014 47th Hawaii International Conference on*, pages 4838–4847, Jan 2014. doi:10.1109/HICSS.2014.594. → pages 116

[54] T. Dorflinger, A. Voth, J. Kramer, and R. Fromm. "My Smartphone is a Safe!" - The User's Point of View Regarding Novel Authentication Methods and Gradual Security Levels on Smartphones. In *SECRYPT 2010 - Proceedings of the International Conference on Security and Cryptography, Athens, Greece, July 26-28, 2010, SECRYPT is part of ICETE - The International Joint Conference on e-Business and Telecommunications*, pages 155–164. SciTePress, 2010. → pages 42

[55] P. Dunphy, A. P. Heiner, and N. Asokan. A closer look at recognition-based graphical passwords on mobile devices. In *Proceedings of the Sixth Symposium on Usable Privacy and Security*, SOUPS '10, pages 3:1–3:12, New York, NY, USA, 2010. ACM. ISBN 978-1-4503-0264-7. doi:10.1145/1837110.1837114. URL http://doi.acm.org/10.1145/1837110.1837114. → pages 40

[56] M. Dürmuth, T. Güneysu, M. Kasper, C. Paar, T. Yalcin, and R. Zimmermann. *Evaluation of Standardized Password-Based Key Derivation against Parallel Processing Platforms*, pages 716–733. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012. ISBN 978-3-642-33167-1. doi:10.1007/978-3-642-33167-1_41. URL https://doi.org/10.1007/978-3-642-33167-1_41. → pages 2, 6

[57] M. Egele, D. Brumley, Y. Fratantonio, and C. Kruegel. An empirical study of cryptographic misuse in android applications. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 73–84. ACM, 2013. → pages xv, 3, 4, 7, 8, 9, 53, 55, 60, 61, 62, 64, 69, 74, 97

[58] S. Egelman, A. Sotirakopoulos, I. Muslukhov, K. Beznosov, and C. Herley. Does my password go up to eleven?: The impact of password meters on password selection. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '13, pages 2379–2388, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-1899-0. doi:10.1145/2470654.2481329. URL http://doi.acm.org/10.1145/2470654.2481329. → pages 49

[59] M. Eiband, M. Khamis, E. von Zezschwitz, H. Hussmann, and F. Alt. Understanding shoulder surfing in the wild: Stories from users and observers. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, CHI '17, pages 4254–4265, New York, NY, USA, 2017. ACM. ISBN 978-1-4503-4655-9. doi:10.1145/3025453.3025636. URL http://doi.acm.org/10.1145/3025453.3025636. → pages 49

[60] W. Enck, P. Gilbert, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N. Sheth. Taintdroid: an information-flow tracking system for realtime privacy monitoring on smartphones. In *Proceedings of the 9th USENIX conference on Operating systems design and implementation*, OSDI'10, pages 1–6, Berkeley, CA, USA, 2010. USENIX Association. URL http://dl.acm.org/citation.cfm?id=1924943.1924971. → pages 96

[61] W. Enck, D. Octeau, P. McDaniel, and S. Chaudhuri. A study of android application security. In *USENIX security symposium*, volume 2, page 2, 2011. → pages 61

[62] S. Fahl, M. Harbach, T. Muders, L. Baumgärtner, B. Freisleben, and M. Smith. Why eve and mallory love android: An analysis of android ssl (in) security. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 50–61. ACM, 2012. → pages 53, 95

[63] N. Ferguson, B. Schneier, and T. Kohno. *Cryptography Engineering: Design Principles and Practical Applications*. John Wiley & Sons, 2011. → pages 110

[64] F. Fischer, K. Böttinger, H. Xiao, C. Stransky, Y. Acar, M. Backes, and S. Fahl. Stack overflow considered harmful? the impact of copy&paste on android application security. In *Security and Privacy (SP), 2017 IEEE Symposium on*, pages 121–136. IEEE, 2017. → pages 54

[65] S. R. Fluhrer, I. Mantin, and A. Shamir. Weaknesses in the key scheduling algorithm of rc4. In *Revised Papers from the 8th Annual International Workshop on Selected Areas in Cryptography*, SAC '01, pages 1–24, London, UK, UK, 2001. Springer-Verlag. ISBN 3-540-43066-0. URL http://dl.acm.org/citation.cfm?id=646557.694759. → pages 81, 97

[66] A. Forget, S. Chiasson, P. C. van Oorschot, and R. Biddle. Improving text passwords through persuasion. In *Proceedings of the 4th Symposium on Usable Privacy and Security*, SOUPS '08, pages 1–12, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-276-4. doi:10.1145/1408664.1408666. URL http://doi.acm.org/10.1145/1408664.1408666. → pages 49

[67] B. G. Glaser. *Theoretical sensitivity : advances in the methodology of grounded theory*. Sociology Press, Mill Valley, CA, 1978. → pages 16

[68] K. Glen. iOS 5.1 Reaches 61% Adoption in Just 15 Days. http://www.mactrast.com/2012/03/ios-5-1-reaches-61-adoption-in-just-15-days/, 2012. Accessed July 18, 2012. → pages 31

[69] C. Gomez, J. Oller, and J. Paradells. Overview and evaluation of bluetooth low energy: An emerging low-power wireless technology. *Sensors*, 12(9): 11734–11753, 2012. ISSN 1424-8220. doi:10.3390/s120911734. → pages 109, 114

[70] P. A. Grassi, E. M. Newton, R. A. Perlner, A. R. Regenscheid, W. E. Burr, J. P. Richer, N. B. Lefkovitz, J. M. Danker, Y.-Y. Choong, K. Greene, et al. Digital identity guidelines: Authentication and lifecycle management. Technical report, 2017. URL https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-63b.pdf. → pages 58, 116

[71] E. Hayashi, J. Hong, and N. Christin. Security through a different kind of obscurity: evaluating distortion in graphical authentication schemes. In *Proceedings of the 2011 annual conference on Human factors in computing systems*, CHI '11, pages 2055–2064, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0228-9. doi:http://doi.acm.org/10.1145/1978942.1979242. URL http://doi.acm.org/10.1145/1978942.1979242. → pages 40

[72] E. Hayashi, O. Riva, K. Strauss, A. J. B. Brush, and S. Schechter. Goldilocks and the two mobile devices: going beyond all-or-nothing access to a device's applications. In *Proceedings of the Eighth Symposium on Usable Privacy and Security*, SOUPS '12, pages 2:1–2:11, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1532-6. doi:10.1145/2335356.2335359. URL http://doi.acm.org/10.1145/2335356.2335359. → pages 7, 42, 46, 48

[73] E. Hayashi, S. Das, S. Amini, J. Hong, and I. Oakley. Casa: Context-aware scalable authentication. In *Proceedings of the Ninth Symposium on Usable Privacy and Security*, SOUPS '13, pages 3:1–3:10, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-2319-2. doi:10.1145/2501604.2501607. URL http://doi.acm.org/10.1145/2501604.2501607. → pages 47

[74] I. Ion, N. Sachdeva, P. Kumaraguru, and S. Capkun. Home is Safer than the Clould! Privacy Concerns for Consumer Cloud Storage. In *Proceedings of Symposium on Usable Privacy and Security*, pages 1–20, Pittsburgh, PA, USA, July 2011. URL http://cups.cs.cmu.edu/soups/2011/proceedings/a13_Sachdeva.pdf. → pages 26

[75] J. Jonsson. On the security of CTR+ CBC-MAC. In *selected Areas in Cryptography*, pages 76–93. Springer, 2003. → pages 110

[76] B. Kaliski. Pkcs #5: Password-based cryptography specification version 2.0, 2000. → pages 57

[77] S. Komanduri, R. Shay, P. G. Kelley, M. L. Mazurek, L. Bauer, N. Christin, L. F. Cranor, and S. Egelman. Of passwords and people: Measuring the effect of password-composition policies. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '11, pages 2595–2604, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0228-9. doi:10.1145/1978942.1979321. URL http://doi.acm.org/10.1145/1978942.1979321. → pages 49

[78] D. Lazar, H. Chen, X. Wang, and N. Zeldovich. Why does cryptographic software fail?: A case study and open problems. In *Proceedings of 5th Asia-Pacific Workshop on Systems*, APSys '14, pages 7:1–7:7, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-3024-4.

doi:10.1145/2637166.2637237. URL
http://doi.acm.org/10.1145/2637166.2637237. → pages 52

[79] H. Lu and Y. Li. Gesture on: Enabling always-on touch gestures for fast mobile access from the device standby mode. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, CHI '15, pages 3355–3364, New York, NY, USA, 2015. ACM. ISBN 978-1-4503-3145-6. doi:10.1145/2702123.2702610. URL http://doi.acm.org/10.1145/2702123.2702610. → pages 47

[80] LWN.net. Ext4 encryption [lwn.net]. https://lwn.net/Articles/639427/, 2017. → pages 118

[81] Z. Ma, H. Wang, Y. Guo, and X. Chen. Libradar: Fast and accurate detection of third-party libraries in android apps. In *Proceedings of the 38th International Conference on Software Engineering Companion*, pages 653–656. ACM, 2016. → pages 66

[82] A. Mahfouz, I. Muslukhov, and K. Beznosov. Android users in the wild: Their authentication and usage behavior. *Pervasive and Mobile Computing*, 32:50–61, 2016. → pages 44, 115

[83] D. Marques, I. Muslukhov, T. Guerreiro, L. Carriço, and K. Beznosov. Snooping on mobile phones: Prevalence and trends. In *Twelfth Symposium on Usable Privacy and Security (SOUPS 2016)*, Denver, CO, June 2016. USENIX Association. URL https://www.usenix.org/conference/soups2016/technical-sessions/presentation/marques. → pages 6, 44, 46

[84] S. McNeeley. Sensitive issues in surveys: Reducing refusals while increasing reliability and quality of responses to sensitive survey items. In *Handbook of survey methodology for the social sciences*, pages 377–396. Springer, 2012. → pages 44

[85] I. Muslukhov, Y. Boshmaf, C. Kuo, J. Lester, and K. Beznosov. Know your enemy: the risk of unauthorized access in smartphones by insiders. In *Proceedings of the 15th international conference on Human-computer interaction with mobile devices and services*, MobileHCI '13, pages 271–280, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-2273-7. doi:10.1145/2493190.2493223. → pages 2

[86] M. Naveed, X. Zhou, S. Demetriou, X. Wang, and C. Gunter. Inside job: Understanding and mitigating the threat of external device mis-bonding on android. In *Proceedings of the 21th Annual Network and Distributed System Security Symposium*, NDSS Symposium'14, San Diego, CA, USA, 2014. → pages 106

[87] P. Oechslin. Making a faster cryptanalytic time-memory trade-off. In *Crypto*, volume 2729, pages 617–630. Springer, 2003. → pages 53, 57

[88] G. Paolacci, J. Chandler, and P. G. Ipeirotis. Running experiments on amazon mechanical turk. *Judgment and Decision Making*, 5(5):411–419, 2010. URL http://papers.ssrn.com/sol3/papers.cfm?abstract_id=1626226. → pages 32, 41

[89] C. Percival, Tarsnap, and S. Josefsson. The scrypt Password-Based Key Derivation Function draft-josefsson-scrypt-kdf-02, Aug 2015. URL https://tools.ietf.org/html/draft-josefsson-scrypt-kdf-02. → pages 57

[90] A. Popov. RFC7465 - prohibiting RC4 cipher suites. https://tools.ietf.org/html/rfc7465, Feb 2015. URL https://tools.ietf.org/html/rfc7465. → pages 81

[91] A. D. Portal. Encryption | android developers, May 2015. URL https://source.android.com/devices/tech/security/encryption/index.html. → pages 2

[92] D. Preuveneers and W. Joosen. Smartauth: Dynamic context fingerprinting for continuous user authentication. In *Proceedings of the 30th Annual ACM Symposium on Applied Computing*, SAC '15, pages 2185–2191, New York, NY, USA, 2015. ACM. ISBN 978-1-4503-3196-8. doi:10.1145/2695664.2695908. URL http://doi.acm.org/10.1145/2695664.2695908. → pages 47

[93] R. Raguram, A. M. White, D. Goswami, F. Monrose, and J.-M. Frahm. iSpy: automatic reconstruction of typed input from compromising reflections. In *Proceedings of the 18th ACM conference on Computer and communications security*, CCS '11, pages 527–536, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0948-6. doi:10.1145/2046707.2046769. URL http://doi.acm.org/10.1145/2046707.2046769. → pages 3, 46, 50

[94] O. Riva, C. Qin, K. Strauss, and D. Lymberopoulos. Progressive authentication: deciding when to authenticate on mobile phones. In *Proceedings of the 21st USENIX Security Symposium*, Usenix Security '12, pages 301–316, Berkeley, CA, USA, 2012. USENIX Association. → pages 7, 42, 46, 48

[95] J. Rizzo and T. Duong. Practical Padding Oracle Attacks. In *Proceedings of the 4th USENIX Conference on Offensive Technologies*, WOOT'10, pages 1–8, Berkeley, CA, USA, 2010. USENIX Association. URL http://dl.acm.org/citation.cfm?id=1925004.1925008. → pages 95

[96] T. Rosa. Bypassing passkey authentication in bluetooth low energy. Cryptology ePrint Archive, Report 2013/309, 2013. http://eprint.iacr.org/. → pages 106, 110

[97] M. Ryan. Bluetooth: With low energy comes low security. In *Presented as part of the 7th USENIX Workshop on Offensive Technologies*, Berkeley, CA, 2013. USENIX. URL https://www.usenix.org/conference/woot13/workshop-program/presentation/Ryan. → pages

[98] M. Ryan. Bluetooth Smart: The Good, The Bad, The Ugly and The Fix. https://www.blackhat.com/us-13/briefings.html#Ryan, 2013. Accessed February 26, 2014. → pages 106, 110

[99] H. Sasamoto, N. Christin, and E. Hayashi. Undercover: Authentication usable in front of prying eyes. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '08, pages 183–192, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-011-1. doi:10.1145/1357054.1357085. URL http://doi.acm.org/10.1145/1357054.1357085. → pages 49

[100] N. Saxena, J.-E. Ekberg, K. Kostiainen, and N. Asokan. Secure device pairing based on a visual channel (short paper). In *Proceedings of the 2006 IEEE Symposium on Security and Privacy*, SP '06, pages 306–313, Washington, DC, USA, 2006. IEEE Computer Society. ISBN 0-7695-2574-1. doi:10.1109/SP.2006.35. → pages 111

[101] R. Shay, S. Komanduri, P. G. Kelley, P. G. Leon, M. L. Mazurek, L. Bauer, N. Christin, and L. F. Cranor. Encountering stronger password requirements: User attitudes and behaviors. In *Proceedings of the Sixth*

Symposium on Usable Privacy and Security, SOUPS '10, pages 2:1–2:20, New York, NY, USA, 2010. ACM. ISBN 978-1-4503-0264-7. doi:10.1145/1837110.1837113. URL http://doi.acm.org/10.1145/1837110.1837113. → pages 57

[102] E. Shi, Y. Niu, M. Jakobsson, and R. Chow. Implicit authentication through learning user behavior. In *Information Security*, volume 6531 of *Lecture Notes in Computer Science*, pages 99–113. Springer Berlin / Heidelberg, 2011. ISBN 978-3-642-18177-1. URL http://dx.doi.org/10.1007/978-3-642-18178-8_9. → pages 42

[103] S. Shuai, D. Guowei, G. Tao, Y. Tianchang, and S. Chenjie. Modelling analysis and auto-detection of cryptographic misuse in android applications. In *Dependable, Autonomic and Secure Computing (DASC), 2014 IEEE 12th International Conference on*, pages 75–80. IEEE, 2014. → pages 62, 64, 95

[104] A. Skillen and M. Mannan. On implementing deniable storage encryption for mobile devices. In *Proceedings of the 20th Annual Network and Distributed System Security Symposium*, NDSS Symposium'13, San Diego, CA, USA, 2013. → pages 2, 46, 105, 116, 121

[105] A. Smith. Nearly half of american adults are smartphone owners. http://pewinternet.org/Reports/2012/Smartphone-Update-2012.aspx. Accessed March 5, 2012. → pages 31, 41

[106] B. Ur, P. G. Kelley, S. Komanduri, J. Lee, M. Maass, M. L. Mazurek, T. Passaro, R. Shay, T. Vidas, L. Bauer, N. Christin, and L. F. Cranor. How does your password measure up? the effect of strength meters on password creation. In *Proceedings of the 21st USENIX Conference on Security Symposium*, Security'12, pages 5–5, Berkeley, CA, USA, 2012. USENIX Association. URL http://dl.acm.org/citation.cfm?id=2362793.2362798. → pages 49

[107] E. von Zezschwitz, A. De Luca, B. Brunkow, and H. Hussmann. Swipin: Fast and secure pin-entry on smartphones. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, CHI '15, pages 1403–1406, New York, NY, USA, 2015. ACM. ISBN 978-1-4503-3145-6. doi:10.1145/2702123.2702212. URL http://doi.acm.org/10.1145/2702123.2702212. → pages 49

[108] S. H. Walker and D. B. Duncan. Estimation of the probability of an event as a function of several independent variables. *Biometrika*, 54(1-2): 167–179, 1967. → pages 38

[109] D. Whiting, N. Ferguson, and R. Housley. Counter with CBC-MAC (CCM). http://tools.ietf.org/html/rfc3610, 2003. → pages 110

[110] V. Woollaston. How often do you check your phone? the average person does it 110 times a day, October 2013. URL http://www.dailymail.co.uk/sciencetech/article-2449632/. → pages 114

[111] B. Xavier. Halting password puzzles – hard-to-break encryption from human-memorable keys. In *16th USENIX Security Symposium—SECURITY 2007*, pages 119–134. Berkeley: The USENIX Association, 2007. Available at http://www.cs.stanford.edu/~xb/security07/. → pages 116

[112] A. K. L. Yau, K. G. Paterson, and C. J. Mitchell. *Padding Oracle Attacks on CBC-Mode Encryption with Secret and Random IVs*, pages 299–319. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005. ISBN 978-3-540-31669-5. doi:10.1007/11502760_20. URL http://dx.doi.org/10.1007/11502760_20. → pages 95

[113] N. H. Zakaria, D. Griffiths, S. Brostoff, and J. Yan. Shoulder surfing defence for recall-based graphical passwords. In *Proceedings of the Seventh Symposium on Usable Privacy and Security*, SOUPS '11, pages 6:1–6:12, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0911-0. doi:10.1145/2078827.2078835. URL http://doi.acm.org/10.1145/2078827.2078835. → pages 49

[114] J. Zdziarski. Identifying back doors, attack points, and surveillance mechanisms in iOS devices. *Digital Investigation*, 11(1):3–19, 2014. → pages 105

Appendix A

User Studies Questions

A.1 Pre-screening Questions

1. What is your gender?

1. Male
2. Female

2. What is your age?

1. under 18
2. 19 – 24
3. 25 – 30
4. 31 – 35
5. 36 – 40
6. 41 – 45
7. 46 – 50

8. 51 – 55
9. 56 – 60
10. 61 – 65
11. over 65

3. What is your highest level of completed education?

1. High-school
2. University (Bachelor?s)
3. Graduate School (Master?s, PhD)
4. Professional School (College degree)
5. Other

4. How many jobs do you have and what are they? *(Record each job title, industry sector)*

5. What is your household income?

1. under $15K$
2. $[15K, 30K)$
3. $[30K, 50K)$
4. $[50K, 80K)$
5. more than $80K$

6. What is your native language?

A.2 Interview Scenario and Coding Sheet

A.2.1 Introduction

Hello Mr/Ms Participant, thank you for taking part in our study. We appreciate your time. This study will be in the form of an interview, and is going to be audio recorded. The audio record will be used only for further analysis and will be securely stored at UBC before being deleted.

In this study we are investigating the use of mobile phones.

If you don't have any questions please read this consent form, and if you are agree to be interviewed today, please sign the form.

Before we start the interview, could I ask you to show us your phone(s)? *(if the participant is ok, photograph their phone(s), and find out exact model(s) and storage capacity and write this information down)*[1]

For what purposes do you use computer and smartphone?

A.2.2 Applications Types and Your Experience Today/Yesterday

Could you, please, describe us how have you used your phone(s) today from the moment you woke up.

How would you describe your daily smartphone usage? *(Give example if necessary)* For example, you could say that you are using Application A a lot during

[1] All instructions to the interviewer are in *italics*.

your usual day; occasionally Application B, and so on.

What other applications do you use on your phone?

A.2.3 Application Specific Questions

Repeat these questions for each application you identified in the previous section.

Let?s talk about each of these applications.

1. Why do you use application **XYZ** on your smartphone?

2. What kind of data do you use with that application?

3. Do you use this application for personal matters of for your work?

Note: sometime interviewee should ask explicitly whenever or not they saved username and password in the application. Likewise, if you can infer what type of data an application might use and participant didn?t mention this type of data, ask him/her explicitly.

A.2.4 Data Types Specific Questions

Repeat these questions for each data type you identified in the previous sections.

How confidential or sensitive are data records of type **XYZ** for you?

Scenario 1: Assume your smartphone got stolen by a person who knows you and you know him. Can you answer the following questions:

Do you see any risks for you, your family, or your friends if this person can see data records of type **XYZ**?
Do you see any risks for you, your family, or your friends if this person can see data records of type **XYZ** and corresponding application?

Scenario 2: Assume your smartphone is stolen by a person who doesn?t know you and you don?t know him. Can you answer the following questions:
Do you see any risks for you, your family, or your friends if this person can see data records of type **XYZ**?
Do you see any risks for you, your family, or your friends if this person can see data records of type **XYZ** and corresponding application?

A.2.5 Current Practices

1. How many computers do you use at home and at work?
2. How many of those computers you connect your smartphone(s) to?
3. What actions do you take, in order to protect your valuable, confidential and sensitive data from risks, associated with threat of your smartphone got stolen, broken, or lost?
4. Do you password protect your smartphone? Why?
5. Assume you have just lost your smartphone. What would you do in first hours?
6. How soon will you get yourself a replacement phone/smartphone?
7. What would you do with the old smartphone before giving it away?
8. Have you ever lost your smartphone or mobile phone?
9. Have you ever lost any data on any device, such as laptop, desktop or smartphone?
IF 8 or 9 IS YES THEN ask question 10
10. How it changed your practices in keeping data safe?

A.3 Study 2 Questionaire

A.3.1 Part I: Consent Forms and Smartphone Task

First participants were asked to consent with the study and kinds of data being collected. If, however, the participants stated that his/her age is less than 19 years, a separate consent form was presented for parents/guardians. We then asked participants to follow a link on their smartphones and fill out some contact details so that we can contact them if they win the raffle. For this purpose the following message was displayed:

Please follow the link with on your smartphone

As you know we do a raffle among participants of this study for one iPad 3 (WiFi, 32GB). In order to be considered for this raffle you have to follow this link on your smartphone and provide your contact details in the form. Otherwise, because this study is anonymous, we will have no means to communicate to you if you win the prize. http://study.csnow.ca/code.aspx
and enter your email and phone. Warning: you should visit this link from your smartphone, only those who did so will be considered for the raffle.

Your activation code is:
CODE: ***ABC123***
Please, click "Continue" button after you submit your contact details from your smartphone.

A.3.2 Part II: Demographic Questions

The demographic section of the survey included the following questions:
Question A: What is your gender?

1. Male
2. Female

Question B: What is your age?

1. Under 10
2. 10-14
3. 15-17
4. 18-24
5. 25-29
6. 30-34
7. 35-39
8. 40-44
9. 45-49
10. 50-54
11. 55-59
12. 60-64
13. 65+

Question C: What is your highest level of completed education?

1. Less than or still in High School
2. High School
3. University (Bachelor's)
4. Graduate School (Master or PhD)
5. Community College or Professional School (College degree)
6. Other

Question D: List any work for which you have been paid in the past 3 months. Provide position title for each job. (Open-ended question.)
Question E: Select in what industry(ies) have you worked for the past 3 months? (Mark all applicable)

1. None or Unemployed
2. Agriculture
3. Forestry, fishing, mining, quarrying, oil and gas (Also referred to as Natural resources)
4. Utilities
5. Construction
6. Manufacturing
7. Trade
8. Transportation and warehousing
9. Finance, insurance, real estate and leasing
10. Professional, scientific and technical services
11. Business, building and other support services
12. Educational services
13. Health care and social assistance
14. Information, culture and recreation
15. Accommodation and food services
16. Public administration

17. Other services

Question F: What is your annual household income in US Dollars?

1. I prefer not to answer
2. Under 5,000 USD
3. From 5,000 USD, up to 9,999 USD
4. From 10,000 USD, up to 14,999 USD
5. From 15,000 USD, up to 29,999 USD
6. From 30,000 USD, up to 49,999 USD
7. From 50,000 USD, up to 74,999 USD
8. From 75,000 USD, up to 99,999 USD
9. From 100,000 USD, up to 149,999 USD
10. More than 150,000 USD

A.3.3 Part III: Smartphone Experience

Question A: How many smartphones currently do you have and use?

1. One
2. Two
3. Three
4. More than Three

Question B: Describe ownership of the smartphones you currently use. (Select all that apply)

1. I bought a new smartphone for personal use
2. I bought used smartphone for personal use
3. My friend/relative gave me a smartphone as a gift
4. My smartphone is given me by my Employer/Company for work
5. My smartphone is given me by my Employer/Company as a gift
6. Other

Question C: Please select what kind of previous experience you have with mobile phones and smartphones (select all that apply).

1. I have lost my mobile phone before and didn't find it"
2. I have broken my mobile phone before, so that it was not usable"
3. I have left my mobile phone at some place, but recovered it later (e.g., at my friends' place, in a restaurant, at parents' house, at school, etc.)"
4. Someone used my mobile phone without my permission with intention to look at some of my data"
5. Someone used my mobile phone without my permission with intention to use its functionality"
6. Someone used my mobile phone without my permission with no bad intentions"
7. I used someone's mobile phone without owner's permission to look into his/her data"
8. I used someone's mobile phone without owner's permission for some functions (phone call, browsing the Internet)

Finally, we asked participants if they used any locks on their smartphone. In order to define what smartphone lock is we first provided them with the following explanation: *In the following question we will ask you about your phone lock. By “phone lock” we mean a protection of the smartphone that requires some "secret", such as password or PIN-code, to unlock it. Here are some examples of phone locks:* (Figure A.1).

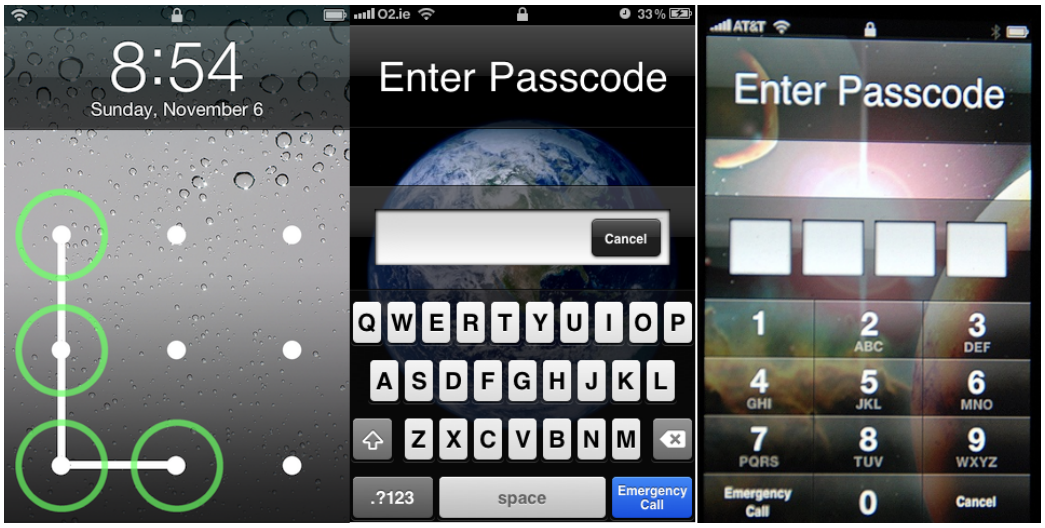

Figure A.1: Different types of smartphone locks.

Question D: Do you use any type of locks, shown above, on your smartphone(s)?

1. Yes, I use lock
2. No, I don’t use lock

A.3.4 Part IV: Smartphone Lock Use

In this section of the survey we asked participants about the reasons they used or not used a smartphone lock. The type of the question a participant saw depended on whether they answered that they used a lock or not. We asked the following question to all of the participants that did not use a lock.

Question A: Why do not you use a lock on your smartphone(s)? (Mark all applicable)

1. I do not have any data that I want to hide on my phone
2. I do not care if my phone services will be used by someone
3. I tried locks before and found them very inconvenient
4. I often need instant access to applications that do not store any sensitive data (e.g. weather forecast, news, games)
5. I do not save my passwords in applications and type it every time I use an application that stores sensitive data (e.g. email application, or Facebook application)
6. It is not worth for me to use smartphone lock, because the amount of data and applications that are sensitive are very small compared to those non-sensitive

The participants that did use a lock were asked the following set of questions:
Question A: Which of the locks do you use in your smartphone(s)? (Mark all applicable)

1. PIN-Code (only digits)
2. Password (could have digits and letters)
3. Draw a Secret (Pattern)
4. Face-recondition
5. Finger print scan
6. Other

Question B: How does the lock in each of your smartphone works? (Mark all applicable)

1. It locks my smartphone after I pushed power button
2. It locks my smartphone after smartphone's display switches off
3. It locks my smartphone when I am not using it for some period of time
4. It locks my smartphone after it switched off completely or rebooted (which might happen because battery might got fully discharged or smartphone got rebooted it
5. It locks my SIM card if phone got switched off completely or rebooted (which might happen because battery might got fully discharged or smartphone got rebooted it
6. Other

Question C: Why do you use lock on your smartphone(s)? (Mark all applicable)

1. My employer requires that
2. I feel comfortable having such protection
3. I have confidential and sensitive data on my smartphone(s)
4. I do not want other people to use my phone services without my permission
5. I do not want other people sneaking into my smartphone, when I do not see it

Question D: If your employer requires you to use a phone lock, assume for a moment that this is not the case and you can decide on your own. Rate your agreement with the following statement: " would rather use a smartphone lock for my entire smartphone, than a lock for specific applications or data items"

1. Strongly Disagree
2. Disagree

3. Neutral
4. Agree
5. Strongly Agree
6. Not Applicable

Question E: If your employer requires you to use a phone lock, assume for a moment that this is not the case and you can decide on your own. Rate your agreement with the following statement: "My need to get fast access to some applications or data on my smartphone influenced or will influence my decision on phone lock use"

1. Strongly Disagree
2. Disagree
3. Neutral
4. Agree
5. Strongly Agree
6. Not Applicable

A.3.5 Part V: Applications and Data Being Used

Question A and B was asked separately for work and personal use cases. Our survey also allowed the participants to add new application or data types, in case if the list was incomplete.
Question A: In the past year, which applications or features have you used in your smartphone(s) for work or personal use? (Mark all applicable)

1. SMS/MMS messages
2. Voice Calling

3. Email Client
4. Calendar
5. Notes
6. Instant messenger (e.g. GTalk, MSN Messenger, ICQ etc.)
7. Social Networking application (e.g. Facebook, Tweeter, Google+, What's UP etc.)
8. Voice recorder
9. Photo Camera
10. Music Player
11. Video Player
12. Maps
13. Training Assistant that helps you to track your exercise performance on the map.
14. Password Keeper/Manager
15. Games
16. Documents viewers or editors (e.g. Word, Excel, Adobe Acrobat etc.)

Question B: Select data which you created, or received, or stored on your smartphone during last 12 months? (Mark all applicable)

1. SMS/MMS Messages
2. Call history
3. Browser Search History

4. Browsing History
5. Photos and Videos
6. Voice Recordings
7. Notes and Memos
8. Contacts Details
9. Music
10. Emails
11. Documents
12. Events in Calendar
13. Data in Social Networking Applications
14. Recorded GPS tracks, from such applications as training assistants
15. Progress in Games
16. Passwords (that includes password managers, passwords in notes and saved passwords in applications)

Note, that in Question C as available options we used which ever data type a participants selected or added as an answer to Question B.
Question C: In the past year, select data which you preferred to delete from your smartphone(s) immediately after reading/using it? (Mark all applicable)

A.3.6 Part VI: Password Saving Habits

Question A: How do you check your email on your smartphone(s)?

1. I open the application and see my emails immediately; the application does not ask me for password

2. I open the application, then I type my email account password, after that I see my emails
3. I use Internet browser to check my emails
4. I do not check email on my phone
5. Other

Question B: With other applications, where an account is required, I usually...

1. Save my account password, so that I can open application faster
2. Do not save account password and type it all the times
3. I do both
4. Other

A.3.7 Part VII: Data Types Sensitivity and Value

All data that were identified in Part V of this questionnaire were listed in this section and users had to rate their agreement with various statements depending on proposed scenario and type of data use, i.e., personal or work. We used the following set of statements to assess data type's sensitivity:

1. I would not have any concerns if Personal/Work *DataType* could be viewed by such a thief
2. I will have some concerns if Personal/Work *DataType* could be viewed by such a thief
3. I think such thief will use my Personal/Work *DataType* for some purpose, that might be detrimental to me
4. I think such thief will not use my Personal/Work *DataType* for any purposes

For the states 1 and 3 we proposed the following scenario:

- Scenario 1: Assume your mobile phone has just been stolen by a thief, who does not know

For the states 2 and 4 we proposed the following scenario:

- Scenario 2: Assume your mobile phone has just been stolen by a thief, who knows you. It could be anyone from your social circle, but you do not know who exactly he or she is.

In order to compare different data types between each other we asked the participants to rank their data based on its value. This task was done through drag and drop, where the participants were modifying the rank list by moving items with their mouse up or down the list. The participants saw the following task statement:

- Rank Task 1: Scenario 1: Assume your mobile phone has just been stolen by a theft, who does not know you. Rank data types so that you would have the biggest concern with the first item being revealed and the least amount of concern with the last item being revealed. Use drag and drop for ordering.

- Rank Task 2: Scenario 2: Assume your mobile phone has just been stolen by a theft, who knows you. It could be anyone from your social circle, but you do not know who exactly he or she is. Rank data types so that you would have the biggest concern with the first item being revealed and the least amount of concern with the last item being revealed. Use drag and drop for ordering.