The Bumpy Relationship of Developers and Cryptography

Mohammadreza Hazhirpasand

b
UNIVERSITÄT
BERN

Faculty of Science
Institute of Computer Science
University of Bern

Prof. Dr. Oscar Nierstrasz

DISSERTATION

The Bumpy Relationship of Developers and Cryptography

for the attainment of the PhD degree of the University of Bern

written by

Mohammadreza Hazhirpasand
from Rasht, Iran

Bern, April 2022

Supervisors:
Prof. Dr. Oscar Nierstrasz
Prof. Dr. Mohammad Ghafari

Examiner:
Prof. Dr. Awais Rashid

Created at:
Institute of Computer Science
University of Bern
Hochschulstrasse 6
CH-3012 Bern

Typesetting:
Overleaf.com
LaTeX, LaTeX Project Public License (LPPL)
UZH/USZ habilitation template from J. von Spiczak, CC BY 4.0
ETH CADMO template from F. Mousset and H. Einarsson

Print:
Copy Shop AG
Bahnhofstrasse 1
CH-8001 Zürich

`https://www.copyshop.ch/`

I would like to acknowledge and give my warmest thanks to Prof. Oscar Nierstrasz. His profound knowledge about various facets of life and his impeccable character not only helped me in computer science but also shed light on other areas throughout all these years. A debt of gratitude is also owed to Prof. Mohammad Ghafari whose expertise and criticality considerably improved my initially dim academic views. This journey could have never been completed without the presence, constructive feedback, and amazing accompany of my dear intelligent colleagues, namely Pascal, Nitish, Manuel, Pooja, and many others who visited us off and on. Moreover, I would like to thank Arvin, Danial, Hamed, Arian, any many others who were not present in Switzerland but they never failed to give their moral support. I am deeply grateful to my wonderful mother whose unconditional love has given me life and hope. Finally, I would like to dedicate my dissertation to my departed father whose delightful memories have always inspired and emboldened me to surmount even the most formidable obstacles.

Abstract

As the cornerstone of the internet, cryptography is becoming increasingly important in software development. Nevertheless, the way this cornerstone is laid is so critical that a mistake can result in grave reputational and financial loss. Given the rapid growth of applications for various platforms and devices, developers with varying levels of expertise are more likely to make catastrophic mistakes in employing cryptography. The imminent threat of misusing cryptography prompted us to investigate what factors impede developer performance.

Having explored how cryptography is used in open-source as well as enterprise projects, we realized that crypto API misuses do occur in both areas. To understand the primary causes, we investigated the prevalence of crypto API misuse from two major aspects, *i.e.*, the API and developer perspectives, and presented feasible remedies.

From the API perspective, we conducted three studies on Stack Overflow: (1) *a large-scale analysis of 91 954 crypto-related questions*, (2) *an analysis of 500 questions with regards to 20 crypto libraries*, and (3) *a close scrutiny of Java crypto APIs*. We realized that there is a distinct lack of knowledge among askers in fundamental concepts, such as certificates, asymmetric and password hashing, and that the complexity of crypto libraries weakened developer performance to correctly implement a crypto scenario. More specifically, libraries are not yet designed so as to help avoid inadvertent misuse, aside from their problematic installation and usage. The API-level analysis showed that APIs require myriad options and leave developers inundated with many alternatives to choose from. Furthermore, the code snippets, as well as solutions on Stack Overflow, contain security violations, resulting in a massive ripple effect as others may end up with untrustworthy sources and examples.

From the developer perspective, we conducted four studies: (1) *an analysis of developer performance in using crypto APIs*, (2) *gathering open-source maintainers' feedback for their crypto misuses*, (3) *a survey with 97 developers who used crypto APIs in open-source projects*, and (4) *an analysis of crypto experts' activity on Stack Overflow and GitHub*. We found out that four factors of developer experience, *i.e.*, developer involvement in multiple projects, the number of crypto commits, crypto API diversity, and the frequency of committed lines of code, did not improve developer performance over time. Developer feedback on GitHub revealed that security hints in API documentation are scarce, that some misuses stem from third-party libraries, and that code context affects the way crypto APIs are used. While being concerned about security, developers often fail to incorporate security standards into their developments, *e.g.*, low rate of adoption of security tools or security-concerned questions on Stack Overflow. They also have a low tendency towards consulting educational sources particularly tailored for cryptography and are more inclined to turn to untrustworthy sources, *e.g.*, Stack Overflow. The findings showed that crypto experts' practices on GitHub accord with the crypto topics and programming languages they feel confident to contribute on Stack Overflow.

As for plausible remedies for alleviating crypto API misuses, we contacted the top 1% of crypto experts to collect their views regarding root causes and solutions. Crypto experts mentioned that the root causes for the challenging areas can be classified into three major categories: learning resources, crypto APIs, and human-related. They also suggested a number of solutions, such as employing misuse-resistant libraries and improving one's knowledge by consulting dependable online sources, *e.g.*, Coursera. We also introduced a tool, *i.e.*, CryptoExplorer, to assist developers by delivering real-world examples. A preliminary study of CryptoExplorer showed that the tool helps developers explore secure crypto examples and learn how to correctly use crypto APIs by comparing examples of correct uses and misuses.

We conclude that existing approaches may arguably have a limited impact, cannot be practical on a large scale, and can only target a specific audience. We believe that there are two promising methods to cope with this issue successfully: (1) developing misuse-resistant crypto APIs to render unintentional API misuse exceedingly improbable, (2) producing high-quality, easy-to-understand, and entertaining online tutorials to broaden developer knowledge in this domain.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Nowadays security is an inextricable part of software development. This is due to the fact that malicious minds employ advanced techniques to bypass security measures. We discovered, for instance, that clever methods in clickjacking attacks allow attackers to lure victims into granting web permissions unwittingly [73]. In one study, we discussed the latent inherent flaw used by attackers that can circumvent the same-origin policy of browsers and severely jeopardize the security of the users' internal network [67]. Similarly, we demonstrated the peril of using JavaScript together with a flaw in WebRTC to scan the users' private network and conduct a seemingly invisible enumeration [68]. To shield software systems, security professionals observe three essential security goals, *i.e.*, confidentiality, integrity, and availability. Overlooking any of the three aforementioned pillars causes a major loss to software companies.

There exist a number of well-known attacks, leaving each of the aforementioned pillars vulnerable. For instance, adversaries conduct snooping attacks to jeopardize the confidentiality, replaying attacks to threaten the integrity, and denial of service attacks to put the availability of software systems at risk. Fortunately, most of such attacks can be effectively prevented provided that developers use the recommended secure coding practices. However, due to the nature of open-source software and the fast-growing pace of information technology, developers might possess varying levels of expertise/experience and educational background. The diversity exacerbates the problem so that developers can be dimly aware of the inevitable consequences of insecure coding practices.

Cryptography is indispensable to information security in supporting the three essential goals, *i.e.*, integrity, confidentiality, and authentication [26]. Hence, developers' mistakes in employing cryptography can bring about undesirable outcomes, *e.g.*, infringing users' privacy. For instance, integrity, as one of the objectives, ensures that data are not changed in transit and manipulation is not permitted by an unauthorized person or program. There exist measures often referred to as cryptographic checksums, *e.g.*, Message Au-

thentication Code (MAC), to verify integrity. Nevertheless, choosing a weak hashing algorithm for integrity checks results in security breaches if adversaries attempt to conduct tampering, injection, or brute-force attacks [113]. The second goal of cryptography is to provide confidentiality in a way that a concealed message cannot be read by unauthorized parties. This process commonly necessitates the usage of cryptographic keys so that the recipient decrypts the message with a key that may or may not be the same as the one used by the sender. However, developers' inadequate knowledge causes poor, insecure choices in areas such as algorithms and key sizes [119]. For instance, developers either unwittingly use DES as an algorithm for encryption/decryption or knowingly choose a strong algorithm but opt for weak key sizes, imposing severe security risks on the entire software system. The third goal of cryptography is to provide authentication in which a user can prove their identity to other users. This can be achieved by digital certificates. Entailing various subjects, digital certificates, however, have a steep learning curve for developers [77].

Indeed, such mistakes stem from two major sources, (1) inadequate knowledge of developers, (2) the complexity of crypto APIs. Regarding cryptography knowledge, developers who do not work in a company may not receive any formal security training or have no access to a security consultant. Even if working in a company, developers may still not benefit from any security training [76]. Yet, general security training, if provided, cannot alone positively impact developers' performance in writing secure code [97]. As a consequence, developers feel a conspicuous lack of confidence while working with cryptographic concepts and seek quick solutions from online sources. Unfortunately, the official documentation of various programming languages does not sufficiently deter developers from using insecure APIs or parameters owing to the lack of security hints [72]. Hence, developers refer to online forums, *e.g.*, Stack Overflow, to resolve the uncertainties but such places have proven to contain unverified code snippets and suggestions. A blind imitation of responses from such online sources can inadvertently influence the security of thousands of software systems.

The second decisive factor in emerging crypto mistakes is the noticeable complexity of crypto APIs. There are commonly several crypto libraries for any given programming language, most of which suffer from usability smells [120]. Improving APIs often tends to be a challenging task since not only is the presence of API authors required but also any drastic changes in such APIs would cause backward compatibility issues. A more feasible approach suggested by researchers is to add code examples to the documentation, which provide immense help for the comprehension of how an API works. Although static analysis tools enable developers to alleviate the hardship of securely using crypto APIs, developers are not willing to use such tools due to various reasons, *e.g.*, project-level constraints, or unfamiliarity with such tools [147].

1.1 Thesis Statement

This study aims to investigate what factors impede developer performance in using cryptography correctly. We explain our thesis as follows:

> *Cryptography is perceived as being hard by developers and requires expert knowledge to understand its various sub-areas, e.g., symmetric/asymmetric encryption. Developers also do not benefit from extensive learning material, reliable sources, and security tools in their development.*

To investigate the root causes of why cryptography is misused, we explore various aspects of crypto APIs and developer practices. We conduct several studies to understand which issues are common, why these issues occur, and what feasible solutions can alleviate the issues. In particular, we are going to address this grave concern with four major research questions (RQs). Each RQ provides valuable insights into why developers struggle with using crypto APIs correctly. The four RQs are as follows:

- RQ1: What crypto mistakes do developers make in the wild?
- RQ2: What hurdles are mostly discussed concerning crypto APIs?
- RQ3: What are the practices of developers in using cryptography?
- RQ4: What are the feasible remedies to alleviate the issue of misusing crypto APIs?

1.2 Contributions

In this study, our contributions are presented in various published papers. There are a number of works in which the participation of developers was part of the research work. However, we acquired the necessary permission and research ethics within our research group to conduct such studies. We list the published research papers with the icon, the proposed tool papers with the icon, and the submitted papers with the icon.

Motivation : The motivation of why security is important:

– Chapter 2 → Mohammadreza Hazhirpasand, Mohammad Ghafari, and Oscar Nierstrasz. Tricking Johnny into granting web permissions. In *Proceedings of the Evaluation and Assessment in Software Engineering*, pages 276–281, 2020
Mohammadreza Hazhirpasand and Mohammad Ghafari. One leak is enough to expose them all. In *International Symposium on Engineering Secure Software and Systems*, pages 61–76. Springer, 2018
Mohammadreza Hazhirpasand, Arash Ale Ebrahim, and Oscar

Nierstrasz. Stopping DNS rebinding attacks in the browser. In *ICISSP*, pages 596–603, 2021

Arash Ale Ebrahim, Mohammadreza Hazhirpasand, Oscar Nierstrasz, and Mohammad Ghafari. FuzzingDriver: the missing dictionary to increase code coverage in fuzzers. In *IEEE 29th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 2022

*RQ*1 : To study how cryptography is used in the wild:

- Chapter 3 → Mohammadreza Hazhirpasand and Mohammad Ghafari. Cryptography vulnerabilities on HackerOne. In *IEEE International Conference on Software Quality, Reliability and Security*. IEEE, 2021

  Mohammadreza Hazhirpasand, Mohammad Ghafari, and Oscar Nierstrasz. Java cryptography uses in the wild. In *Proceedings of the 14th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pages 1–6, 2020

*RQ*2 − 3 : Hurdles in cryptography:

- Chapter 4 → Mohammadreza Hazhirpasand, Oscar Nierstrasz, Mohammadhossein Shabani, and Mohammad Ghafari. Hurdles for developers in cryptography. In *37th International Conference on Software Maintenance and Evolution (ICSME)*, 2021

  Mohammadreza Hazhirpasand, Oscar Nierstrasz, and Mohammad Ghafari. Dazed and confused: What's wrong with crypto libraries? In *18th Annual Conference on Privacy, Security and Trust (PST)*. IEEE, 2021

  Mohammadreza Hazhirpasand, Mohammad Ghafari, Stefan Krüger, Eric Bodden, and Oscar Nierstrasz. The impact of developer experience in using Java cryptography. In *2019 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pages 1–6. IEEE, 2019

  Mohammadreza Hazhirpasand, Oscar Nierstrasz, and Mohammad Ghafari. Worrisome patterns in developers: A survey in cryptography. In *Proceedings of the 36th IEEE/ACM International Conference on Automated Software Engineering Workshops*, 2021

  Mohammadreza Hazhirpasand, Oscar Nierstrasz, and Mohammad Ghafari. Crypto experts advise what they adopt. In *Proceedings of the 36th IEEE/ACM International Conference on Automated Software Engineering Workshops*, 2021

  Mohammadreza Hazhirpasand, Mohammad Ghafari, and Oscar

Nierstrasz. Java cryptography uses in the wild. In *Proceedings of the 14th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pages 1–6, 2020
Sophie Gabriela Pfister (2021), Jenny in Wonderland Exploring the Difficulties of Symmetric Encryption [Bachelor's thesis, University of Bern]

*RQ*4 : To point out feasible remedies and the lessons learned:

– Chapter 6 → Mohammadreza Hazhirpasand, Oscar Nierstrasz, Mohammadhossein Shabani, and Mohammad Ghafari. Crypto heroes: Views and recommendations. In *Proceedings of the 37th ACM/SIGAPP Symposium on Applied Computing*, 2022
Mohammadreza Hazhirpasand, Mohammad Ghafari, and Oscar Nierstrasz. CryptoExplorer: An interactive web platform supporting secure use of cryptography APIs. In *2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 632–636. IEEE, 2020

1.3 Outline

The thesis investigates each RQ with reference to the state of the art, motivation, methodology, results, and contributions.

Chapter 2 explains why security is important and presents the state of the art of cryptography with respect to each research question. The chapter highlights the gap that exists, which we address in this thesis.

Chapter 3 explores how cryptography is employed in the wild. We first check the extent to which crypto vulnerabilities are problematic in industry. Then, we check developers' coding practices in using crypto Java crypto APIs in open-source projects.

Chapter 4 examines issues in cryptography with respect to three facets. For the first aspect, we consider the issues developers encounter while working with crypto APIs. To do so, we check on which common problems developers ask for help on Stack Overflow. Then, we take a closer look at questions that are specifically related to crypto libraries. Lastly, we choose a popular language, *i.e.*, Java, to obtain a view of what problems are common in Java crypto APIs that are intended to use for symmetric encryption.

Chapter 5 studies developer practices with respect to four facets. First, we examine whether developers experience an improvement in their performance based on several factors, *e.g.*, increase in the number of commits, or days of contribution. Then, we contact some of the developers who had the

experience of working with crypto APIs and collect their reactions to their performance in using cryptography. We also survey developers who had crypto-related commits on GitHub's open-source projects. We ask them about their coding and security practices to understand their characteristics. Finally, to see how crypto experts on Stack Overflow use cryptography on GitHub, we attempt to find their practical experiences on GitHub.

Chapter 6 demonstrates the root causes as well as possible temporary solutions for facilitating the correct usage of cryptography. To this end, we contact crypto expert users on Stack Overflow and ask their opinions for improving the state of cryptography, both for the developer and library perspectives. We also present a tool called CryptoExplorer, which enables developers to search for crypto code snippets. The code snippets are analyzed by a static analysis and if there are any misuses, CryptoExplorer highlights the security issues.

Chapter 7 reviews the research questions, delineate the concluding observations, and designates potential future work in this direction.

Chapter 2

State of the Art

Cryptography has been studied from various aspects, such as how to break crypto algorithms, how it has been adopted by developers, and how crypto APIs should be devised and improved in regard to usability. We first elucidate why security, in general, is a critical element in software development based on our previous studies. Then, we focus on the studies with respect to cryptography that target the areas relevant to this thesis. In the following, we briefly explain the selected areas. First, knowledge acquisition consists of three subcomponents, *i.e.*, training, education, and online sources, that either boost or, if not adequately acquired, adversely affect the performance of developers. Second, crypto API misuse is one possible consequence of thoughtless selection, faulty implementation, and improper assessment of using cryptography. Third, various security tools are proposed to assist professionals to ease the use of cryptography; nevertheless, the extent to which such tools are usable and adopted is unclear. Lastly, developer performance in cryptography widely varies, the causes of which have been investigated from various aspects. At the end of each section, we present our findings in brief and how they are in line with the associated literature.

2.1 The importance of security

The importance of incorporating security into all phases of software development cannot be overstated. However, in contrast to software developers, adversaries are continuously looking for ingenious methods to circumvent the already in place security measures. For example, we observed that adversaries can effortlessly conduct IP and port enumeration via four different methods, *e.g.*, WebSocket, to discover live hosts in the web visitor's private network [68]. We found that an attacker requires fewer than 5 minutes to relatively map a /24 subnet mask of the victim. As regards browsers, we detected a design flaw in the permission box of browsers [73]. This issue enables nefarious attackers to design a decoy element, place it precisely in a particular

coordinate, and trick victims into playing a game, resulting in allowing critical permissions, *e.g.*, webcam or location. In addition, we demonstrated how adversaries can bypass the same-origin policy of browsers with the help of DNS rebinding attacks and stealthily interact with the internal web-based platforms. Comparing various preventive measures, we confirmed that the most effective way to halt DNS rebinding attacks is to use signed SSL/TLS certificates [67]. Lastly, both security professionals and sophisticated hackers employ fuzzers to find security vulnerabilities in software. For instance, Grieco *et al.* developed a novel fuzzer, called ContractFuzzer, to test Ethereum smart contracts and discovered 459 vulnerabilities in 6991 smart contracts [63]. In the same vein, to increase the efficiency of coverage-based greybox fuzzers, we introduced FuzzingDriver, which provides fuzzers with useful dictionaries to outperform their code coverage and to increase the likelihood of discovering security vulnerabilities [8].

2.2 Knowledge acquisition

Training: A prime factor in successful information security management is effective compliance of security policies and suitable integration of people, processes, and technologies. Security awareness training of employees is one of the mechanisms in which the effectiveness of integration can be enhanced. Emina *et al.* conducted a security awareness program in a company with 2900 employees and subsequently audited how effective and successful the program was [46]. The results showed positive outcomes in relation to the impact of human awareness of the success of information security management programs. Siponen *et al.* conducted a field survey to comprehend which factors assist employees' compliance with security policies [135]. Their results demonstrated that the visibility of information security policies has a considerable impact on employees' adherence to the policies. Furthermore, employees comply with information security policies if they are fully aware of how vulnerable their organization is to security hazards and the severity of these threats. A group of researchers explained a theoretically grounded information security training that was validated through action research [79]. The findings imply that by using constructiveness in the context of security training, the security behavior of employees can be enhanced. Kweon *et al.* attempted to discover a relationship between cybersecurity training and the number of incidents of organizations [96]. They realized that the role of security training and education has a positive impact on reducing the number of incidents in organizations. The need for regular information security training is undeniable in companies [143]. From the training frequency viewpoint, quarterly security awareness training is recommended to renew employee knowledge concerning the latest threats and trends, and in case some difficulties exist, biannual training could be the minimum required time frame [57]. Puhakainen *et al.* stressed that in-

formation security trainings and communication efforts should be continuous and integrated into the organization's usual communication efforts otherwise security policies lose their efficacy [125]. According to the SANS Institute, a security awareness program should consider who is going to be in the training course, which topics are suitable for the audience, and ultimately how participants engage in order to identify how frequent security training should take place.[1] Last but not least, the lack of an official role in organizations as security champions is evident, and oftentimes this role is given to someone on the development team with limited security knowledge. By hiring security consultants, managers can benefit from the resulting security level of products, and security testers would largely capitalize on the presence of such knowledgable consultants [140] [123].

Education: Cryptography is an essential component of computer security, network security and all information security-related courses [106]. However, a cryptography course must be carefully adopted based on target audience since they can be from various fields, such as computer science, mathematics, and information security [6]. In a study, the researcher focused on design thinking (DT) activities for teaching cryptography to students [9]. They devised a cryptography curriculum for a semester and the pilot study showed improvements in solving complex cryptographic problems as well as better student comprehension. Hamdani *et al.* addressed the issue of creating a stand-alone cryptography course independent of other subjects, such as discrete mathematics, statistics, number theory, and modern algebra[7]. They proposed two detailed curricula for the undergraduate and graduate students. Aydin contextualized abstract ideas from algebra and number theory, taught in a mathematics course by utilizing computer science and engineering examples from cryptography and coding theory [12]. For students, the theory and applications of cryptography are considered complicated. Adamovi *et al.* eased the learning curve of cryptography by introducing a different, interactive open-source tool, called CrypTool [4]. They received affirmative feedback with regard to the teaching method compared to the traditional approaches. Bajracharya *et al.* argued the paramount importance of incorporating data security disciplined (*i.e.*, cryptography, steganography, and watermarking) into academic courses [16]. They concluded that such fundamental knowledge leads the students to careers such as digital media forensics, steganography, and cryptography expert. All in all, mathematics and cryptography are inextricably linked to each other, and for instance, public-key cryptography requires significant background in algebra, number theory, and geometry, which renders the understanding of the main subject difficult for inexperienced, average developers [55].

Online sources: Massive Open Online Courses (MOOCs) supply free or

[1] `https://www.sans.org/security-awareness-training/blog/wrong-question-how-long-should-security-awareness-training-be`

affordable online courses available for anyone interested in studying cryptography. There are two MOOC platforms, (1) Coursera (offered by Stanford University), and (2) edX (offered by MITx), that present a large number of computer science courses. There are many cryptography courses, *e.g.*, "Cryptography 1" [2] and "Cryptography" [3] , available on the Coursera platform for interested students to learn [95]. There exists a large number of online cybersecurity courses, including cryptography, on other platforms, *e.g.*, Khan Academy [60]. [4]

Although using Stack Overflow might help the functional correctness, it leads to more insecure copy-pasted code snippets [2]. Ye *et al.* worked on a system called insecure code snippet detection (ICSD) to detect the imminent insecure code snippets available on Stack Overflow [157]. In a survey with 87 Stack Overflow visitors, researchers found that Stack Overflow answers contain outdated answers, wrong solutions, and buggy code [126]. Acar *et al.* conducted a comprehensive study by surveying 295 application developers, and a lab study with 54 Android developers (professionals and students) in which they were allowed to resolve coding issues with one of the following four means: any resources, Stack Overflow only, official Android documentation only, or books only [2]. Their findings suggest that developers use Stack Overflow as a major source. Interestingly, developers who could use any resources had similar performance (functional and security correctness) to those who were assigned to use Stack Overflow only. Acar *et al.* pointed out that official API documentation commonly (1) lacks security-related hints, (2) crypto libraries should offer a simple (3) and accessible documentation with secure, easy-to-use code examples [1].

In this study, we show that developers who used cryptography had a strong tendency to use insecure resources, *e.g.*, Stack Overflow, and there was a lack of security training and consultants at workplace. As for the educational tools, we propose CryptoExplorer. In particular, CryptoExplorer aim was to help developers to learn the correct way of using crypto APIs by looking into real-world pre-analyzed crypto examples. Suggestions by crypto experts highlight the need for a dedicated curriculum for cryptography in universities, abundant dependable, useful resources, *e.g.*, Coursera, and improvements in code examples and security warnings in documentation.

2.3 Crypto API misuse

The improper usage of cryptography can expose imminent threats. For instance, authentication is the first gate to every software and in microservices, JSON Web Token (JWT) plays an important role in token-based user-to-

[2] `https://www.coursera.org/learn/crypto`
[3] `https://www.coursera.org/learn/cryptography`
[4] `https://www.khanacademy.org/`

service authentication [155]. Nonetheless, the JWT tokens are prone to some attacks, such as failing to verify the signature, kid parameter injection and allowing the None algorithm.[5] The "Public-Key Cryptography Standards," or "PKCS" standards consist of a number of components, called PKCS #1, #3, and #5-15. PKCS is defined for both binary and ASCII data types, describing the syntax for messages in an abstract manner. For instance, PKCS #8 is a private-key information syntax standard, defining a method to store private key information [148]. However, there exist some security caveats for PKCS #11 and PKCS #1 1.5 [27, 84]. There are a number of encryption modes, which can be puzzling for inexperienced developers. For instance, the block ciphers, *e.g.*, DES which deals with encrypting fixed length of data, use various chaining modes, *e.g.*, cipher block changing or CBC, to encrypt bulk data. The advantages and disadvantages of encryption modes, such as CBC, ECB and CTR, have been well studied from the security and performance perspectives [101, 19, 42]. Initialization vectors (IV) have been seen as a problematic issue in developers' questions on Stack Overflow. Developers should be aware of IV attacks as a security risk of the CBC encryption mode in block ciphers and that they can be applied also in IPsec [105]. In such attacks, an unauthenticated IV in CBC encryption is used so that the adversary can control the first block of the decrypted plaintext. IV attacks also exposed the security of WEP protocol in wireless networks [139]. The length of cryptography keys and random-number generators play a crucial role in the security of the outcome cipher [133]. For instance, recommendations such as using 128-bit keys rather than 40-bit weak keys, choosing triple-DES over DES, 2,048-bit RSA is stronger than 1,024-bit RSA, or using secure random-number generators are often seen in various security standards.

The study by Lazar *et al.* is the most relevant one to our work [98], wherein they performed a systematic study of 269 cryptographic vulnerabilities reported in the CVE database from 2011 to 2014. They categorized crypto vulnerabilities into four main groups, *i.e.*, plaintext disclosure, man-in-the-middle attacks, brute-force attacks, and side-channel attacks. Braga *et al.* carried out a data mining technique, namely Apriori, to extract association rules among cryptographic bad practices, platform-specific issues, cryptographic programming tasks, and cryptography-related use cases from three popular forums: Oracle Java Cryptography, Google Android Developers, and Google Android Security Discussions [29]. They classified their findings into nine categories: namely Weak Cryptography (WC), Bad Randomness (BR), Coding and Implementation Bugs (CIB), Program Design Flaws (PDF), Improper Certificate Validation (ICV), Public-Key Cryptography (PKC) issues, Poor Key Management (PKM), Cryptography Architecture and Infrastructure (CAI) issues, and IV/Nonce Management (IVM) issues. Chatzikonstantinou *et al.* conducted a study on how developers use cryptography in mobile ap-

[5] `https://www.netsparker.com/blog/web-security/json-web-token-jwt-attacks-vulnerabilities/`

plications [35]. They discovered that 87% of the applications had at least one crypto misuse in one of the four cryptographic categories defined in the study: usage of weak cryptography, weak implementations, weak keys, and weak cryptographic parameters.

In spite of crypto library authors' good intentions, their strategy of warning developers seems to be largely a failure until now. Green and Smith proposed ten principles to make cryptography libraries more usable and constructing more secure APIs [62]. For instance, they emphasized that APIs should sufficiently satisfy both security and non-security requirements, or APIs must be easy to learn, even in the absence of cryptographic expertise. To extend the Green and Smith work, Patnaik *et al.* investigated the extent to which crypto libraries implement the ten principles [120]. They selected seven crypto libraries and identified 16 underlying usability issues. While Patnaik *et al.* study provided corroborative evidence to validate Green and Smith's principles, they also pointed out issues that were missed in the previous study. They derived four usability smells, which are indicators that an interface may be challenging to use for its intended users. Mindermann *et al.* demonstrated a list of recommendations for designing crypto libraries according to an experiment using Rust crypto APIs [108]. They pointed out insecure defaults and options, the use of authenticated encryption in low-level libraries, the absence of warnings concerning deprecated, broken features, and the scarcity of examples in the documentation. They also declared that their recommendations are just more specific and are not in conflict with Green and Smith's top ten principles.

Unlike other studies, we attempt to provide a fairly comprehensive account in five stages of what general developers might encounter while working with cryptography. In the first two stages, we check the status quo of cryptography usage both in industry and open-source projects. We provide eight main categories of problems in using cryptography. Thereafter, we pinpoint the prevailing challenges, mined from Stack Overflow, that developers generally encounter in cryptography. Then, we narrow down the scope to 20 crypto libraries to understand what kinds of problems are more prevalent. In the end, we choose symmetric APIs in a programming language to discern the issues related to such APIs in minute detail.

2.4 Tools

Security tool adoption: Johnson *et al.* conducted interviews with 20 developers to understand the determinant factors why static analysis tools were not adopted by many developers [86]. Participants mentioned reasons such as the high rate of false positives, the way that warnings are displayed, faulty integration of the tool into the development process, lack of detailed explanation of bugs with automatic fixes, and not including understandable configuration

options in the tool for all levels of developers. Other researchers investigated the reasons for a low rate of security tool adoption [13] [14]. They found that organization and team policies affect the usage of security-related tools and larger organizations use security tools more than small ones. The greater adoption of security tools can be influenced by factors such as the culture of the company, security concerns, training, and dedicated security and testing teams. Witschey *et al.* conducted a study to understand what factors affect the usage of security tools [150]. Strangely enough, being more concerned about security did not lead to greater security tool usage while having training or academic background in the security field did.

Tool support: CryptoLint, developed by Egele *et al.*, checks real-world Android applications for the violation of six security rules [43]. They succeeded to find 10 327 of 11 748 Android applications analyzed by CryptoLint that use cryptographic APIs exposed to at least one mistake. CryptoLint is not yet open source. Yong Li *et al.* proposed iCryptoTracer, which performs a combination of static and dynamic analysis on iOS applications [100]. Their research showed that nearly 65.3% of the examined applications suffered from a cryptographic misuse. Rahaman *et al.* describe CRYPTOGUARD, a deployment-quality static analysis tool to identify Java cryptographic misuses [127]. They provide contextual refinements for false positive reduction, on-demand flow-sensitive, and context-sensitive analysis. Kim *et al.* introduced a code search engine that merges results from API documents with code example summaries, mined from the web [90]. However, it is not tailored to mine secure examples. Krüger *et al.* presented a tool called CogniCrypt, an Eclipse plugin that empowers developers to identify cryptographic misuses in Java code[92].

Our findings also provided corroborative evidence that developers generally do not welcome security tools. The findings of our survey showed that only one-fifth of developers use static analysis tools. The root causes of this issue can be multifaceted and are outside of this study's scope.

2.5 Developer performance

The significance of correctly employing cryptography and obtaining professional help from online sources has been discussed by numerous authors in the literature. Sifat *et al.* studied three popular online sources, *i.e.*, crypto Stack Exchange, Security Stack Exchange, and Quora, to find out the common challenges concerning implementing security in data transmission [85]. They uncovered that Transport Layer Security (TLS) is the most discussed technique and users' interest has increased in this topic over the years. Yang *et al.* carried out a large-scale analysis of security-related questions on Stack Overflow and reported a classification of five topics [153]. A recent study conducted by Meng *et al.* has recognized the challenges of writing secure Java

code on Stack Overflow [107]. Their results provide compelling evidence to the fact that the security implications of coding options in Java, *e.g.*, CSRF tokens, are only partially grasped by many developers. A study confirmed that developers are uncritically using the insecure code snippets found on Stack Overflow [53]. The aforementioned findings jeopardize the security of software[54]. We observed that relying on poorly validated responses on online forums was inextricably linked to software systems' security implications. Acar *et al.* assigned 307 active GitHub users to complete several security-relevant programming tasks and surprisingly, found no statistically significant differences concerning functional correctness and security perception among the participants who registered their status as a student, professional developer, or those who had security background [3]. Interestingly, they found that years of experience was not an effective factor for security perception. Oliveira *et al.* designed a study for 109 developers to use some APIs that had some blind spots, *i.e.*, containing underlying causes to misuse an API, and some easy to use ones [115]. The results show that developer expertise and experience did not predict their ability to identify blind spots. In another study, the outcome of an experiment with 54 professional and inexperienced developers for writing security-related code explains that development experience is not a decisive factor for code security [2]. Nadi *et al.* conducted two surveys and asked developers about the issues they face when working with crypto tasks [112]. The participants mentioned several types of issues including lack of documentation, difficulty in API use, and indirection between the APIs and the underlying implementation. The authors also realized that developers from various knowledge level groups still face the same types of issues in cryptography. Robillard *et al.* conducted surveys and interviews with Microsoft developers, and realized that poor documentation is a major learning obstacle for learning APIs [130]. Gorski *et al.* conducted a controlled online experiment with 53 participants to learn the effectiveness of API-integrated security advice which informs about an API misuse as guidance close to the developer [61]. They achieved a high rate of success as 73% of the participants who received the security notice fixed their insecure code accordingly. Kafader *et al.* developed a fluent API called FluentCrypto to facilitate the secure and correct use of cryptography in Node.js [88]. They solution provides task-based approach and hides the low-level complexities from the developer.

Developer mapping: A series of recent studies have focused on profiling developer expertise either on single or multiple platforms [152] [28]. A common concern in profiling developer expertise cross-platforms is to track developer identity, as developer activity can be dispersed from one platform to another [91]. For instance, Zhang *et al.* used the developer email and the hashing approach to identify the same developer with the same email address on another platform [159]. Yung *et al.* looked into the challenge of expert finding with the Topic Expertise Model (TEM) [102]. Their approach jointly modeled topics and expertise by combining textual content model and link structure analy-

sis. Tian *et al.* proposed a novel methodology to identify experts who utilize various user attributes and related platform-specific information, for instance, high-quality Stack Overflow answers in specific programming technologies and high-quality projects measured using source code metrics [141]. Sajedi *et al.* checked the features that overlap in GitHub and Stack Overflow [15]. They defined three high-order metrics related to both networks (*i.e.*, development, management and popularity). Their findings revealed moderate and strong correlations between the derived metrics within each platform. Vasilescu *et al.* analyzed the differences of 46 967 active users both on Stack Overflow and GitHub to understand Stack Overflow's involvement of GitHub's developers [146]. They discovered that the users who provide more answers on Stack Overflow tend to have a high number of commits. Their results imply that users with a high number of commits on GitHub have a greater tendency to take the role of a "teacher" instead of asking more questions on Stack Overflow. Vadlamani *et al.* focused on perceiving what constitutes the notion of an expert developer and what key elements affect developer contribution[142]. They conducted a survey with active software developers both on Stack Overflow and GitHub. Their results show that developers consider personal drivers to be more critical than professional factors for GitHub contribution, and the majority of experts participate in both private and public repositories.

Developer concern: Research indicates that some organizations use external resources,*e.g.*, penetration testers, to encourage developers to pay extra attention to security in development, however, without strong support, the motives tend to lose priority compared to the important functional requirements [151]. Likewise, managers sometimes are obliged to make vital decisions, such as releasing the code with some known problems, due to business forces [140].

In this study, we take a different approach in order to observe whether developer experience affects developer performance. We conduct the first mapping of experts' crypto practices between two popular software platforms, Stack Overflow and GitHub, to learn if there is a connection between the activities performed in the two platforms. Furthermore, we contact the crypto experts to ask for opinions and views regarding the root causes and plausible remedies of misusing crypto APIs. The results of our survey validate that developer concerns are incongruent with their practices.

Chapter 3

Cryptography in the wild

Cryptography, hereafter crypto, is widely employed nowadays to safeguard against any threat posed to the integrity, confidentiality, authentication, and non-repudiation of our data. For instance, in the absence of cryptography it is infeasible to use any wireless/wired network and perform any online banking activity. Unfortunately, prior research has showed that crypto has often been used insecurely causing undesirable outcomes for businesses and major companies. For instance, researchers detected a weakness in the generation of session-ids in Java Servlet 128-bit and introduced an efficient practical prediction algorithm to impersonate a legitimate client [65]. Similarly, in 2008, a researcher found that the random number generator in Debian's OpenSSL package is predictable.[1] More recently, researchers found in the widely-used online meeting software, *i.e.*, Zoom, that in each Zoom meeting, a single AES-128 key is employed in ECB mode, which is not recommended, by all participants to encrypt and decrypt audio and video.[2]

Previous studies have shown that crypto has been misused in a wide range of software systems. For instance, Wickert *et al.* developed a static analysis tool that discovered misuses in Python crypto APIs [149]. They examined 895 popular Python projects from GitHub and 51 MicroPython projects for embedded devices, reporting that 52.26% of the Python projects had at least one misuse and that, the design of crypto APIs in python prevented developers from misusing APIs. Likewise, by developing CryptoLint to check Android applications, Egele *et al.* discovered that 88% of analyzed applications had at least one crypto mistake [44]. As regards crypto flaws in popular software, Lazar *et al.* studied 269 cryptographic vulnerabilities reported in the CVE database from 2011 to 2014 [98]. Their findings suggested that only 17% of the bugs are rooted in cryptographic libraries with catastrophic consequences and the remaining are misuses of cryptographic libraries. This emphasizes

[1] https://www.debian.org/security/2008/dsa-1571

[2] https://citizenlab.ca/2020/04/move-fast-roll-your-own-crypto-a\
-quick-look-at-the-confidentiality-of-zoom-meetings/

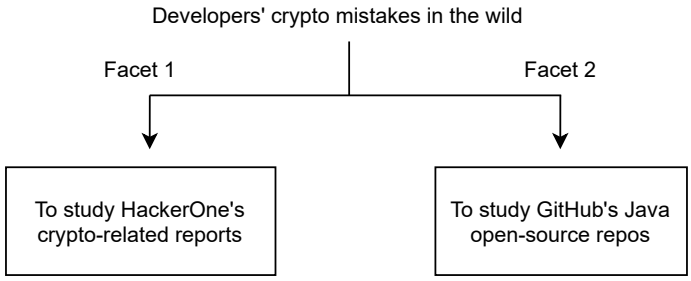

Figure 1: The methodology followed to answer the first research question

that careful examination of recent crypto vulnerabilities/misuses magnifies the importance of this area and why this domain requires further research.

3.1 Study design

In this chapter, we aim to examine the current status of crypto misuses/vulnerabilities. To that end, we address the first research question (RQ1) of our study: *What crypto mistakes do developers make in the wild?*, from two facets, illustrated in Figure 1.

Facet 1 → We reviewed recent vulnerability reports on the HackerOne bug bounty platform to understand what types of cryptography vulnerabilities exist in the wild. We extracted eight themes of vulnerability and discussed their real-world implications and mitigation strategies. SSL-related attacks received the most reports due to a lack of awareness or attention among professionals about SSL/TLS certificates.

Facet 2 → We used a static analysis tool to analyze 489 open-source Java projects that rely on Java Cryptography Architecture, and manually inspected 50% of the analysis results to assess the tool's performance. We learned that 85% of crypto APIs are misused and among the manually investigated records, 74 records (6%) were flagged as rejected, meaning that the records were mistakenly marked as misuses by the tool.

In the following, we explain how we collect the data and analyze them for each of the aforementioned facets.

3.1.1 HackerOne

Various bug bounty platforms exist such as HackerOne[3], Bugcrowd[4], Wooyun[5], and Synack[6], wherein highly trusted security researchers discover security vulnerabilities in companies' bounty programs. Although some studies analyzed the Wooyun platform, the website has not been accessible since July 20, 2016 [160].[7] HackerOne, Bugcrowd, and Synack are among the three most searched keywords, respectively, for bug bounty platforms on Google. [8] Despite the advantages of such platforms, we could only consider the ones in which there is a tendency to disclose vulnerability reports and the reports must entail adequate explanations. We conducted a preliminary check on HackerOne, Synack, and Bugcrowd to observe which platforms present detailed vulnerability reports. Bugcrowd enables access to bug bounty programs, detailed user statistics, the number of vulnerabilities discovered, and the average payout in each program. However, of 257 bug bounty programs, there were only 17 programs that had limited disclosed reports (*i.e.*, 54 reports). Similarly, the Synack platform also does not provide access to the detailed reports of discovered vulnerabilities. In contrast to the previous two platforms, HackerOne permits hackers as well as organizations to study a vast number of preceding vulnerabilities' reports that are marked as disclosed. We, therefore, decided to study HackerOne.

Recent bug reports are accessible on the hacktivity page[9] and search filters allow one to navigate disclosed reports. A more detailed inspection showed that HackerOne employs the GraphQL query language to retrieve data from remote APIs. The *data.hacktivity_items.pageInfo.endCursor* property in the retrieved JSON is a key that can be used for retrieving the next page's content. We wrote a Python script to automatically fetch disclosed reports from two APIs on HackerOne. The accumulated data are publicly available.[10]

The collected data contain 19 fields (See Table 1). The title and URL fields are about the headline and the full web address of the vulnerability report. The *to_whom* and *by_whom* fields describe the company that receives the vulnerability report and the person who found the vulnerability. Regarding the vulnerability, the chosen severity rating and score are stored in the *severity_rating* and *severity_score* fields. The *user_reputation* field describes the points received or lost based on report validity. The *user_rank field* explains user ranking based on the earned reputation. The *user_signal field* stores information for identifying experts who have had consistently valid reports. The *user_impact* describes the expert's activity in terms of severity

[3] `http://hackerone.com`
[4] `https://www.bugcrowd.com/`
[5] `http://www.wooyun.org/`
[6] `https://www.synack.com/`
[7] `https://en.wikipedia.org/wiki/WooYun`
[8] Top bug bounty platforms searched on Google
[9] `https://hackerone.com/hacktivity`
[10] `http://crypto-explorer.com/hackerone/data.sql`

level. The *user_percentile* and *user_imp_percentile* fields help the security experts to compare their percentile signal and impact rank to other experts on the platform. The *report_state* shows the decision made for the report. The dates for when the report is submitted and disclosed are specified in the report_date and disclose_date fields. The weakness field reports the vulnerability type of the report and is our field of interest for finding cryptography-related vulnerabilities. The award field specifies the amount of money given to security researchers, and the summary field reflects the explanation of security experts for the identified vulnerability.

Table 1: The structure of the collected data from HackerOne

Field	Description
Title	the report's title
URL	the url of the report
To_whom	the company to which report is sent
By_whom	the person who sent the report
Username	the username of the sender of report
User_profile_pic	the user's profile picture
Severity_rating	the severity of the report (low/medium/high)
Severity_score	the score of severity (0 ∼10)
User_reputation	the reputation of the user
User_rank	the rank of the user
User_signal	the average reputation per report
User_percentile	the user signal percentile relative to others
User_impact	the average reputation per bounty
User_imp_percentile	the user impact percentile relative to others
Report_state	the state of the report (duplicate, informative, not-applicable, resolved, spam)
Report_date	the date that the report was submitted
Disclose_date	the date that the report was disclosed
Weakness	The weakness type assigned to the report
Award	The amount of money given to the reporter
Summary	The description of how the vulnerability is discovered

Analysis: Our aim is to analyse vulnerability reports related to cryptography. Hence, two reviewers reviewed all the weaknesses, *i.e.*, 120 weaknesses, from the *weakness* field and extracted crypto-related keywords. Each weakness commonly consists of 3 or 4 words explaining the corresponding category of vulnerability, *e.g.*, *Brute Force, or Command Injection - Generic*. After cross-checking the extracted keywords, they resolved their disagreement and achieved a consensus on two keywords, namely "crypto" and "encrypt". Table 2 describes the selected weakness types and their associated number of reports in the data. The "Cryptographic Issues - Generic" type has the highest number of reports (*i.e.*, 161) and the "Reusing a Nonce, Key Pair in Encryption" type has the lowest number of reports (*i.e.*, 1) among the nine weakness types. In total, there are 187 vulnerability reports whose weakness type contained these two keywords. Of the 187 reports, there are 33 vulnerability reports in 2014 and there is a peak in 2017 with 52 vulnerability reports (See Figure 2). In 2018, 2019 and 2020, the number of crypto reports dropped

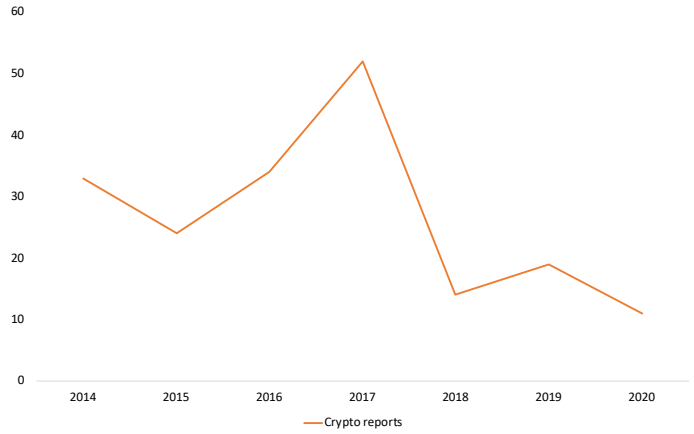

Figure 2: The total number crypto reports per year

Table 2: The selected weakness types and the associated number of reports

Weakness	**# reports**
Cryptographic Issues - Generic	161
Weak Cryptography for Passwords	7
Use of a Broken or Risky Cryptographic Algorithm	4
Inadequate Encryption Strength	3
Missing Encryption of Sensitive Data	3
Missing Required Cryptographic Step	3
Use of Cryptographically Weak Pseudo-Random Number...	3
Use of Hard-coded Cryptographic Key	2
Reusing a Nonce, Key Pair in Encryption	1

to 14, 19, and 11, respectively. However, the data for 2020 may not be complete as we have not updated the data since the end of 2020 and more reports could be marked as disclosed. To increase the number of reports from other weakness types, we searched 40 crypto keywords, identified in the previous study [77], in the reports' summary. We found 221 unique reports containing at least one keyword in their summary.

Afterward, we used thematic analysis, a qualitative research method for finding themes in text [30], to find the frequent themes in the reports. We did not prepare a list of themes beforehand to assign the reports to the suitable themes. We derived the themes by finding patterns, commonalities over the course of the thematic analysis. The two reviewers individually read the selected reports (the title and summary fields), extracted the core problems of each report, built a list of themes, and assigned each report to a suitable theme. Each of them improved the extracted themes by reviewing the reports iteratively. Each reviewer stopped after three rounds of refinement and extraction since they noticed that the iterative process no longer adds anything of significance to the analysis. However, of 187 crypto-related reports, 33 reports did not have sufficient explanation, and accordingly we omitted them. Of 221 reports whose summary contained crypto keywords, only 19 reports (*i.e.*, 8%) were related to cryptography. In general, we omitted the reports that were

marked *not applicable* by the companies. Finally, the reviewers discussed the themes and associated reports with each other. We calculated Cohen's kappa, a commonly used measure of inter-rater agreement [39], between the two reviewers and obtained 71% Cohen's Kappa score, which manifests a substantial agreement between them. The reviewers re-analyzed the specific reports in a session where they had disagreements and achieved a consensus. They realized that their wording mechanism of themes differed in some cases and thus, unified themes were devised.

3.1.2 Java projects

We started with a set of projects identified in previous work [94] that used JCA APIs. We used GitHub APIs to fetch the collaborators of these projects and check what other Java projects they contributed to, which helped us to collect more projects. Next, we used the GitHub search code API to check whether a Java project uses any of the JCA APIs, such as *Cipher*, or *KeyStore*. The GitHub search code API limits the number of requests to 30 per minute, therefore we executed this phase in parallel with different GitHub accounts. If the project is forked, then we cloned the original repository. Forked projects significantly increase the chance of having duplicate projects in the dataset. We also did not limit our search criteria based on the number of project forks or stars, as we were only interested in collecting crypto API uses regardless of factors such as project size, popularity, or the degree of recent activity.

We compiled each project in preparation for the static analysis phase. We used a bash script to check for the existence of the build file (POM) in the project's path and then proceed to compile the project using Maven. We excluded any projects that cannot be compiled due to unresolved dependencies.

Analysis: We used an open-source static analysis tool called CogniCrypt, which detects known misuses of JCA APIs [93]. It uses a set of rules to analyze method-call patterns, parameter constraints, and secure compositions of cryptography-related classes. We chose CogniCrypt as it supports a wide range of APIs, is open-source, and relies on an extensive rule set created by crypto experts. We extended the tool to collect and report information regarding at what line number each API is used and in which user-defined method the API use occurred.

We fed each project's binary code (*i.e.*, *.class* files) to CogniCrypt. Most projects were analyzed within 10 minutes. Accordingly, we aborted lengthy analyses that take more than 15 minutes. Ultimately, 48 analyses were terminated.

For every successful analysis, we used the GitHub API to obtain metadata of each project, *i.e.*, the number of stars, the number of forks, the creation and the last updated date of the project.

Schema: We used a bash script to extract information from the generated analysis reports with the help of regular expressions. We presented the

extracted values in the CryptoMine dataset as a comma-separated CSV file. Each record describes a single crypto API use. Table 3 presents each field and its description in a record. Each data record represents meta-information about a crypto API use in a project such as line number of the API use, Java file path containing the API use, or project's address on GitHub.

Table 3: Fields of each API use in CryptoMine

Field	Description
PS_url	project's address on GitHub
Star_count	number of stars of a project
Fork_count	number of forks of a project
Creation_date	creation date of a project
Updated_date	last updated date of a project
Last_visited	the last time we checked a project
File_path	file path containing a crypto use
S_object	status of a use (0 means a misuse, otherwise it is 1)
API_name	name of the crypto API
Line_number	line number of the crypto use
User_method	the user-defined method where the crypto API is used
Misuse_type	a string referring to the type of the crypto misuse (wrong type, wrong object, wrong constraint, incomplete operation, incomplete order)
Misuse_desc	the description of the crypto misuse
Manual_check	the manually checked status of an API use (Accepted, Rejected, Unvalidated)

The *user_method* field provides information about the user-defined function where a crypto API use exists.

The *misuse_type* field can represent any of the following five types provided by the static analysis tool. The "wrong type" means when a developer incorrectly uses a certain reference type. For instance, the constructor of PBEKeySpec requires the password to be passed as a character array, and should not be as a string object. The "wrong object" occurs when an object is passed to another object but not in the correct way to fulfill expected security requirements. The "wrong constraint", which is a common misuse type, occurs when a developer selects wrong values for integer or string objects to pass to a crypto API, like key sizes, algorithm names, or iteration counts. The "incomplete operation" indicates the whole path for the desired cryptographic purpose is not fulfilled, *e.g.*, failing to call PBEKeySpec.clearPassword(). Finally, the "incomplete order" misuse shows that the expected method call sequence to be made is incorrect, *e.g.*, failing to call to init() in the *Cipher* API.

The *misuse_desc* field explains for what reason, which is provided by the tool, a crypto API use violates CogniCrypt's rules. The *manual_check* field indicates the manual cross-validation status of an API use. In case of approval, *i.e.*, agreement with the tool, we set the value of the field to *Accepted*,

otherwise, the value is set to *Rejected*. Non-validated records are indicated by *Unvalidated*. Lastly, interested researchers can request to receive the cloned version of the projects.

Manual investigation: Two reviewers manually checked 1280 records of CryptoMine (48% of the dataset). They relied on their expertise and the CrySL rules provided by the static analysis tool. The CrySL rules determine the secure uses of a crypto API. The reviewers examined the 1280 records separately and finally cross-check their individual judgments. In case of conflicts, they referred to the tool's rules and discussed them.

3.2 Results and discussions

In the following, we present and discuss our results regarding the two facets of this chapter.

3.2.1 HackerOne

We present each theme of vulnerabilities, their prevalence in HackerOne's reports, and explain mitigation strategies suggested by the literature.

SSL-related attacks

POODLE: The POODLE vulnerability can expose a man-in-the-middle possibility for attackers when using SSL 3.0 or under some circumstances for the TLS 1.0 - 1.2 protocols. Attackers are able to make 256 SSL 3.0 requests to reveal one byte of encrypted data. *Mitigation:* In order to mitigate the POODLE attack, SSL 3.0 must be disabled [109]. If disabling SSL 3.0 is not practical, the use of the TLS_FALLBACK_SCSV cipher suite ensures that SSL 3.0 is used only when a legacy implementation is involved, and thus, attackers are not capable of forcing a protocol downgrade. [109].[11]

Sweet32: The Sweet32 vulnerability enables attackers to reveal small parts of an encrypted message produced by 64-bit block ciphers, such as Triple-DES and Blowfish, under limited circumstances for TLS, SSH, IPsec and OpenVPN protocols. *Mitigation:* changing the default ciphers, such as 3DES or Blowfish, avoiding legacy 64-bit block ciphers, and selecting a more secure cipher like AES approved by NIST [22].[12]

DROWN: The DROWN vulnerability affects the OpenSSL library, SSL, and TLS on servers wherein SSLv2 connections are allowed [11]. Attackers can passively decrypt collected TLS sessions when a server supports SSLv2 as a Bleichenbacher padding oracle. The DROWN exploitation entails a chosen-ciphertext attack in order to steal a session key for a TLS handshake. *Mitigation:* server administrators must upgrade OpenSSL to the latest version

[11] https://tools.ietf.org/html/rfc7507
[12] https://sweet32.info/

and disable SSLv2, *e.g.*, use the following command in Apache webserver: *SSLProtocol All -SSLv2*.[13]

BREACH: The BREACH vulnerability is a weakness in HTTPS when HTTP compression is used [124]. The attack is agnostic to the version of TLS/SSL and applicable to any cipher suite. The attacker can obtain information about secrets in a compressed and encrypted response by attacking the LZ77 compression. In practice, the attacker injects random guesses into HTTP requests and measures the size of the compressed and encrypted responses to collect the smallest response sizes, meaning that the random guess matches the secret. *Mitigation:* to make the attack infeasible, HTTP compression must be disabled [132]. There are other countermeasures such as masking the secret with a one-time random value with each request and resulting in producing a new secret every time or to monitor and enforce request rate-limiting policy to discern nefarious visitors from genuine visitors [132].

SSL stripping: In an SSL stripping scenario, attackers downgrade the interaction between the client and server into an unencrypted channel to orchestrate a man-in-the-middle attack. There are several ways such as creating a hotspot, conducting ARP spoofing, and DNS spoofing to lure victims into the wicked network [82]. *Mitigation:* the SSL pinning technique is about avoiding man-in-the-middle attacks by checking the server certificates with a pinned list of trustful certificates added by developers during the application development phase [116] [50]. Other effective approaches against SSL stripping include: the HSTS (HTTP Strict Transport Security) header [81], History Proxy [114], and static ARP table [38].

Freak: The Freak vulnerability empowers attackers to intercept secure HTTPs connections between clients and servers and persuade them to employ "export-grade" cryptography which presents out-of-date encryption key lengths [21]. The exploitation entails downgrading the RSA key length to 512-bit export-grade length in a TLS connection. *Mitigation:* upgrade OpenSSL and the EXPORT grade ciphers must be disabled on the client side.[14] Moreover, it is necessary to use Perfect Forward Secrecy (PFS) cipher suites for key exchange,*e.g.*, Diffie-Hellman(DH) or Elliptic Curve DH in the ephemeral mode [5].

BEAST: The BEAST vulnerability provides a man-in-the-middle opportunity for attackers so as to reveal information from an encrypted SSL/TLS 1.0 session. The attack phase necessitates an adaptive chosen plaintext attack with predictable initialization vectors (IVs) and a cipher block chaining mode (CBC) [45]. *Mitigation:* enable TLS 1.1 or preferably 1.2 which employ random IV and can also switch to using RC4 instead of using block ciphers [132].

Certificate mis-issue: The Certification Authority Authorization (CAA)

[13] https://www.openssl.org/blog/blog/2016/03/01/an-openssl-users-guide-to-drown/
[14] https://access.redhat.com/articles/1369543

DNS resource record allows a domain owner to set which Certificate Authorities are permitted to issue certificates for the domain. In that case, other CAA-compliant certificate authorities should refuse to issue a certificate. *Mitigation:* server managers should use the CAA feature in order to prevent issuing certificates by other unauthorized CAs [66].

On HackerOne: There are various type of attacks against SSL in the vulnerability reports. The highest number of references belong to the POODLE and SWEET32 attacks that appeared in 13 and 7 reports. The Breach and Drown attacks each appeared 4 times in reports and the SSL pinning and Beast attacks each discussed 3 times in reports. Other types of attacks, such as SSL stripping, freak attack, CBC cut and paste attack, invalid curve attack, divide-and-conquer session key recovery, appeared only once in all reports. There are 25 reports that are about certificate-related issues, such as the CAA record or validation of a certificate. One of the major issues that security experts found was the missing DNS Certification Authority Authorization (CAA) record. Failure in certificate validation in mobile apps and insecure enabled RC4 cipher suites are the other evident reasons in the reports.

Weak crypto defaults

Cryptographic algorithms require various parameters which developers should provide. If an insecure argument or a weak crypto hash function, *e.g.*, MD5, is used, it may pose a severe threat to the security of a program. For instance, the use of insufficiently random numbers in a cryptography context leads to predictable values, or impersonating other users and accessing their sensitive information. Moreover, using a weak key length can also weaken the security of crypto algorithms to withstand brute-force attacks. *Mitigation:* testing is an effective way to ensure that the implementation is able to pass the basic security tests, *e.g.*, the National Institute of Standards and Technology (NIST) offers test vectors for a number of cryptographic primitives.[15] Moreover, NIST provides cryptographic standards and guidelines which is a reliable source for finding the right parameters and algorithms for various circumstances in cryptographic scenarios.[16] Employing static analysis tools in order to check crypto misuses in code snippets can immensely help developers in detecting crypto API misuses in the development phase, *e.g.*, CryptoLint [44] and MalloDroid [49].

On HackerOne: There are 25 reports that contained weak crypto defaults as one of their main issues. Experts found that weak encryption, *e.g.*, 512-bit RSA key or MD5, could lead to remote command execution, bypassing the download restriction for confidential files, decrypting data sent over SSH, gaining local file inclusion, and corrupting or modifying files on the server.

[15] http: //csrc.nist.gov/groups/STM/cavp/

[16] https://csrc.nist.gov/projects/cryptographic-standards-and-guidelines

There are six reports that security experts found bugs due to the existence of weak random number generators in systems. For instance, a bug could give access to the attacker to obtain access to OAuth access_token as a result of using PRNG instead of SecureRandom. In a report, a security researcher noticed that the target uses the TLS_RSA_WITH_3DES_EDE_CBC_SHA cipher, which is not marked as weak on SSL labs.[17] In four reports, the specified key size was problematic. Another security researcher noticed that using a specific cryptographic function reduces the permutations of cryptographic keys. As a result, reduced permutations boost the chances of IV re-use. In two reports, the short length of keys, *e.g.*, 64 bit, increased the chance of finding key collisions and conducting brute force attacks.

OpenSSL bugs

There are implementation bugs in crypto libraries that jeopardize the security of software systems using such libraries. What's worse, the software developers commonly cannot fix such vulnerabilities, and identifying them are beyond developer responsibility. Such flaws can be extremely dangerous as they expose thousands of software to security risks. *Mitigation:* applying formal verification methods to examine the security properties of a cryptographic protocol can be considered, such as Cryptol [47] and PCL [41]. Furthermore, fuzzing approaches can also be part of library test suites or continuous integration in order to run before any versions are released, *e.g.*, the TLS-attacker framework is being used in MatrixSSL and Botan libraries [138].

On HackerOne: Security experts found 25 bugs in the OpenSSL library and in total, they received a $24 500 award. Security researchers reported vulnerabilities specifically to the implementation of OpenSSL. For instance, a security researcher reported that there is a mismatch in accepting a nonce value for the AEAD cipher. Other vulnerabilities concerned about consuming excessive resources or exhausting memory, recovering key on Diffie–Hellman small subgroups, performing padding oracle in AES-NI CBC MAC check, heap corruption, out-of-bounds read, and denial of service attacks.

HTTP/HTTPs mixed content

The improper way of using HTTPS and mixing it with insecure network protocols, *e.g.*, HTTP or WebSocket (WS), can bring about undesirable outcomes for the website. Attackers can eavesdrop and conduct complicated attacks, *e.g.*, CRIME, JavaScript execution, cookie stealing, due to the unavoidable mistake of combining HTTPS and HTTP [37]. *Mitigation:* all the traffic should go through secure channels and the features that enforce secure connections on browsers, devices, and servers must be utilized. There are a number of HTTP headers that enforce and log the usage of HTTPS, namely

[17] `https://www.ssllabs.com/`

Content-Security-Policy-Report-Only and Upgrade-Insecure-Requests.[18] The latter HTTP header necessitates that the browser must upgrade all insecure URLs before making any requests. For establishing secure WebSocket communications one must use the "wss://" instead of the "ws://" scheme, which utilizes port 443 [52].

On HackerOne: There are 22 reports in which security experts observed an insecure redirect from HTTPS to HTTP, lack of HTTPS on the login page, downloading resources from insecure channels, compromising HTTPS by loading resources from non-secure sources, downgrading from HTTPs to HTTP, and invitation reminder emails including HTTP links.

Timing attacks

Timing attacks exploit execution time differences. A timing attack is possible when a password check module immediately returns "false" as soon as the first character in the supplied password is different from the stored one. The attacker can observe the time it takes the system to respond to various queries until the correct password is extracted. *Mitigation:* in the aforementioned scenario, the developers should compare all of the characters before returning true or false. Returning an early response will leak information. Likewise, when the developers compare strings of equal length and drop the comparison when one string is longer or shorter causes information leakage about the secret string length. There are other preventive approaches, for instance, eliminating cache information leakage, including "random noise" into the computation, constant time implementations in the code, or using special APIs (*e.g.*, hmac.compare_digest in Python[19]) for checking hashed passwords[83] [136] [59].

On HackerOne: There are 11 reports that the timing attacks were the root cause of the vulnerability. Security researchers noticed that using == operator performs a byte-by-byte comparison of two values and once the two differ it terminates. In other words, the longer the process takes time, the more correct characters the attacker has guessed.

Hard-coded secrets

Hard-coded secrets can provide vital information for attackers in order to simply authorize themselves with privileged access in a software system. Such information can be divided into three categories, namely passwords/tokens, hard-coded usernames, and hard-coded private cryptographic keys. *Mitigation:* one advantage of static analysis tools is to prevent developers from using the same cryptographic key multiple times, hard-coded cryptographic keys for encryption or hashes [134]. The Common Weakness Enumeration (CWE-798)

[18] `https://www.w3.org/TR/upgrade-insecure-requests/`

[19] `https://docs.python.org/3/library/hmac.html#hmac.compare_digest`

Table 4: The eight crypto themes on HackerOne, underlying causes, and mitigations

Themes	# reports	Underlying causes	Remedies
SSL-related attacks	58	Using insecure SSL versions *e.g.*, 2.0, 3.0 and TLS 1.0/1.1	Upgrade to more secure protocols (TLS 1.3) Upgrade OpenSSL to the latest version Avoid using 3DES, RC4,and Bluefish
Weak crypto defaults	25	Using wrong parameters and hashing algorithms (MD5, SHA-1) Short keys Insecure random number generators	Use stronger hashing algorithms *e.g.*, SHA-2 and SHA-3 Use guidelines to find the recommended key sizes Use cryptographic PRNG
OpenSSL bugs	25	Implementation flaws	Formal verification methods Fuzzing approaches
HTTP/HTTPs mixed content	22	Loading third-parties that have no SSL Having no measures to check the mixed-content	Install SSL certificate for the website Enforce HTTPS connections by the Upgrade-Insecure-Requests and Content-Security-Policy-Report-Only HTTP headers
Miscellaneous attacks	12	Using normal hashing algorithms (MD5) and prepending a message The KRACK attack Using early versions of RSA padding	The usage of authenticated encryption *e.g.*, AES-GCM or RSA-OAEP The usage of standard HMAC Upgrading router firmware
Timing attacks	11	Byte-by-byte comparison of two values	The usage of specific APIs to check two hashes (*e.g.*, constant_time_compare in Django) Including random-noise into computation
Hard-coded secrets	11	Placing static secret keys or hard-coded passwords in source codes or servers	File system encryption techniques Secure architectural design Manual source code review
HTTP header issues	9	Using cookies without the secure parameters Not enforcing the usage of HTTPS from client-side	The usage of the "HttpOnly" and "secure" flags The usage of the HSTS header

also suggests file system encryption techniques, secure architectural design, manual source code review, and dynamic analysis with manual results interpretation mitigation strategies.[20]

On HackerOne: There are 11 reports that the disclosure of secret keys or hard-coded passwords was the main theme of the reports. Security experts found secret keys or hard-coded passwords in different areas such as an error page, log files, or within a JavaScript source.

HTTP header issues

The attacker can eavesdrop on the HTTP connection to steal the victim's cookie or an XSS attack can retrieve the victim's cookie in case the cookie is not set properly. *Mitigation:* The "HttpOnly" flag in cookies prevents the access of the cookie from the client-side via JavaScript code in case of XSS exploitation. Moreover, to prevent hackers from eavesdropping on the cookies sent between client and server, the secure flag in cookies forces that the cookie must be only sent over an HTTPS connection [31]. The HSTS header resides on the browser, automatically redirects HTTP requests to HTTPS for the target domain, and does not permit a user to override the invalid certificate message on browsers [81].

[20]https://cwe.mitre.org/data/definitions/798.html

On HackerOne: There are 9 reports that contained problems related to HTTP cookies and the HSTS security HTTP header. The main issue related to cookies is tied with not setting the SSL flag and HttpOnly in cookies. We also observed a lack of HSTS for websites and setting improper max-age for the header are frequent issues in the reports. Moreover, experts found that duplicate HSTS headers lead to ignoring HSTS on Firefox and it is feasible to downgrade a chosen victim from an HTTPS connection to HTTP.

Miscellaneous attacks

KRACK: The KRACK vulnerability concerns weaknesses in the WPA2 protocol four-way handshake for wireless networks. The exploitation process includes bypassing the official countermeasure of the 802.11 standard and reinstalling the group key, by combining WNM-Sleep frames with EAPOL-Key frames [145]. *Mitigation:* The users must update their Windows, OSX, Linux, Android, iOS, and firmware of home routers to address KRACK attacks [51].[21] The Wi-Fi Alliance should not solely test products for interoperability, but should also employ fuzzing techniques to detect vulnerabilities [145].

Hash length extension: The hash length extension attack occurs if a developer prepends a secret value to a message, misusing a hashing algorithm in order to build a naive message authentication system [40]. In spite of the fact that the value of the prepended secret is confidential, if the attacker has knowledge about the string and the hash, he can still generate valid hashes. *Mitigation:* Developers must use the standard HMAC and avoid using MD5, SHA1, SHA-256, and other hashing algorithms [89].

Chosen ciphertext: The attack happens when the adversary can obtain the decrypted version of chosen ciphertexts. With the obtained pieces of information, the adversary can make attempts to recover the hidden secret key that was used for decryption [25]. *Mitigation:* To protect from such attacks, one can use secure crypto algorithms such as AES-GCM or RSA-PKCS#1 version 2 (RSA-OAEP) or consider the integrity and authenticity of the ciphertext [25] [87]. Developers must avoid using the RSA encryption standard PKCS #1 v1.

On HackerOne: In this category, we observed several attacks that were repeated in 12 reports. The padding oracle attack was repeated four times, and encryption without authentication as well as insecure data storage each were repeated twice in the reports. The remaining attacks, *i.e.*, the hash extension length attack, the chosen-cipher text attack, the KRACK attack, and the Heartbleed attack appeared only once in the reports.

[21] https://www.kaspersky.com/resource-center/definitions/krack

Summary

In Table 4, we show the eight crypto themes observed in the analyzed reports. The SSL-related attacks theme appeared more than other themes while the least appeared themes are OpenSSL bugs and HTTP header issues. We observe that the key reason for major issues is rooted in avoidance of using secure solutions by developers. For instance, despite the availability of TLS 1.3, there exist servers wherein SSL 3.0 is enabled. Developers commonly overlook the security parameters in cookies and this severely leads to downgrading the security levels of web applications. Such options are clearly elucidated in various reliable sources and even practical examples do exist. [22] [23] Furthermore, there exist encryption/decryption scenarios in which developers should comprehend several intricate concepts such as IV, differences between AES-ECB and AES-GCM, and key lengths. For example, despite the inevitable consequence of deterministic random bit generators (DRBGs), developers paid inadequate scrutiny, leading to several fatal mistakes in software systems, *e.g.*, stealing $5 700 bitcoin due to a bug in entropy use in the Android DRBG. [24] Similarly, IV attacks also jeopardized the security of WEP protocol in wireless networks [139]. Some of the developers' mistakes can be avoided by employing security tools. For instance, the usage of outdated hashing algorithms or hard-coded secrets is easily identifiable with crypto misuse detectors. Nevertheless, there are various factors to dishearten developers from using such tools, *e.g.*, organizational and project-level constraints. Notably, the total number of participants in a study who use a static analysis tool is fewer than one-fifth (*i.e.*, 18) of the total number of participants (*i.e.*, 97) [76].

The themes are a summarized version of what has been improperly implemented in the industrial environment as well as signifying the seriousness of companies in addressing the issues. The provided real-world crypto vulnerability themes can assist developers, who lack knowledge in cryptography, to become more cautious while working with various elements, *e.g.*, OpenSSL, HTTP security headers, or hard-coded secrets, in the development phase. We also provided remedies, corroborated by previous research, for each theme to facilitate the hassle of finding secure practices. Future research should combine the current results with prevalent issues in other similar areas, *e.g.*, authentication, and authorization circumventions.

3.2.2 Java projects

In this section we first report on the current status of crypto API uses in open-source projects, and then present the key messages of developer feedback.

[22] `https://developer.mozilla.org/en-US/docs/Web/HTTP/Cookies`

[23] `https://owasp.org/www-community/controls/SecureCookieAttribute`

[24] `https://arstechnica.com/information-technology/2013/08/google-confirms-critical-android-crypto-flaw-used-in-5700-bitcoin-heist/`

The state of crypto uses

We investigated the use of 15 JCA APIs in 489 projects. We found that only two projects are completely healthy, and remaining 487 projects suffer from at least one crypto misuse. The mean of the project forks is 139, and the median value is 7.5. The mean of the project stars is 348, and the median value is 5.

Among the manually investigated records, 74 records (6%) were flagged as rejected, which means according to the tool's rules and the opinions of reviewers they are mistakenly marked as misuses. For instance, before using the *sign* method in the *Signature* API, a developer needs to call either the *initSign* or the *update* method. However, in some cases, developers used the *update* method in a loop, while the automatic analysis could not recognize it.

Figure 3 summarizes the uses vs. misuses of each of these APIs as well as the total number of each API use in parenthesis. Developers seemingly have severe difficulties in using more than half of the APIs whose correct uses were less than 51%. For instance, the correct uses of five APIs namely, *SecretKeySpec*, *IvParameterSpec*, *KeyStore*, *Cipher*, *MessageDigest*, and *Signature* were at most 10%. In contrast, developers had a promising performance in using the *SecretKey*, *Mac*, *SecureRandom* and *KeyPair* APIs, *i.e.*, at least 90% uses were correct.

Various misuse types may pose threats with different levels of severity depending on how a project is intended to be used, *e.g.*, selecting a wrong constraint MD5 versus skipping to dispose of a crypto object. Table 5 gives information about the distribution of crypto misuse types in the top six most misused APIs. Most of the misuses were of the *ConstraintError* type followed by *RequiredPredicateError* and *TypestateError*. The *ConstraintError* type made up the largest proportion of misuses, showing that developers struggle with choosing correct parameters for crypto APIs. The second most common misuse is the *RequiredPredicateError* type, which means an insecure object is passed to other objects as an arguments. The *NeverTypeOfError* and *IncompleteOperationError* types account for nearly 23% of the total misuse types that exist in all analyzed projects.

The complete analysis results are publicly available via the CryptoMine dataset,[25] which facilitates investigation of the following research questions: (1) What are the most common crypto mistakes, and what should we do to improve learnability in this domain? (2) How do crypto uses evolve in a project? (3) How do the quality characteristics of a project correlate with crypto uses in that project? (4) In what context are crypto APIs commonly used? (5) Why do some developers perform better in using cryptography? (6) What is the performance of the static analysis tool in detecting crypto misuses? (7) What is the benchmark result of comparing several static analysis tools in detecting crypto uses of the dataset's projects? Even though some of

[25] `http://crypto-explorer.com/cryptomine/`

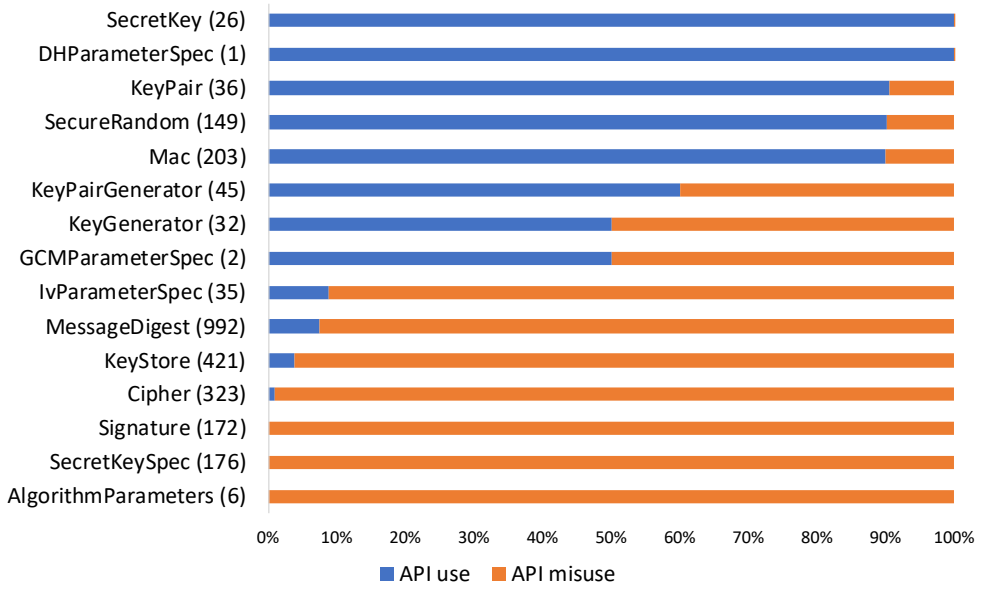

Figure 3: The misuses vs. uses of each API in percentage

the aforementioned research questions might not fit in the scope of this study, it certainly helps to explore relevant areas.

Table 5: Mostly misused APIs with more than 10 misuse types

JCA API	IncompleteOperationError	NeverTypeOfError	TypestateError	RequiredPredicateError	ConstraintError
SecretKeySpec				170	
Signature	5		1	49	26
Cipher	42			83	52
KeyStore	32	160	16		
MessageDigest	72		178		400
IvParameterSpec				30	

3.3 Threats to validity

In the following, we describe the threats to validity of each of explored facets.

3.3.1 HackerOne

To alleviate the errors of report categorization, each reviewer separately checked each bug report, and finally, they cross-checked their results. Besides checking the reports whose weakness field contained the “crypto” and “encrypt” terms, we also analyzed 221 reports whose summary contained 40 crypto-related keywords [77]. However, studying other reports (*i.e.*, 8903) on HackerOne may also uncover new relevant reports. In this work, our preliminary study revealed that Bugcrowd, and Synack do not provide detailed reports of discovered vulnerabilities, and hence, we only focused on vulnerabilities reported

on HackerOne. We cannot lose sight of fact that the number of vulnerability reports in recent years has increased but many could have not yet been disclosed. On HackerOne, studying private or not yet disclosed reports is not possible. Furthermore, studying all the available reports in detail is not feasible since the examined end-points or applications are not accessible to the public. Lastly, the analyzed vulnerabilities reports are associated with distinguished companies, and hence, such companies are more security conscious than the average software firms.

3.3.2 Java projects

It is infeasible to manually identify crypto (mis)uses in source code at large-scale. In contrast to manual analysis, adoption of static code analysis tools can considerably help developers to automatically detect crypto misuses and write more secure code. Therefore, we employed a static analysis tool, *i.e.*, CogniCrypt, in order to assess the status of crypto uses in hundreds of Java projects. The primary reason for this choice is that the tool is open-source and supports a wide range of crypto rules for different APIs, which are easily extendable. However, the CryptoMine dataset does not represent all the JCA APIs and their various usages, *e.g.*, different parameters or method calls. This can be addressed by increasing the number of analyzed projects which contain different usages of crypto APIs. To provide a better level of reliability, we have manually cross-checked 48% of the results in the CryptoMine dataset. Nevertheless, the manual analysis of a large dataset is a non-trivial task and to accelerate the process we invite interested researchers to join us. Moreover, security assumptions may change over a period of time. For instance SHA1, which was judged to be secure in the past, is considered insecure now.

3.4 Summary and conclusion

We were curious to observe what types of crypto flaws security experts find in real-world programs. We extracted disclosed vulnerability reports from HackerOne and analyzed the ones which were labeled as cryptography sensitive report. We found eight themes of crypto vulnerability in 173 reports. We introduced each theme's vulnerability, implications, prevalence on HackerOne, and suggest mitigation strategies. This study showed that developers do not commonly employ secure options in cryptography, leading to severe security vulnerabilities. Voracious readers and developers can learn from the findings, which are based on real bug reports, to avoid making the same fatal mistakes in practice.

We analyzed hundreds of projects in which JCA APIs were used, to observe the status of API use in open-source projects, to learn what crypto misuse types exist, and to investigate the influential factors in misusing such APIs. We found that 85% of the crypto APIs suffered from at least one misuse,

though not all misuses were at the same level of severity. To support the research community, we publicly share the CryptoMine dataset, including the analysis results, and information about each project such as its metadata information, the precise locations of API use, and the safety status of these APIs, to name but a few.

Chapter 4

Crypto hurdles - the API perspective

We witnessed in the previous chapter how cryptography is misused by mainstream developers and what dire consequences these misuses may have. There are various reasons that contribute to the emergence of crypto vulnerabilities. However, broadly speaking, we classify the various issues into two major perspectives: crypto APIs and developer performance. In this chapter, we only focus on the crypto APIs perspective and investigate the latent factors why cryptography is regarded as difficult for developers.

To alleviate the issue of crypto misuse, one might suppose that developers must examine their code with the most recent comprehensive list of security rules concerning crypto APIs. Regrettably, creating and updating such a list can be challenging as the standards might frequently change. This can be due to the fact that security researchers oftentimes discover new vulnerabilities against existing primitives. For example, researchers recommended using stronger cryptographic hash functions owing to a collision attack against SHA-1.[1] Furthermore, the sheer number of crypto libraries presents various approaches to use crypto APIs with varying levels of usability [108]. Analyzing the usage of the *PBEKeySpec* API in Java reveals that there are four imminent threats that must be addressed [56]. Consequently, developers have to struggle with the diversity of parameters and concepts in crypto APIs, as well as with the high complexity of creating a list of security rules.

To clear the confusion, developers commonly seek quick solutions on online question and answer (Q&A) forums and reuse the unconfirmed code snippets [53]. Stack Overflow is used as a knowledge base by many users who are not part of the discussions and this one-way communication may unwittingly contribute to the spread of insecure code, as the community's voting mechanisms does not effectively deter them from answers [144]. Fischer *et al.* demonstrated that 30% of crypto code examples found on Stack Overflow were insecure, and such vulnerable code snippets were reused in approximately 190 000 Android

[1] `https://security.googleblog.com/2017/02/announcing-first-sha1-collision.html`

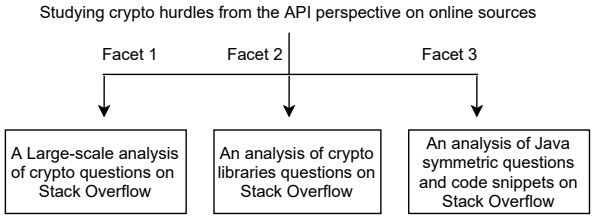

Figure 4: The methodology followed to answer the second research question

applications [53]. What is worse, they understood that the feedback of Stack Overflow's users was not effective in stopping the reuse of insecure code snippets. In another study, researchers noticed that on average posts containing insecure snippets received higher view counts and scores [36]. Yang *et al.* conducted a large-scale analysis of security-related questions on Stack Overflow [153], finding five main categories, i.e., web security, mobile security, cryptography, software security, and system security; however, they did not delve into the challenges of each topic. We believe that detailed scrutiny of such online forums contributes to clarifying the state of developers' problems with crypto APIs.

4.1 Study design

The aim of this chapter is to observe what crypto hurdles developers encounter and enquire about on online sources, especially Stack Overflow. To that end, we address the second research question (RQ2) of our study: *What hurdles are mostly discussed concerning crypto APIs?*, from three facets, illustrated in Figure 4.

Facet 1 → We conducted a large-scale study on crypto-related questions on Stack Overflow. We were interested in the reoccurring cryptography themes. We clustered 91 954 questions with a machine learning technique and manually analyze 383 questions. We found three major themes in crypto-related discussions and reported that developers either have a distinct lack of knowledge in understanding the fundamental concepts, *e.g.*, OpenSSL, public-key cryptography or password hashing, or the usability of crypto libraries undermined developer performance in correctly realizing a crypto scenario.

Facet 2 → We specifically focused on discussions on Stack Overflow that target crypto libraries. We manually studied 500 posts on Stack Overflow associated with 20 popular crypto libraries, inferring that there were ten themes in the discussions in which the majority of posts (*i.e.*, 112) were about encryption/decryption problems and 111 posts about installation/compilation issues of crypto libraries.

Facet 3 → We further narrowed down the analysis to a specific topic rather than analyzing a broad crypto topic as in previous facets. We selected 150 Stack Overflow's posts with respect to Java crypto APIs (JCA, or Java Cryptography Architecture) that were linked to symmetric encryption. In this analysis, our purpose was to understand *(1) what issues are prevalent with JCA APIs while developers have to accomplish symmetric encryption scenarios*, and (2) *the degree to which security risks exist in code snippets and also the question and answer body.* We observed that most of the identified issues were related to the generation of parameters (e.g., keys) or instantiating a Cipher object (e.g., specifying encryption mode). The results revealed that the majority of security risks were notably present in code snippets of questions. Moreover, the identified risks were mainly related to both the use of unsafe encryption modes and constant/static values as a key or initialization vector.

In the following, we explain how we conducted the data collection and analysis phases for each of the aforementioned facets.

4.1.1 General crypto questions analysis

Data extraction:

To collect crypto-related posts on Stack Overflow, we assumed that the attached tags to a question mainly reflect the question's topic. We first used the "cryptography" tag, *i.e.*, *base tag*, to fetch crypto-related posts, *i.e.*, 11 130 posts, with the help of the (Stack Exchange) Data Explorer platform. We found 2 184 tags (*candidate tags*) that occurred in posts together with the "cryptography" tag. However, not all candidate tags were crypto-related *e.g.*, C#.

To find relevant posts with the base tag, we used two metrics to determine which of the candidate tags are exclusively related to the base tag. We introduced the first metric as *affinity* to determine the degree to which a candidate tag (T) is exclusively associated with the base tag (BT). For each T, we used the *posts with tags* function, for brevity pwt(), to calculate the number of posts whose tags contain both T and BT . We used pwt() to obtain the number of posts whose tags contain T. Given these two values, we compute *affinity(T,BT) = |pwt(T,BT)| / |pwt(T)|*, whose result ranges from zero to one.

The smaller the value of the first metric, the weaker the association between T and BT. For example, the "C++" and "encryption" tags each appeared 639 897 and 29 737 times respectively in the entire Stack Overflow. The "C++" tag appeared together with BT 540 times and "encryption" was used 3535 times with BT. The value of affinity for the "C++" tag is 0.0008 and 0.1188 for the "encryption" tag, values which demonstrate a strong affinity for "encryption" and BT.

However, higher values of affinity for some candidate tags do not necessarily indicate tags that are closely related to cryptography. For example, the "s60-3rd-edition" tag appeared once with the base tag and in total 11 times in Stack Overflow. The value of affinity for this candidate tag is 0.09, which is close to the value of the "encryption" tag, even though it appeared only once with the base tag. To resolve this issue, we introduced a second metric, *coverage(T,BT) = |pwt(T,BT)| / |pwt(BT)|*. The second metric indicates the coverage of the BT posts by T. As an example, the value (*i.e.*, 0.00008) of coverage for the "s60-3rd-edition" tag proves that the candidate tag does not exclusively cover the base tag while the "C++" tag covers 0.04 of the cryptography-related questions.

Two reviewers examined various combinations of thresholds for the two metrics, and manually reviewed the resulting tags. We noticed that the thresholds to collect *only crypto-related tags* from the candidate tags (*i.e.*, 2 184) are the ones above the affinity: 0.025 and coverage: 0.005. There are 40 crypto-related tags that fall within the selected threshold domain. The list of crypto-related tags as well as their frequencies are available online.[2] Next, we again used Stack Exchange Data Explorer to extract posts containing each of the selected tags (*i.e.*, 40 tags) but not the base tag, and recorded them in CSV files, which are available online.[3]

Topic modeling: We combined the title and body of a post in order to create a document. We removed duplicate post IDs in multiple CSV files, and finally obtained 91 954 unique documents, without considering when the posts were created. Evidently, each of the documents contained a large number of unnecessary text elements that could produce noise in the output of a topic modeling algorithm. We preprocessed the documents in the following steps: (1) we removed all the code blocks enclosed by the "<code>" tag, (2) we removed all the HTML elements with the help of the Beautiful Soup library,[4] (3) we removed newlines and non-alphanumeric characters, (4) we used the NLTK package to eliminate English stop words from the documents, and finally (5) we used the Snowball stemmer to normalize the text by transforming words into their root forms, *e.g.*, playing converts to play. We found 269 795 stemmed words in total. Finally, we used the CountVectorizer class in Scikit-learn to transform the words into a vector of term/token counts to feed into a machine learning algorithm.

We used Scikit-learn,[5] a popular machine learning library in Python that provides a range of supervised and unsupervised learning algorithms. Latent Dirichlet Allocation (LDA) is an unsupervised learning algorithm based on a generative probabilistic model that considers each topic as a set of words and each document as a set of topic probabilities [24]. LDA has been used

[2] `http://crypto-explorer.com/paper_data/tags.csv`
[3] `http://crypto-explorer.com/paper_data/`
[4] https://www.crummy.com/software/BeautifulSoup/
[5] https://scikit-learn.org/

to discover latent topics in documents in a large number of prior studies [17] [153] [131].

Before training a model, LDA requires a number of important parameters to be specified. LDA asks for a fixed *number of topics* and then maps all the documents to the topics. The *Alpha* parameter describes document-topic density, *i.e.*, higher alpha means documents consist of more topics, and generates a more precise topic distribution per document. The *Beta* parameter describes topic-word density, *i.e.*, higher beta means topics entail most of the words, and generates a more specific word distribution per topic.

The optimal values of hyperparameters cannot be directly estimated from the data, and, more importantly, the right choice of parameters considerably improves the performance of a machine learning model [118]. We therefore used the GridSearchCV function in Scikit-learn to perform hyperparameter tuning to generate candidates from an array of values for the three aforementioned parameters, *i.e.*, *Alpha*, *Beta*, and the *number of topics*. As research has shown that choosing the proper number of topics is not simple in a model, an iterative approach can be employed [161] to render various models with different numbers of topics, and choose the number of topics for which the model has the least perplexity. Perplexity is a measure used to specify the statistical goodness of fit of a topic model [24]. We therefore specified the number of topics from 1 to 25. We also used the conditional hyperparameter tuning for Alpha, which means a hyperparameter may need to be tuned depending on the value of another hyperparameter [103]. We set *alpha* = 50 / *number of topics* and *beta* = 0.01, following the guidelines of previous research [64].

Optimizing for perplexity, however, may not always result in humanly interpretable topics [34]. To facilitate the manual interpretation of the topics, we used a popular visualization package, named pyLDAvis[6], in Python. The two reviewers separately checked the resulting top keywords of the topics, *i.e.*, from 1 to 25, and the associated pyLDAvis visualizations to ensure that the given number of topics is semantically aligned with human judgment.

Data analysis: We computed the required sample size for 91 954 documents with a confidence level of 95% and a margin of error of 5%, which is 383 documents. We then used stratified sampling to divide the whole population into smaller groups, called strata. In this step, we considered each topic as one stratum, and randomly selected the documents proportionally from the different strata. We then used thematic analysis, a qualitative research method for finding topics in text [30], to extract the frequent topics from the documents. Two reviewers carefully reviewed the title, question body, and answer body of each document. Each author then improved the extracted topics by labeling the posts iteratively. We then calculated Cohen's kappa, a commonly used measure of inter-rater agreement [39], between the two reviewers. The results indicated 79% agreement between the two reviewers. Finally, the two review-

[6]https://github.com/bmabey/pyLDAvis

Table 6: The selected crypto libraries and their associated number of posts on Stack Overflow

Tag name	# of posts	Tag name	# of posts
OpenSSL	14 254	libsodium	272
Bouncy Castle	2 799	M2Crypto	263
CryptoJS	1 286	Web Crypto API	215
mcrypt	924	JCA	200
PyCrypto	842	CommonCrypto	199
phpseclib	796	node-crypto	170
Crypto++	713	Botan	117
CryptoAPI	584	Spongy Castle	115
pyOpenSSL	436	SJCL	77
Jasypt	336	wolfSSL	50

ers compared their final labelling results, and re-analyzed the particular posts in a session where they disagreed in order to discuss and arrive at a consensus.

4.1.2 Crypto libraries analysis

We aim at studying posts associated with popular crypto libraries on Stack Overflow. We assumed that discussions related to crypto libraries contain the name of the library as a tag. Hence, we selected the "cryptography" tag, *i.e.*, base tag, to observe what other tags were used together with the base tag. We used Stack Exchange Data Explorer to run a query in order to fetch tags that appeared together with cryptography.[7] We realized that there are 2 184 tags, *i.e.*, candidate tags. The reviewers separately checked each of the candidate tags. Each of the reviewers selected the ones that are crypto libraries. They used the internet to explore a tag in which they had a lack of certainty. Then, they cross-checked the choices and discussed the tags. They arrived at the conclusion that tags that do not represent a crypto library or only provide a particular, limited service in cryptography (*e.g.*, hashing) should not be considered. As a result, there were 6 tags that were eliminated from the list, namely rsacryptoserviceprovider, aescryptoserviceprovider, rijndaelmanaged, bcrypt, javax.crypto, and hashlib. The aforementioned tags are either a crypto class, namespace, or a dedicated module only for hashing. Ultimately, they agreed on a list of 20 crypto libraries, illustrated in Table 6.

Crypto library selection: The selected crypto libraries are all widely used in practice and have been examined in research projects. For instance, six of the selected libraries, *i.e.*, OpenSSL, libsodium, Bouncy Castle, SJCL, Crypto-JS, and PyCrypto, were studied for finding usability issues [120]. MCrypt is the successor to the Unix crypt command, which supports modern encryption algorithms.[8] The phpseclib library offers pure-PHP implementations of SSH2, SFTP, RSA, DSA, and many other algorithms.[9] Crypto++

[7] https://data.stackexchange.com/

[8] `http://mcrypt.sourceforge.net`

[9] `https://github.com/phpseclib/phpseclib`

and Botan are both C++ crypto libraries that support a wide range of crypto algorithms and security protocols.[10] [11] The Microsoft CryptoAPI interface enables developers to employ authentication, encoding, and encryption to Windows-based applications.[12] Jasypt and Java Cryptography Architecture (JCA) are both intended for Java developers, and the latter is part of the Java security API.[13,14] The Web Crypto API is intended to present basic cryptographic operations for web applications and defines cryptographic primitives in a native JavaScript API. [15] The wolfSSL TLS library is a lightweight, C-language-based library designed for IoT, embedded systems, and smart grids.[16] There are also popular OpenSSL wrappers in languages such as node-crypto in Node.js and pyOpenSSL in Python. There have been numerous studies to investigate the security point of view of aforementioned crypto libraries and their strengths and weaknesses were examined [32, 154, 138]. However, the security evaluation of these crypto libraries falls outside the scope of this study.

Manual analysis: In total, there are 24 648 posts that contained the selected crypto libraries' tags. We computed the required sample size for the population with a confidence level of 95% and a margin of error of 4.34%, which results in sampling 500 posts. We then equally selected 25 posts from each tag (*i.e.*, a crypto library). We queried the posts containing a crypto library tag, *e.g.*, OpenSSL, and set the search criteria to "recent activity", so that Stack Overflow returns the recent active discussions. Since we observed questions that are either unanswered or received negative votes, we decided to choose the posts for which the question received at least one upvote and at least one answer. The list of the selected questions is available online.[17]

Thereafter, we employed thematic analysis, a qualitative research method for finding themes in texts [30], to deduce the frequent topics from the chosen posts. Since our study is of an exploratory nature, we did not devise a list of themes prior to studying the posts. Hence, in order to link each post to a suitable theme, two reviewers were responsible to separately study the posts and deduce the main issue (*i.e.*, theme) of the post. The reviewers carefully reviewed the title, question body, and answer body of each post. Despite the fact that each post may entail several crypto concepts, the reviewers' objective was to find the key issue of each post. They employed open coding in which a short explanation label was assigned to each post [99]. Each author reiterated the coding phase three times to improve their deduced list of themes. To

[10] https://www.cryptopp.com
[11] https://botan.randombit.net
[12] https://docs.microsoft.com/en-us/windows/win32/seccrypto/cryptography-portal
[13] http://www.jasypt.org/
[14] https://www.oracle.com/java/technologies/javase/javase-tech-security.html
[15] https://www.w3.org/TR/WebCryptoAPI/
[16] https://www.wolfssl.com/
[17] http://crypto-explorer.com/crypto_libs/

Table 7: The selected weakness types and the associated number of reports

Query	# of posts	# of selected posts
[java] Cipher.getInstance(AES)	3233	126
[java] Cipher.getInstance(DESede)	295	12
[java] Cipher.getInstance(DES) -DESede	300	12

evaluate the inter-rater agreement between the two reviewers, we employed Cohen's kappa to assess the agreement level [39]. Deducing the themes from the posts, the reviewers received 68% Cohen's Kappa score, which indicates a substantial agreement between the two reviewers. Finally, the two reviewers compared the two lists and discussed any disagreements. The two reviewers used different wording for building the list of themes and the total number of themes was not identical. They re-analyzed the particular posts in multiple sessions where they had different views. In some scenarios, they realized that one of the reviewers broke down one theme into several sub-themes, which they then merged if necessary. Ultimately, they agreed on 10 themes for the analyzed posts.

4.1.3 Java symmetric APIs on Stack Overflow

To look for posts on Stack Overflow relevant to Java symmetric APIs, we defined a set of queries. We used the *Java* tag combined with a minimal `Cipher.getInstance()` statement for each symmetric algorithm. We chose the `Cipher` class as it supports a wide range of symmetric and asymmetric encryption algorithms in Java. The `Cipher.getInstance()` statement must be employed in all encryption scenarios using the `Cipher` class. Not all symmetric algorithms supported by JCA are very popular, and hence the corresponding queries returned only a small number of posts, based on which we therefore decided to exclude unpopular algorithms, such as RC2, and instead focus on the three most popular symmetric encryption algorithms: AES, 3DES, and DES. We calculated the sample size of our study with a confidence level of 95% and a margin of error below 8%, which returns 150 posts. Then, according to the number of posts returned by each query, we computed the sample size per query proportionally. The defined queries, as well as the associated number of selected posts, are presented in Table 7.

Stack Overflow enables users to customize their search query based on various criteria, *e.g.*, date of creation or popularity. In order to achieve a balanced view in our sample set, we chose 50% of the sample size from the newest posts and the remaining from the most popular ones since the majority of developers first search for solutions before posting a question on Stack Overflow. We also excluded all posts that did not refer to the implementation of symmetric encryption using the JCA library. A post commonly consists of more than one issue, and hence we included the posts in which at least one issue, question, or piece of advice was found relevant to our scope.

Analysis of Issues

The goal of the first part of the analysis is to answer *"what issues are prevalent with JCA APIs while developers have to accomplish symmetric encryption scenarios"*. To that end, we conducted a qualitative content analysis following the guidelines presented by Mayring [104]. The guidelines enabled us to employ method-integrative approaches that combine qualitative and quantitative elements. These requirements are all key characteristics of Mayring's guidelines. We conducted the analysis in three rounds in which two reviewers were involved. We first summarized the chosen posts to elicit the relevant information (issues and questions), and then conducted two rounds of classification.

In the summarization step, the goal was to extract and record all relevant information so that we did not have to read over the posts again. There were two coders involved in summarization, each of whom independently performed and then discussed their results with the other to create a consistent and more accurate list of records. Notably, we eliminated the posts that did not precisely refer to our scope, namely issues that referred to the conversion of plain text or ciphertext (*e.g.*, character-encoding).

For each post, we recorded an issue or a series of issues and questions with which the original poster was facing. To find the solutions, we first studied the accepted answers of the posts and then looked at other responses. We also recorded the suggested security caveats by the responders. Finally, we had one record for each post that consisted of a short description (*e.g.*, an error message, a shortened form of a question) and a longer description for possible solutions. Afterward, all records were classified in two rounds: first based on technical aspects and then on the requirements that the original poster was not able to meet. In each round, we assigned at most one category to each record.

Technical Aspects

In the first round of classification, we concentrated on the technical aspects of implementing symmetric encryption, *e.g.*, mistakes in the original poster's code that lead to errors. We began with a set of predefined main categories and inductively refined them during the classification. Each time, we outlined a new category and then restarted the classification.

The five main categories that developers must consider when implementing symmetric encryption using JCA are as follows: *Cipher Object Instantiation, Generating Algorithm Parameters, Cipher Object Initialization, Transformation, and Transmitting Algorithm Parameters.* We defined subcategories either to obtain deeper insights (*i.e.*, if an issue targeted only one aspect of the main task) or to provide detailed classification (*i.e.*, dependencies between two properties). The five main categories and their associated subcategories

(depicted in Figure 5) are described as follows.

- *Cipher Object Instantiation*: We assigned this category to all issues and questions referring to an erroneous use of the `Cipher.getInstance(...)` statement. As a parameter, developers must pass a transformation string consisting of:
  - **Algorithm** (mandatory)
  - **Encryption Mode** (optional)
  - **Padding** (optional)

  As a result, we defined the following subcategories:
  - **Dependency Encryption Mode–Padding**: The encryption mode determines whether padding is required or not. We assigned this category to all issues caused by an improper specification of these two properties.
  - **Cipher Object Instantiation–Other**: For issues and questions related to the `Cipher` object instantiation but not any of the aforementioned aspects.
- *Generating Algorithm Parameters*: Depending on the specification of the `Cipher` object, different kinds of parameters are required. For encryption, the developer might need to perform the following tasks:
  - **Key Derivation**: For issues and questions referring to random key generation, password-based key derivation, or key exchange protocols.
  - **Initialization Vector / Nonce Generation**: For issues and questions referring to the generation of the IV or nonce used for the transformation.
  - **Generation of Other Algorithm Parameters**: For issues dealing with generating other necessary parameters, *e.g.*, for Galois/-Counter Mode (GCM) - `GCMParameterSpec`
- *Cipher Object Initialization*: We assigned this category to all issues caused by the misuse of the `init(...)` statement, (*e.g.*, not passing all required parameters).
  - **Dependency Algorithm–Key**: The algorithm determines which data type the key must be stored in. It also defines the permitted key sizes. We assigned this category to issues caused by passing an improper key to the `init(...)` method or questions about this dependency.

  - **Dependency Algorithm/Encryption Mode - IV**: The encryption mode determines whether an IV is required or not. For some encryption modes (*e.g.*, CBC), the IV must be the same size as the algorithm's block size.
  - **Cipher Object Initialization–Other**

- *Transformation*: This category was assigned to all issues and questions targeting the actual transformation methods `update(...)` and `doFinal(...)` (*e.g.*, passing the wrong input parameters or questions about the output).

- *Transmission of Parameters*: As all parameters from encryption must be reused for decryption, they must either be stored or transmitted. This category was assigned to all issues and questions referring to storing, restoring, or transmitting parameters. We further defined the following subcategories:

  - **Key Transmission**: The key must be kept secret.
  - **Transmission of Other Parameters**: IV and other parameters must be transmitted along with the ciphertext since secrecy is not a requirement.

- *Dependency Encryptor–Decryptor*: The `Cipher` objects used for encryption and decryption must be specified and initiated in the exact same way except for the parameter specifying the operation in the `init(...)` statement. We assigned this category to all issues caused by conflicting configurations.

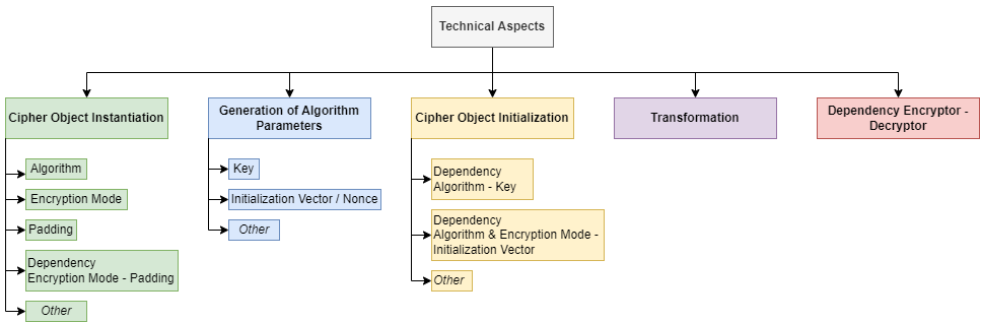

Figure 5: Hierarchy of technical aspects categories

If all tasks are correctly implemented, the code compiles and runs without any errors. Thus, if developers ask questions on Stack Overflow about a technical aspect, they either implemented a task incorrectly or have a question regarding one of these categories. During this first classification round, each

reviewer asked the following two questions in order to mark the issue: (1) *What implementation step was performed incorrectly that caused the error?* , (2) *What implementation step is targeted by the question?*

Requirements

Not every issue is a technical issue. For this reason, we defined a second set of categories concerning the design of an application. Consulting Sommerville as a theoretical base [137], we identified various types of functional and non-functional requirements as categories. During the analysis, we addressed this question: *Which requirements are the askers unable to meet?*. Not all requirements defined by Sommerville occurred in our analysis, and therefore we only assigned the following categories:

- **Use Case** (functional requirements)
- **Performance**
- **Space**
- **Reliability**
- **Portability**
- **Interoperability**
- **Security**

During the analysis phase, we only assigned a category to a post if either the original poster was not able to meet a certain requirement or someone warned that the shared code snippet might cause some issues regarding one of the requirements (*e.g.*, a security or performance hint).

Analysis of Security Risks

The goal of the second part of the analysis is to investigate *"the degree to which security risks exist in code snippets and also the body of Stack Overflow's posts."* For this purpose, we defined a set of security rules about the implementation of symmetric encryption. Then we manually checked the posts against the security rules to observe if there is any violation.

Table 8: The collected security rules

Rule ID	Rule - Cipher object instantiation
R-01	Use AES or Blowfish algorithm
R-02	Do not use ECB or CBC encryption mode
Rule ID	**Rule - Generating algorithm Parameters**
R-03	*Rules for key derivation*
R-03-a	Do not use a static key
R-03-b	Do not use static salt for key derivation
R-03-c	Use at least 64 bits of salt for key derivation
R-03-d	Use at least 1000 iterations for key derivation
R-03-e	Do not use a weak password
R-03-f	Do not reuse passwords multiple times
R-04	*Rules for IV / nonce generation*
R-04-a	Do not use a static IV
R-04-b	Do not use a static seed for IV generation
R-04-c	Use SecureRandom for IV generation
Rule ID	**Rule - Cipher object initialization**
R-05	Do not reuse the same key-IV pair
Rule ID	**Rule - Parameter transmission**
R-06	Do not use a static password to store

Security Rules

We derived our rules from two popular crypto static analysis tools, namely CRYLOGGER (Piccolboni *et al.* [122]) and CogniCrypt (Krüger *et al.* [94]). We only considered the rules that were applicable to symmetric encryption and organized them based on the technical aspect classification. We did not take into account the context of usage to simplify the evaluation. The resulting rules can be found in Table 8.

Tracking Security Rule Violations

We studied the original sample to observe any security rule violations. To that end, we only considered the question body, the accepted answer, and the comments. We also differentiated between "question" and "answer" as well as "code" and "text". Two reviewers analyzed the four aspects independently, compared their results, and eventually made a list for each:

- **Question–Code**
- **Question–Text**
- **Answer–Code**
- **Answer–Text**

While analyzing the code snippets, we concentrated on the parts in which encryption, decryption, key derivation, IV generation, and key storage were implemented. For instance, due to debugging purposes, if someone defined

a key as `String` in the main method and passed it to the encryption section as a parameter, we did not consider this a security risk. The encryption section can still be safe if an appropriately derived, non-static key is passed. Furthermore, some askers did not post their complete source code with their questions, which could have otherwise created some biases in our judgment.

4.2 Results and discussions

In the following, we present and discuss our results regarding the three facets of this chapter.

4.2.1 General crypto questions analysis

Our hyperparameter tuning demonstrated that the best number of topics is three. Similarly, after analyzing pyLDAvis's visualizations and top keywords for 1 to 25 topics, the two reviewers achieved a consensus on three as the number of topics. The pyLDAvis interactive visualization for the three topics is available online.[18] The reviewers named the topics by considering the general themes of top keywords returned by LDA (See Table 9). We determined that the first topic is about digital certificates and configuration issues, the second one is about programming issues concerning encryption and decryption, and the third concerns passwords/hashes and basic crypto-related algorithms (See Figure 6). As an influential indicator of topic relevancy, we realized that the frequencies of the candidate tags used in the three topics are aligned with the general themes of the topics.[19] For instance, we observed that the AES, DES, Encryption, and RSA tags are mostly used in programming issues, the Hash, SHA, SHA256, MD5, XOR, and Salt tags are more frequent in the password/hash topic, and finally, the Digital-signature, Keystore, OpenSSL, Private-key, Public-key, Smartcard, and X509certificate tags are more common in the digital certificate topic.

With respect to stratified sampling, we considered the number of documents in each stratum (*i.e.*, each topic) as 139, 124, and 119 documents from the first topic to the third one respectively. The selected documents were created in the last 5 years on Stack Overflow. Extracting the themes, the reviewers achieved 79% Kappa score, which demonstrates a substantial agreement between the two reviewers.

Topic One: Digital certificate and configuration problems

The manual analysis for the first topic depicts that developers discussed two main areas, namely certificate/OpenSSL (63%) and SSH (37%). For instance, the discussions were related to OpenSSL configuration, signing and verifying

[18] `http://crypto-explorer.com/paper_data/lda.html`
[19] `http://crypto-explorer.com/paper_data/tags-topics.csv`

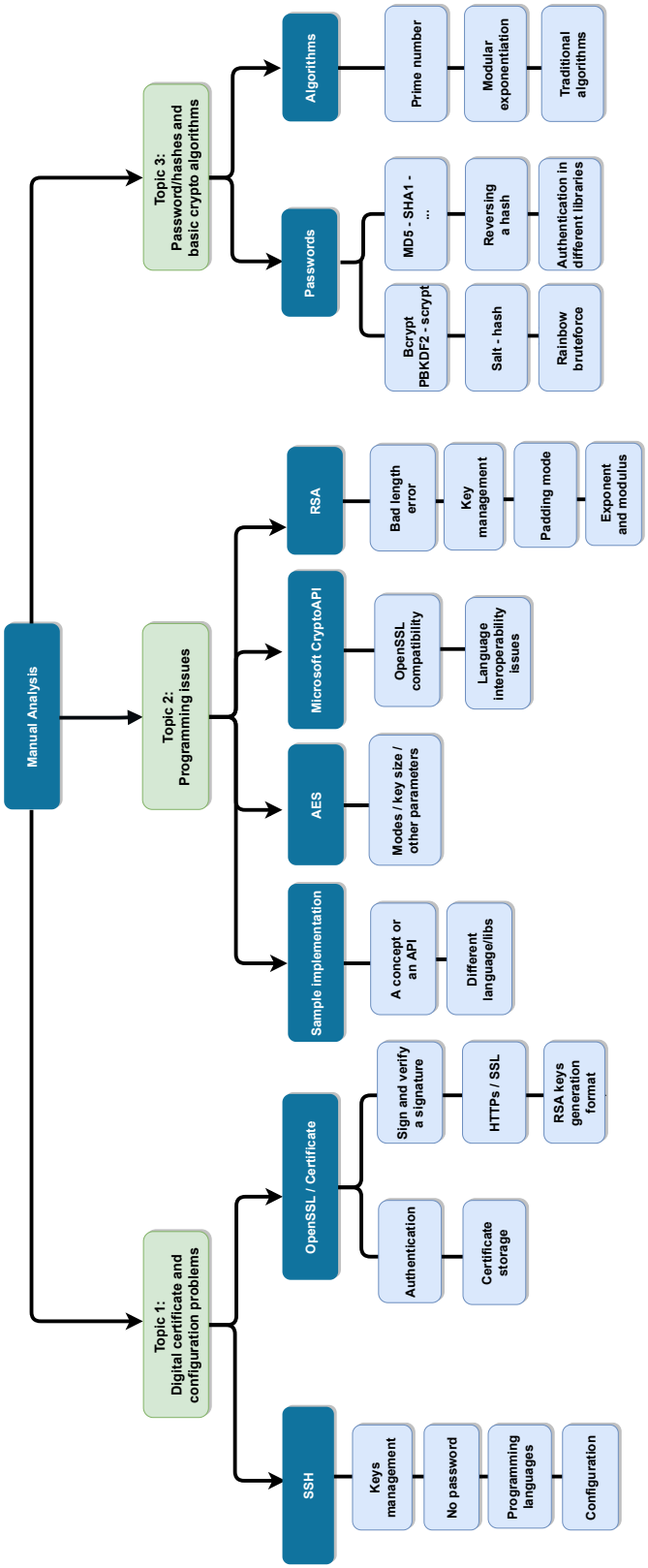

Figure 6: The results of manual analysis for the three topics

Table 9: The three topics and their top keywords

Topic	Top keywords
Digital certificate and configuration problems	use, certif, file, server, key, openssl, client, work, tri, need, sign, user, error,applic, creat, code, secur, app, encrypt, ssl, store, instal, like, connect, problem, want, way, run, request
Programming issues	key, encrypt, use, decrypt, code, data, file, string, tri, public, work, ae, im, byte, need, java, generat, messag, encod, privat, rsa, cipher, algorithm, block, like, implement, error, problem, function, text
Password/hashes and basic crypto algorithms	hash, use, valu, password, function,like, array, string, need, code, number, key, want, way, store,data, tabl, im, salt, tri, differ, time, algorithm, work, md5, user, make, generat, object, implement

a signature, and generating PEM files using OpenSSL. There were also questions concerning how to generate self-signed certificates, access a certificate store, create a Certificate Signing Request (CSR), establish https and secure connections, and configure certificate-based authentication in ASP.NET. In the SSH-related questions, the majority of the users had difficulty setting SSH with no password, checking the right permission for SSH keys, using SSH programmatically, and connecting to SSH servers of other platforms (*e.g.*, Amazon).

Topic Two: Programming issues

We observed that the three most frequently discussed programming languages were Java (*i.e.*, 44), C/C++ (*i.e.*, 31), and C# (*i.e.*, 19). In 31% of the posts developers discussed issues related to the AES algorithm such as different encryption modes (*e.g.*, CBC and ECB) and key sizes (*e.g.*, 128, 192, and 256-bit). In addition to symmetric encryption, 47% of the posts were related to working with asymmetric encryption (*i.e.*, RSA). The challenges were mostly concerned with different padding modes (*e.g.*, OAEP), how to calculate or understand the raw modulus and exponent numbers, and how to generate and work with different key file encodings in RSA (*e.g.*, DER-encoded format, PEM, or XML). Moreover, another evident problem was dealing with different RSA key formats, *i.e.*, Public Key Cryptography Standards (PKCS). The users commonly asked how to convert PKCS#8 to PKCS#1 or other standards, and how to programmatically generate or use different key standards in various crypto libraries (*e.g.*, Bouncy Castle). There were users who had problems with illegal block size errors, often misunderstanding the suitable usage of RSA, *e.g.*, encrypting a long text. Nevertheless, the discussions were resolved by proper responses that suggested incorporating AES and RSA into the encryption/decryption scenario. Another type of question was about the

issues in Microsoft CryptoAPI (12%). Developers reported issues on working with OpenSSL or using RSA keys from other sources, *e.g.*, importing keys from OpenSSL into Crypto API, converting RSA keys to be used by Bouncy Castle, verifying an OpenSSL DSA signature using CryptoAPI, having extra fields in generated keys by PHP OpenSSL, and signing a message with py-OpenSSL in Python and verifying it with CryptoAPI. Moreover, there were questions (10%) associated with how to either implement a scenario, *e.g.*, encryption of a string with RSA public key with Swift on iOS, or deal with problems while working with more than one crypto library or programming language, *e.g.*, encryption of a string with RSA in JavaScript and decryption in Java, or decryption of a string in Java which is already encrypted using AES-256 in iOS.

Topic Three: Password/hashes and basic crypto algorithms.

Our findings for the password/hash topic suggest that users primarily discussed problems associated with either passwords (86%) or basic crypto algorithms (14%). Different facets of producing secured passwords were the topic of most discussions. First and foremost, users were uncertain which hashing algorithms (*e.g.*, MD5, SHA-1) can provide a higher level of reliability and how password length contributes to the strength of the resulting hash. Users lacked the required knowledge as to what salt is and how salt can maximize the security of a hash. In addition to pointing out the pros and cons of static salt vs random salt, respondents encouraged users to use salted passwords in order to render the brute-force or the rainbow table attack prohibitively expensive. Developers were doubtful about which crypto functions, *i.e.*, bcrypt(), PBKDF2(), or Scrypt(), are more secure and faster, and what key differences distinguish the three functions from other hashing algorithms, *e.g.*, MD5, SHA-256. As regards the basic crypto algorithms, users contributed to responses concerning how to produce or find prime numbers, how to use the BigInteger class for RSA modular exponentiation, how to produce unique URL safe hash or IDs, and how to solve a Caesar Cipher or substitution ciphers. Lastly, a few users discussed how to program an authentication module in web programming frameworks such as Laravel, or CakePHP.

Topic difficulty and popularity

We checked the popularity and difficulty level of each topic so as to determine which questions attracted more attention or received acceptable answers with a longer time span, which the same approach was used in the previous study [153]. We used four factors to measure the popularity of a topic, namely the average number of views of documents, the average number of comments, the average number of favorites, and the average score of documents. The four fac-

tors can be found in the CSV files,[20] namely CommentCount, FavouriteCount, Score, and ViewCount. We considered the average number of ViewCount as the foremost factor to judge the popularity of a topic, the question's score and the number of FavouriteCount as the second most important factors, and the average number of comments as the last factor. To find the most difficult topic, we used two factors, namely the average time it takes for a document to obtain an accepted answer, and the ratio of the average number of answers in documents to the average number of the views. We avoided recently posted questions from affecting the analysis by only including those that are older than six months.

We infer that questions related to the usage of digital certificates, and configuration problems are the most popular (highest average ViewCount and FavouriteCount), and questions related to hashing and passwords are also viewed as popular based on the other two factors (*i.e.*, average CommentCount and Score). From the difficulty standpoint, we notice that the programming issues topic is the most difficult topic as it had a greater average response time, and its proportion of average answers to average views is the lowest.

Summary

The challenges in each theme were studied in detail to demonstrate how developers struggle to use or comprehend various areas of cryptography. According to our findings, we believe that there are two foremost reasons with which developers mainly encounter problems in cryptography. The first leading cause is a distinct lack of knowledge to discern *why* or *what* they need to use to accomplish a crypto task. We observed ample evidence where developers lacked the confidence to choose the best algorithm or parameter, for instance, the right and safest padding option in AES. Consequently, developers may use boilerplate code snippets from the provided answers, in spite of the answers' reliability and security[144]. In the second factor, although the fundamental concepts are the same, the implementation approach of a crypto concept in various crypto libraries is influential to developer performance. Compelling evidence in findings urges that working with more than one crypto library due to using various architectures or platforms in a project creates confusion for developers regarding *how* a particular problem can be resolved. They commonly have trouble in creating keys with one library and import them into another library or verifying a signature in a different crypto library. Furthermore, adequate explanations and the existence of useful examples in documentations can alleviate the difficulty of using cryptography.

[20] `http://crypto-explorer.com/paper_data/`

Table 10: The deduced themes, number of posts in each theme and associated description

Theme	# of posts	Description
Encryption/Decryption	112	Technical problems, *e.g.*, modes of encryption, AES, or IV
Library installation	111	Posts related to installation or compilation
Certificate-related issues	74	Posts related to SSL, self-signed certs, PEM, PKCS7, DER
Library interoperability	63	Posts related to working with more than a crypto library
Generate/store keys	45	Posts related to loading or generating a crypto key
Hashing	37	Posts related to MD5, SHA, and other hashing algorithms
Digital signature	34	Posts related to how to sign or verify a signature
Sample implementation	19	Posts where a sample code was requested
Random number generator	3	Posts related to generating a true random number
Cryptography attacks	2	Concerns for cryptographic attacks in discussions

4.2.2 Crypto libraries analysis

Table 10 lists the themes, the associated number of posts in each theme, and a brief summary of what each theme is. The highest number of posts is associated with encryption/decryption of a file while the least number of posts is associated with cryptography attacks. Table 11 describes in more detail the number of assigned posts to each theme in the 20 crypto libraries. The highlighted cell demonstrates the highest number of posts in each theme compared to other libraries. For instance, of 25 analyzed questions in pyOpenSSL, 17 posts were assigned to certificate-related issues. In the following, we discuss each of the 10 themes of developer challenges in the 20 crypto libraries.

Encryption/Decryption

In this theme, developers struggled with how to conduct file encryption or decryption. However, the range of sub-problems varies. The first group of challenges is with those who could encrypt a piece of data but the decryption phase was not successful. For instance, a developer encrypted a string with Spongy Castle and the decryption code was not working due to not employing AndroidKeyStore for retrieving the private key. Another observed issue was misusing the `doFinal, init` and `update` methods in the Cipher API. A developer missed all the important elements, *i.e.*, keys, IV, encoding, and padding, to perform the decryption process when working with the CryptoJS library.

Another type of discussion was related to the mode of encryptions. For instance, a developer asked for ways of checking the authenticity of an encrypted text and the responses suggested Galois/Counter Mode (GCM) in the AES encryption. In another discussion, a developer was unsure of the internals of cipher-block chaining (CBC) and why only the first block could be corrupted but the subsequent blocks will be as expected if the initialization vector (IV) is incorrect, whereas in other discussions, either the IV was forgotten in the decryption process or unequal IVs were used. A developer confused the difference between Electronic codebook (ECB) and CBC, and which one requires the IV for encryption/decryption of a file. With regards to different modes of encryptions, a common uncertainty was about the correct length of the IV.

One of the prevalent sub-problems was concerning the correct way of en-

coding/decoding the ciphertext. For instance, a developer forgot to use UTF-8 to convert plaintext to an array of bits. Other discussions had the same problem of converting the cipher to either hexadecimal or Base64.

There were also some challenges that could not be grouped together, and hence we classified them as miscellaneous. For instance, a developer did not know how to encrypt/decrypt a binary file. To do so, a parameter `use_binary=true` must be passed to the *DataSink_Stream* API in the Botan library. In another example, developers discussed how large files can be encrypted by the WebCrypto API. Developers struggled to use the provided functions in libraries to generate secured random numbers.

Other discussions were centered on password-based encryption (PBE). Developers asked how to configure PBE APIs in libraries such as Jasypt. The PBE API commonly requires a password, iteration count, and salt generator, for which developers struggled to assign the correct values.

Lastly, different padding schemes created technical problems for developers. Discussions were related to the security level provided between PKCS1.5 and OAEP, the usage of zero padding in OpenSSL, PKCS#7 padding with AES, and how padding can be disabled in a crypto library.

In the encryption/decryption theme, we found sub-problems in which developers mainly asked for help. The sub-problems include password-based encryption, paddings, encoding/decoding, modes of encryption, library specific issues, and decryption issues.

Library installation

This theme depicts problems regarding installation, compilation, usage issues, and setting up the prerequisites for a library to work. This theme has the second-highest number of posts as developer platforms and integrated development environments (IDEs) varied when developers worked with a specific crypto library. For instance, developers discussed the dependencies of Spongy Castle in the Gradle file, setting up Android Studio with Spongy Castle, adding JCE to JRE 8 on macOS Sierra, and building the Botan library with the nmake command. Each crypto library commonly uses a specific way to install or compile of the library or its modules. For instance, in the PyCrypto and M2Crypto libraries, developers commonly need to resolve their issues with the pip command, and similarly, the usage of npm was the key reason for other discussions related to the node-crypto library.

It is patently evident that the process of getting a crypto library up and running under different circumstances, e.g.*, platforms or IDEs, can be troublesome for developers.*

Table 11: The number of assigned posts to each theme in a crypto library

	Encryption Decryption	Library installation	Certificate-related issues	Library interoperability	Generate/store keys	Hashing	Digital signature	Sample implementation	Random number generator	Cryptographic attacks
OpenSSL	7	4	6	3			3	1		1
Bouncy Castle	5	1	8	1	7		2	1		
CryptoJS	7	1	1	13		2		1		
mcrypt	5	16		4						
PyCrypto	6	8	2	6			1	2		
phpseclib	4	7	2	4	2		6			
Crypto++	7	3		2	3	4	4	1		1
CryptoAPI	6		8	1	4	5	1			
pyOpenSSL		4	17		1		3			
Jasypt	9	11		1		4				
libsodium	8	8		3	2	1	1	1	1	
M2Crypto	2	12	3		3		3	2		
Web Crypto API	6	1	2	7	4	2	2	1		
JCA	6	4	4		4	3	3	1		
CommonCrypto	5	5		1		8	2	4		
node-crypto	9	1	1	7	3	3		1		
Botan	7	11	2	1	1	2			1	
Spongy Castle	4	5	6		8		1	1		
SJCL	8	1		8	2	3		2	1	
wolfSSL	1	8	12	1	1		2			

Certificate-related issues

We found two sub-problems with the theme of the certificate-related issue. The first challenge developers encountered was working with various file formats, *e.g.*, p7b, and various encodings *e.g.*, Privacy Enhanced Mail (PEM) or Distinguished Encoding Rules (DER). Developers asked about how to read or save PEM files using crypto libraries, storing/reading private keys in a Public Key Cryptography Standards (PKCS#8) file, differences between DER and PEM file formats, and storing/reading public and private keys in a PKCS#12 file. The other issue of developers was to extract various elements from a certificate, *e.g.*, expiration date, list of Subject Alternative Name (SAN) and Certificate Authority (CA), and cipher list. They also had challenges in checking a valid certificate, generating a self-signed certificate, using different versions of TLS and SSL, and TLS handshake issues.

Certificates are coupled with many cryptographic concepts and this fact complicates working with certificates. Various PKCS standards and the correct way of establishing a TLS connection are still problematic.

Library interoperability

It is common for developers to work with more than one crypto library in a large project. However, there might be some discrepancies between the libraries. A common issue was that developers encrypted a piece of data with OpenSSL, *i.e.*, via command line, and then they had issues with decryption of the ciphertext with another library. This is due to the fact that the default values in libraries commonly do not match. For instance, a developer encrypted a text with OpenSSL but could not decrypt it with the Botan library because of the default usage of PKCS#1 v1.5 padding in OpenSSL. Furthermore, on closer inspection, root causes are mainly the inappropriate encoding of the ciphertext, incorrect IVs, generating cryptographic keys differently, and using unequal key formats and padding options.

Working with more than one library seems to be a challenging task for developers due to different default values in APIs, encodings, paddings, and key generation methods.

Generating/storing crypto keys

For every cryptography scenario, developers need to generate and store their crypto keys. In the analyzed discussions, the challenges are related to storing keys, *e.g.*, AndroidKeyStore, generating a valid ECDSA or RSA key pair, generating a symmetric key, differences between trust store and keystore, generating keys with Key Based Key Derivation Function (KBKDF), the correct length of possible keys for various algorithms, and the meaning of modulus (n) and public key exponent (e) in RSA keys.

Developers has difficulties dealing with the differences of crypto keys among symmetric, e.g.*, AES, and asymmetric,* e.g.*, RSA or ECDSA, algorithms.*

Hashing

It appears that developers still talk about the possibility of reversing a hashed string. However, most of the discussions were about generating a hash string, the right way of using salt, calculating checksum for large files, issues in using Hash-based Message Authentication Code (HMAC), and the usage of hash functions, *i.e.*, Password-Based Key Derivation Function 2 (PBKDF2), bcrypt and scrypt.

Compared to the other themes, hashing requires developers to understand fewer concepts and hence, there are fewer discussions in the recent active posts.

Digital signature

Developers faced issues when signing and verifying a signature. A developer misunderstood the application of the Cipher API and the Signature API for signing a piece of data in JCA. Another developer was worried about performance bottleneck when there is a massive dataset. In other discussions, developers failed to verify a signature due to the wrong encoding of RSA keys in browsers (URL encoding), using a hash as data to be signed instead of the data itself, using the wrong key for signing or verification, the mismatched padding for the signature, and verifying a certificate in the chain of trust.

It seems developers suffered from the lack of technical knowledge about digital signatures and issues that are indirect to the topic, e.g.*, browser encoding and default values for padding in crypto libraries.*

Sample implementation

Developers mainly asked for two types of sample implementation. In the first type, developers had a sample code from a language or a specific crypto library and were looking for an equivalent piece of code in another language or library. For instance, a developer had a piece of encryption code in Objective-C but was not able to do the same in Swift. In the second type, developers had a goal but did not know how the task could be accomplished. For example, a developer requested a sample implementation of AES256 CBC in the M2Crypto library.

Documentation of crypto libraries should provide extensive secure examples so that developers have a reliable source of sample implementations.

Cryptographic attacks

Only 0.4% of the analyzed posts were concerned about cyber attacks. The first discussion was about conducting a man-in-the-middle attack when a self-signed certificate is used. The asker received comprehensive responses regarding why a self-signed certificate is not recommended. In the second discussion, a developer was not able to comprehend how the length extension attack works.

Just two discussions explicitly discussed attacks against cryptography. However, such discussions may appear more in crypto.stackexchange.com. At the same time, most general developers consult Stack Overflow as it is more general compared to crypto stackexchange

We attempted to cast some light on the common technical issues of developers with various crypto libraries. We observed that developer uncertainty in a particular crypto library not only is related to one or two areas but is frequently linked to more than five themes. There are some libraries, such as OpenSSL and WolfSSL, that are intended to be used for special purposes, *i.e.*, secure communications over computer networks. This increases the likelihood of identifying more questions related to the certificate issues in such libraries. Moreover, a popular crypto library, such as Bouncy Castle, presents a wide range of crypto APIs and be can be utilized in two popular programming languages, *i.e.*, C#, and Java. This can explain why identified questions are linked to seven themes. Some of the extracted themes are interrelated, *e.g.*, certificate-related issues, digital signatures, and generating/storing keys. For instance, a developer may need to generate an RSA key pair to work with certificates. However, we attempted to carefully identify the core issue of the posted challenge.

The detailed issues in working with various crypto libraries could provide valuable support for professionals to identify the probable pitfalls in the design phase of software development. Admittedly, identifying crypto pitfalls in earlier stages can substantially boost the security and the speed of development of software. As a result, such forethoughts can facilitate the use of cryptography in the implementation phase and prevent inexperienced developers from making fatal security mistakes that may have pernicious effects after the release phase. Further research is needed to shed light on how similar APIs in popular crypto libraries are misunderstood and whether the complexity of APIs has an impact on creating more problems or not.

4.2.3 Java symmetric APIs on Stack Overflow

In the following, we first discuss the results obtained from the analysis of JCA APIs relevant to symmetric encryption and then explain the state of security violations in analyzed posts on Stack Overflow.

JCA symmetric APIs issues

In the first analysis, depending on what the asker was struggling with, we classified the issues according to the *technical aspect* and/or *requirement* categories. In total, we recorded 219 issues, of which 197 (90%) were related to technical aspects while 76 (35%) were related to requirements, and 62 records were classified twice. We could not classify only one post due to the insufficiency of the provided information.

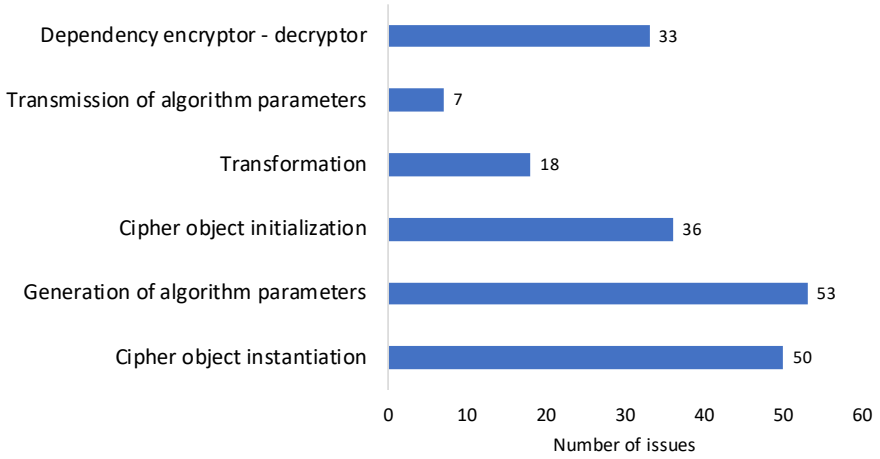

Figure 7: Number of issues assigned to the technical aspect categories

As shown in Figure 7, the most common categories in the first round of categorization were *Generation of Algorithm Parameters* (53) and *Cipher Object Instantiation* (50), both of which referred to specifying and generating the properties used during encryption and decryption. During the generation of algorithm parameters, developers might derive a key, an IV, and other algorithm parameters such as advanced authentication data. The askers especially struggled with key derivation (36 records). During the instantiation of a `Cipher` object, developers specify algorithm, encryption mode, and padding. In this context, the askers were particularly dealing with the latter two aspects, namely encryption mode (18), padding (11), and dependency of these properties (11). The third most common category was *Cipher Object Initialization* (36). Most issues were assigned to the *other* subcategory as the askers often did not pass all the required parameters of the `init(...)` method. The fourth most common category was *Dependency Encryptor–Decryptor* (33). The high prevalence of issues in this category implies that many developers lack knowledge about (symmetric) encryption in general. The fact that the `Cipher` objects for encryption and decryption must use the exact same algorithms and parameters is the basic principle of symmetric encryption.

Table 12: The number of issues assigned to the technical aspects subcategories

Subcategory	%	#	Main category (relative frequency)
Algorithm	14%	7	Cipher object instantiation - 22.83%
Encryption mode	36%	18	
Padding	22%	11	
Dependency encryption mode - padding	22%	11	
Cipher object instantiation - other	6%	3	
Key derivation	67.9%	36	Generation of algorithm parameters - 24.2%
IV / nonce generation	26.4%	14	
Generation of other algorithm parameters	5.6%	3	
Dependency algorithm - key	22.22%	8	Cipher object instantiation - 16.44%
Dependency algo/encryption mode - IV	22.22%	8	
Cipher object initialization - other	55.56%	20	
Transformation	100%	18	Transformation - 8.22%
Key transmission	71.4%	5	Transmission of algorithm parameters - 3.2%
Transmission of other algorithm params	28.57%	2	
Dependency encryptor - decryptor	100%	33	Dependency encryptor - decryptor - 15%

Some askers confused the two methods that perform transformation, *i.e.*, `update(...)` and `doFinal(...)`. They did not know which method must be called in their scenario. Among issues referring to the transmission of algorithm parameters (7), the majority (5) were about the transmission of the key. The remaining issues were related to the IV (1) and the salt used for password-based key derivation (1). Table 12 presents the complete list of subcategories for technical aspects and the associated frequency of each issue.

Security was the predominant category with respect to the requirements categories (46 records), which seemed logical as the posts were all centered around cryptography. However, only three askers were concerned about the security implication of their code snippets. They were worried about the safety of generated IV, or about whether it is secure to reuse algorithm parameters (e.g., key or IV) for several transformations, or about whether encryption becomes safer if salt is added to the plain text. A more common security issue was that programmers were not able to run AES-256 due to missing security policy files (5 posts).

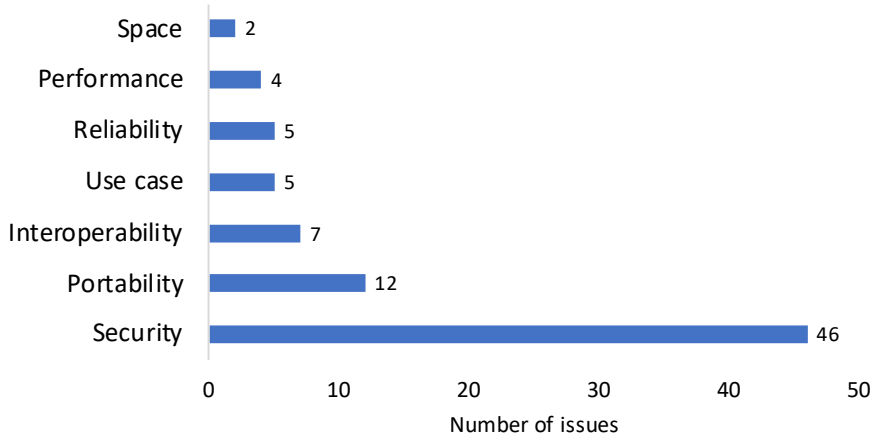

Figure 8: Number of issues assigned to the requirement categories

Figure 8 presents the number of issues assigned to the requirement categories. There were 12 issues related to the portability of an application, often referring to original askers who did not specify all values. For example, several programmers only passed "AES" to the `Cipher.getInstance(...)` method. In that case, default values were automatically used for encryption mode and padding. These values, however, varied among different providers as well as different platforms. Seven records were assigned to the interoperability category that often occurred if the askers implemented one task in a library in which the default values (*e.g.*, padding) were different from the standard crypto APIs in Java. As a result, the askers incorrectly instantiated or initialized the Java `Cipher` objects. For example, some of these default values were also not supported by JCA (*e.g.*, ZeroPadding) or a developer used a non-standardized function (*i.e.*, SHA1PRNG) for random number generation. The issues referring to the use-case category were mostly about the misuse of encryption for an inappropriate use case (2) and the use case that was not supported by the JCA APIs (2). Most of the issues assigned to the reliability category were due to declaring the `Cipher` objects statically in global space. Moreover, the applications frequently crashed because the `Cipher` objects were not thread-safe.
Some developers complained that the execution of the `getInstance(...)` method was taking too long (lacking performance). One developer also observed that the encryption of a large file was time-consuming. Yet, others reported that an `OutOfMemoryException` occurred when they attempted to encrypt a large file all at once.

As previously mentioned, some records were classified twice in both technical and requirements aspects. Figure 9 indicates the relative overlapping of categories. We observed that the highest overlap is for *Generation of Algorithm Parameters* and *Security*, accounting for 21% of the records. Most of them referred to static values being used for key or initialization vectors. The second-highest overlap is between the *Cipher Object Instantiation* and *Security* categories (16%). Almost all of the issues referred to the use of an unsafe encryption mode. The issues assigned to the *Portability* or *Interoperability* category and some other technical aspects category were mostly due to default values that varied among platforms and libraries.

Security violation

We manually checked 150 questions, 84 answer posts, and related comments to detect any violations against the specified ruleset and found a total of 331 security risks. Most of the violations (249, 75%) stemmed from code snippets in question posts, the body of which only included 38 security risks. We also found 35 rule violations in answers' code snippets and nine in answers' text. Some answers just fixed the functionality of a question-related code snippet

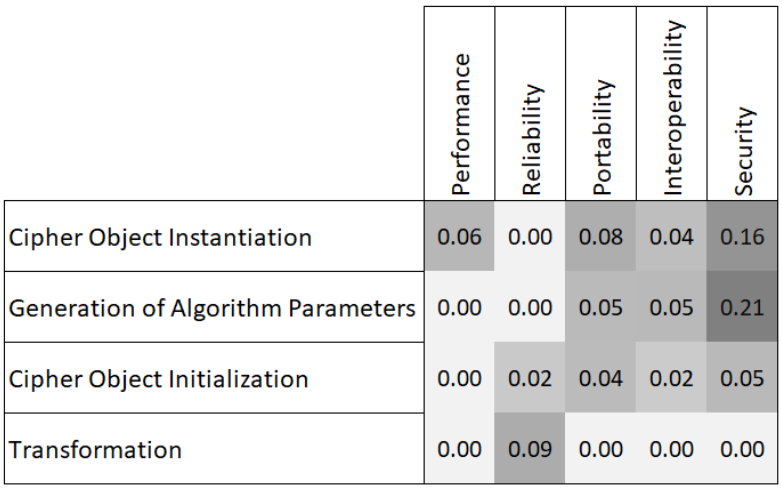

Figure 9: The relative overlapping of technical aspects and requirements categories

Table 13: The number of found security violations based on the collected rules

Rule		Q_code	Q_text	A_code	A_text	Total
R-01	Use AES or blowfish algorithm	24	11	0	0	35
R-02	Do not use ECB or CBC encryption mode	113	20	20	6	159
R-03-a	Do not use a static key	43	2	3	2	50
R-03-b	Do not use static salt for key derivation	7	1	1	0	9
R-03-c	Do not use short salt for key derivation	3	0	1	0	4
R-03-d	Do not use <1000 iterations for key derivation	6	0	1	0	7
R-03-e	Do not use a weak password	10	0	1	0	11
R-03-f	Do not reuse passwords multiple times	3	0	1	0	4
R-04-a	Do not use a static IV	24	2	5	1	32
R-04-b	Do not use a static seed for IV generation	1	0	0	0	1
R-04-c	Use SecureRandom for IV generation	0	0	0	0	0
R-05	Do not reuse the same key-IV pair	12	2	2	0	16
R-06	Do not use a static password for store	3	0	0	0	3

without enhancing its security. The resulting code, therefore, carried the security risks from the question. Another common observation was that users correctly provided excellent advice that ECB should not be regarded as a safe mode of encryption. However, they suggested using CBC instead, which must not be employed in client-server scenarios. Such advice is not safe, especially if the context is unknown in the question's body.

The further analysis focused on the code snippets. However, some askers only shared the code section where the issue occurred and did not show how the other relevant parts (*e.g.*, key derivation) were implemented. On average, each question contained 1.66 security risks in its code snippets. We noted that the average for the most popular posts (1.91) is slightly higher than that the newest posts (1.43). In total, 24 question posts did not contain any security risks in their code snippets. The most frequently violated rule was *R-02: Do not use ECB or CBC encryption mode*. In more than 75% of question posts, the asker used one of these unsafe block cipher modes. This is also due to the fact that ECB is the default encryption mode for most crypto providers.

The second and third most violated rules were *R-03-a: Do not use a static (= constant) key*, *R-04-a: Do not use a static (= constant) IV*, and *R-01: Use AES or Blowfish algorithm*. It is worth mentioning that we included 24 posts where DES or 3DES was used. Some askers stated that they used static values only for Stack Overflow to simplify their code. Nevertheless, this is a potential security risk if a developer naively copies and pastes the code snippet for personal usages. Using both static key and IV could lead to the reuse of key-IV pairs (*R-05*), which was the fifth most often violated rule. The least violated rules were *R-04-c: Use SecureRandom for IV generation*, *R-04-b: Do not use a static seed for IV generation*, and *R-06: Do not use a static (=constant) password for store*. Since most askers used ECB, which does not require an IV, and the rest used static IV, there were not many code sections showing IV generation. There were a few questions' bodies that showed or explained how the key was stored.

4.3 Threats to validity

In the following, we describe the threats to validity of each of the explored facets.

4.3.1 General crypto questions analysis

In this study, we concentrate on one major platform where developers discuss crypto topics. This may not be sufficient as there are many other platforms, such as crypto Stack Exchange, which can provide more data to analyze. We measured topic difficulty and popularity based on metrics used in the previous study. Nevertheless, these observations may not be sufficient to determine what type of crypto questions are more challenging than others. Users may not always feel responsible for selecting a reasonable answer as an acceptable answer. Therefore, not having an accepted answer does not necessarily determine if the question is challenging for others.

4.3.2 Crypto libraries analysis

We selected 25 posts from each crypto library. This may not be a representative sample of the whole population; however, we were particularly interested in the common themes of issues in various libraries, not just one library. We selected the latest posts that are active on Stack Overflow that had at least one answer and skipped the questions to which nobody responded as well as the questions with no positive received votes. Nonetheless, there are various approaches to choose the posts, *e.g.*, the number of answers or the number of views, while each of them can impose some threats to validity. To reduce subjectivity, two reviewers carefully performed thematic analysis to extract the themes. The final list of themes is deduced based on their discussions and

cross-checking. Nevertheless, a few posts could have been assigned to other themes or a current theme could have been divided into several sub-themes. We may not have covered all the crypto libraries discussed on Stack Overflow, but we indeed selected the popular ones.

4.3.3 Java symmetric APIs on Stack Overflow

In relation to the methodical approach, a major threat to validity is that we did not verify the intercoder reliability of our issue classification or our security check according to existing metrics. However, the two reviewers consulted their results after independently conducting their analysis. Nonetheless, our data is published on GitHub.[21] Another limitation is that the total number of Stack Overflow posts matching our scope cannot be precisely elicited. There is no guarantee that the employed queries returned all posts referring to our scope. Additionally, we had to exclude almost two-thirds of all threads as they did not fit into our scope. The sample size may not be the perfect representative of problems in the scope of interest. Many of the identified violations could be due to the lack of our knowledge of the askers' intention. Some users may not want to share their keys and IVs and hence, they use some static strings deliberately to render the question easier for others to understand and answer.

4.4 Summary and conclusion

We conducted a large-scale study on crypto issues discussed on Stack Overflow to find out what crypto challenges users commonly face in various areas of cryptography. Our findings suggest that developers still have a distinct lack of knowledge of fundamental concepts, such as OpenSSL, asymmetric and password hashing, and the complexity of crypto libraries weakened developer performance to correctly realize a crypto scenario. We call for dedicated studies to investigate the usability of crypto APIs.

In particular, we were curious to observe what technical problems are common among different crypto libraries. We selected 25 discussions from 20 crypto libraries on Stack Overflow and to the best of our knowledge, we did not find any study in which 20 crypto libraries were considered. We identified 10 themes in the discussions and the majority of libraries were involved in more than five themes. There exist 0.04% of questions concerning attacks against cryptography, whereas 112 questions were related to encryption/decryption issues. The developers also asked questions mostly about library installation, digital certificates, crypto keys, and library interoperability. The implications of these findings can assist security and software professionals to correctly guide their team members when dealing with cryptography, and

[21] data directory

especially crypto libraries. Further work is certainly required to disentangle the problematic commonalities among various crypto libraries.

We addressed two goals defined earlier in the third facet, namely (1) *issues faced by developers when accomplishing symmetric encryption scenarios*, (2) *the degree to which security risks exist in code snippets, and also the body of Stack Overflow's posts*. Considering tasks with which developers are struggling, most of such tasks involved generating algorithm parameters, especially deriving the key from a password. The second most problematic task was to instantiate a `Cipher` object. The askers specifically failed to correctly specify encryption mode and padding. The third most problematic task was initializing the `Cipher` object where the developer did not pass all required parameters to the `init(...)` method. One of the major issues in this context was that default behavior and values were dependent on the crypto provider. The platform, however, decides which providers are available and which provider is chosen by default. This reduces the portability of applications. An additional issue was the high number of overloaded methods, particularly `getInstance(...)` and `init(...)`. Moreover, some askers failed to choose the right usage of two methods to perform transformations, `update(...)` and `doFinal(...)`.
For the second part of analysis, we found that security risks were particularly present in code snippets from question posts. Several answers provided advice regarding or even improved security, whereas others only focused on the functionality of the code. The most common security risk was the use of an unsafe encryption mode. This is related to the most common providers that use ECB as the default block cipher mode of operation. However, some answers suggested using CBC instead of ECB, which is not safe depending on the context. Other common security risks were the use of static values for either the key or IV, or both. The procedures used for password-based key derivation also contained security risks. JCA supports PBKDF2 as a safe procedure for password-based key derivation. However, it was rarely used.

Chapter 5

Crypto hurdles - the developer perspective

In the previous chapter, we investigated the prevalent problems of cryptography discussed on online Q&A forums. We found that there are multiple areas, *e.g.*, OpenSSL, hashing algorithms, or padding options, that expose developers to some new concepts to comprehend. This can be due to the fact that developers are not adequately educated about cryptography. In this chapter, we focus on the other pillar, *i.e.*, developer perspective, to shed light on what developer practices are common in this domain.

Developers evidently do not hold the same level of knowledge in cryptography. Accordingly, their practices and concerns about the safety of their code are varied. For instance, Nadi *et al.* conducted two surveys with 48 developers in which the participants indeed faced difficulties in using Java crypto APIs with varying levels of knowledge in cryptography [112]. Their findings showed that the participants spent at least several hours reading online resources completing their crypto tasks and that 65% of them found the APIs hard to use. Similarly, Robillard *et al.* surveyed Microsoft developers and found out that poor documentation is the major obstacle to learning APIs [130].

Developers with varying levels of expertise and experience differ in both opinions and performance. Acar *et al.* had an experiment with 307 active GitHub users to determine how such developers complete several security-related coding tasks [3]. Interestingly, they noticed no significant differences between the analyzed factors including security perception, the participants' self-reported status as a student or a professional developer, years of experience, and security background. Likewise, Oliveira *et al.* devised a study for 109 developers to use and perceive some APIs with embeded blind spots [115]. The results confirmed that developer expertise and experience did not predict their ability to distinguish the blind spots. However, we believe that developer performance and practices can be further studied to reveal more influential aspects of cryptography.

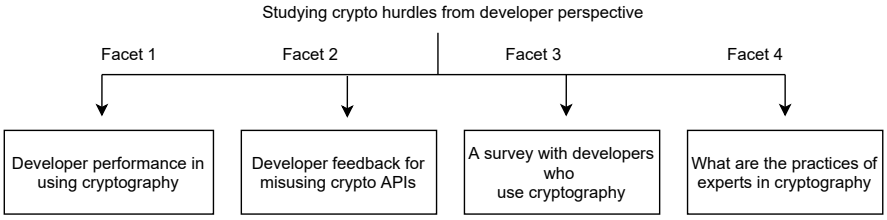

Figure 10: The methodology followed to answer the third research question

5.1 Study design

The aim of this chapter is to investigate developer performance and practice in using cryptography. To that end, we address the third research question (RQ3) of our study: *What are the practices of developers in using cryptography?*, from four facets, illustrated in Figure 10.

Facet 1 → We analyzed 2 324 open-source Java projects from GitHub that rely on Java Cryptography Architecture (JCA) to understand how crypto APIs were used in practice, and what factors affected the performance of developers in using these APIs. We found that, in general, the experience of developers in using JCA did not correlate with their performance. The number or frequency of committed lines of code, for instance, showed no correlation with developer performance.

Facet 2 → We collected 489 projects on GitHub in, and then contacted the maintainers of each project, in chapter 4, by creating an issue on their repository to discuss their project's misuses. We then classified developer feedback into eight themes, some of which included: security caveats in the documentation of crypto APIs were scarce; developers may overlook misuses that originate in third-party code; some repositories were not maintained; developers were uncertain concerning about the correct use of an API; and developers complained that the context in which a crypto API was used was not security-related.

Facet 3 → We surveyed 97 developers who had used cryptography in open-source projects, in the hope of identifying developer security and cryptography practices. We asked them about individual and company-level practices, and divided respondents into three groups (*i.e.*, high, medium, and low) based on their level of knowledge. We found differences between the high-profile developers and the other two groups. For instance, high-profile developers enjoyed more years of experience in programming and more background in security, had attended more security and cryptography courses, showed grave concerns over security, and tended to use security tools more than the other two groups.

Facet 4 → We investigated whether users who were active in cryptography discussions also used cryptography in practice. We collected the top 1% of responders who had participated in crypto discussions on Stack Overflow, and manually analyzed their crypto contributions to open source projects on GitHub. We identified 319 GitHub profiles that belonged to such crypto responders and found that 189 of them used cryptography in their projects. Further investigation revealed that the majority of analyzed users (i.e., 85%) used the same programming languages for crypto activity on Stack Overflow and crypto contributions on GitHub. Moreover, 90% of the analyzed users employed the very cryptography concepts on GitHub about which they themselves advised on Stack Overflow.

In the following, we explain how we conduct the data collection and analysis phases for each of the aforementioned facets.

5.1.1 Developer performance

Data extraction

We mined Java projects on GitHub, downloaded, and compiled those that use JCA. We then ran the CogniCrypt tool on these projects to identify crypto issues. The experiment entailed the following steps: we initially started with 183 projects using JCA APIs that were identified in previous work [93]. We were interested in collecting as many JCA projects as possible that we could associate to each developer. Therefore, instead of an exhaustive search on GitHub, we checked what other projects each developer had contributed to, and whether they use JCA. We continued this process for every new project, and stopped when we obtained 2 324 projects that use JCA APIs. Based on the import statements in each project, we used the GitHub API search to check whether a project uses any of the crypto classes specified in the CogniCrypt rule set. In case a project was forked, we looked for the original repository to download.

Building projects

We needed to compile each project in order to run a static analysis that identifies crypto misuses. We wrote a bash script to build all the downloaded projects. The bash script first checked the existence of the POM file in the project's path, and if it existed, we would continue with compilation, otherwise we excluded the project. We tried to compile projects using the Maven build tool, while skipping the running of the tests (to save time). Many projects could not be compiled due to dependencies that were not resolved. We excluded such projects and did not manually investigate the issue. Ultimately, we were able to compile and build 2 324 projects.

Analyzing projects

We used a static analysis tool called CogniCrypt to detect known misuses of cryptographic APIs in Java bytecode [93]. We chose CogniCrypt for two main reasons: first, we had contact with the main developers of this tool, who were willing to support us in the execution of the experiment. Second, CogniCrypt covers a wide range of crypto APIs while keeping false positives at a manageably low rate (usually below 10%).

In this step, a bash script went through the folders of all projects and recursively looked for the *target/classes* folder where the *.class* files reside, to feed them into CogniCrypt.[1] We specified a timeout of 15 minutes to abort lengthy analyses. In the end, roughly 15% of the analyses were interrupted, and we succeeded in analyzing 2,141 projects.

5.1.2 Developer feedback

As described in chapter 3, we collected 489 Java projects in which JCA APIs were used and investigated how developers used such APIs. The findings highlighted that JCA API misuse is prevalent among the projects. To understand the reasons behind the API misuses, we contacted 216 maintainers of repositories on GitHub, which represents a sample size with a 95% confidence level and 5% margin of error. For each repository, we opened an issue on the GitHub page, explained our objectives for reporting crypto misuses, provided an explanation for each misuse, pinpointed the affected Java files, associated line numbers, and API names. We waited 20 days for responses from the developers of repositories, and then we manually extracted their responses. Thereafter, two reviewers reviewed each response to determine the key message of each response. Finally, they cross-checked their findings and, in case of a conflict, they revisited the concerned response.

5.1.3 Developer survey

We conducted an online survey with developers identified in a recent work [72] as having used Java Crypto APIs in real-world applications, to understand what security and cryptography practices such developers report. In the following subsections, we explain our methodology and present the survey results. We adopted an anonymous online survey approach which involves the following steps: (i) selecting developers, (ii) designing the survey, (iii) testing and publishing the survey, and (iv) analyzing the survey. The questions and the responses of the survey are available online.[2]

[1] We used the publicly available command-line version of CogniCrypt and its rule set available at the time of running this analysis on Jan 16th, 2019.

[2] `http://crypto-explorer.com/crypto/`

Table 14: Factors to explore in the survey

Demographics	Developer-level	Company-level
Developer age	Security course attendance and experience as code auditor	Security training by company
Years of experience in programming	Crypto knowledge level and experience in using Crypto API	Existence of security consultant
Years of experience in Java	Background in IT security and security concern level	Company security concern level
Educational level	Ways of solving crypto problems and evaluating crypto code	The percentage of security developer in company

Objective

Previous studies have shown that developers have difficulty in using cryptography securely [70][112]. However, we want to tackle the reasons why developer performance varies in using cryptography. The objectives of this research are as follows: (1) except for technical difficulties of crypto APIs explored in previous research [43] [72] [110], the findings can shed some light on the worrisome and promising practices among developers with respect to cryptography, (2) finding the indicative factors can assist professionals to correctly guide and lead such developers at workplaces.

Selecting developers

We selected developers from a recent study conducted by Hazhirpasand *et al.* [72], where the authors investigated Java Cryptography Architecture (JCA) uses and misuses in 489 open-source projects. We identified developers who committed code containing crypto uses to these repositories, *i.e.*, they extracted their names and email addresses using the git blame command.

Survey Design

In the survey, we collected information about the participants in three sections, and then evaluated the factors to determine which of them influence developer knowledge. To determine the explored factors, we studied the literature and identified studies wherein individual or work-related facets were studied with regard to cryptography (See Table 14). Thereafter, we constructed a list of explored factors that could influence developer knowledge, namely security tool adoption [86] [13], security concern [151] [140], means of resolving crypto challenges [2] [123], security training and its frequency [143] [111] [57], and work and technical experience [129] [3] [115] [2]. Nevertheless, the aim of the previous studies was to evaluate developer performance or developer practice but not developer knowledge.

The initial section is dedicated to the demographic information of developers. Within this section, we asked for their degree, field, age, years of programming and Java programming experience. Participant demographics help us to determine which factors may affect a respondent's answers.

In the second section, we mostly focus on developer practices, *e.g.*, security or cryptography course attendance. They are asked to specify their level of knowledge about cryptography as well as their experience with crypto APIs. We used a 5-point Likert scale to ask developers about their security concerns in development. Further, we asked developers what information sources they use to solve a crypto scenario or how they evaluate a crypto code snippet.

In the last section of the survey, we primarily concentrate on company-level factors. We provided them with questions regarding the existence of security consultant, company-level security concern (5-point Likert scale), and the percentage of developers responsible for secure development.

Testing, and Publishing the Survey Tool

We used Google Forms to create our online questionnaire. As overlong questionnaires are rarely completed on the internet [18], we limited the completion time of the survey to less than 5 minutes. To evaluate the survey before asking the real participants, we asked five colleagues to review the survey to reveal potential misunderstandings. Then, based on the received recommendations, we refined the questions and rearranged them. Next, we emailed the 1231 developers. We noticed that 128 email addresses were not valid, which left us with 1 103 potential survey participants.

Survey Analysis

We received 97 responses (8.7%) within a month. To perform the analysis, we did not consider missing values in the analysis. We used percentage graphs in order to analyze responses of Likert scale questions. Notably, the explanations of respondents to the open-ended question consisted of fewer than 20 words. However, to minimize human errors, two reviewers coded the responses and cross-checked the consistency of the results.

Knowledge factor

Nadi *et al.* conducted two surveys with 48 developers and devised a four-level classification for developer crypto knowledge [112]. We used the same levels in the survey. However, we attempted to minimize the impact of wrong assumptions with regard to how developers report their level of knowledge in cryptography. As a result, we provided the participants with an explanation

of what each level means in this study. The four-level items can be viewed in the survey file.[3]

Ethics

The developers' email addresses were identified by the use of the git blame command in a recent study [72]. We also asked developers to read the statement of the survey and state their agreement before participating. Moreover, we did not collect any personal information except for the information explicitly gathered by the survey instrument.

5.1.4 Experts' practices

We describe how we choose crypto tags on Stack Overflow, and our approach to fetch the top 1% of crypto responders, extract their GitHub profiles, and identify their crypto contribution (See Figure 11).

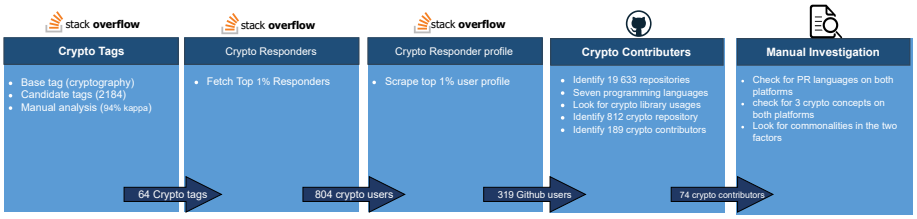

Figure 11: The pipeline for collecting and analyzing top crypto responders

Crypto Tags

To find top crypto responders on Stack Overflow, we had to identify crypto-related tags. We started with the "cryptography" tag, *i.e.*, the *base tag*, to find other tags that were used together with the base tag. To access the data, we used the Data Explorer platform (Stack Exchange).[4] We found 11,130 posts that contained the base tag. Together with the base tag, there were 2,184 other tags, *i.e.*, *candidate tags*. However, not all the candidate tags were related to cryptography. The list of candidate tags is available online.[5]

To discern crypto-related tags, two reviewers separately examined all the tags and marked the crypto-related ones. We then calculated Cohen's kappa, a commonly used measure of inter-rater agreement [39], between the two reviewers, and achieved 94% Cohen's Kappa score between the two reviewers, which indicates almost perfect agreement. Next, we compared their list of

[3] `http://crypto-explorer.com/crypto/`
[4] `https://data.stackexchange.com/stackoverflow/query/new`
[5] `http://crypto-explorer.com/mapping_data/`

Table 15: The 64 crypto tags and associated unique top 1% crypto responders

Responders	Tag	Responders	Tag
202	encryption	2	encryption-asymmetric
176	hash	2	cryptoapi
98	cryptography	2	pbkdf2
76	openssl	2	jca
29	md5	2	jasypt
20	keystore	2	commoncrypto
16	xor	2	libsodium
14	digital-sig	2	phpseclib
13	sha1	1	ellipticurve
12	x509certificate	1	ecdsa
11	rsa	1	diffie-hellman
10	mcrypt	1	rsacryptoserviceprovider
8	sha256	1	bcrypt
8	private-key	1	node-crypto
8	sha	1	sjcl
8	public-key	1	spongycastle
7	bouncycastle	1	cryptoswift
7	smartcard	1	hashlib
6	public-key-encryption	1	wolfssl
5	x509	0	crypto++
5	salt	0	pkcs11
5	hmac	0	jce
5	pycrypto	0	pkcs7
4	cryptojs	0	cng
4	pyopenssl	0	cryptographic-hash-function
3	aes	0	aescryptoserviceprovider
3	encryption-symmetric	0	rijndaelmanaged
3	rijndael	0	webcrypto-api
3	3des	0	mscapi
3	m2crypto	0	charm-crypto
3	botan	0	javax.crypto
2	des	0	nacl-cryptography

crypto tags and discussed the inconsistencies. Finally, we came up with a list of 64 crypto-related tags.

Crypto Responders

We executed a query on the Data Explorer platform to fetch the top 1% of crypto responders for each of the identified tags from Stack Overflow. Table 15 presents the 64 tags and associated top 1% of *unique* crypto responders. We excluded the crypto responders that we had already found in other tags. For instance, the crypto++ tag had four top crypto responders, considering that they were among other tags. In total, we retrieved 804 top crypto responders. The list of top crypto responders is available online.[6]

Crypto Responder Profile

Stack Overflow offers the ability to its users to share their social media addresses (*e.g.*, Twitter, GitHub, and personal websites) on their profile. Nev-

[6] `http://crypto-explorer.com/mapping_data/`

ertheless, the aforementioned information is not accessible on Stack Exchange Data Explorer. Hence, to find the selected users' GitHub profiles, we automatically scraped profiles of the 804 Stack Overflow top crypto responders. Using the BeautifulSoup library in Python, we parsed each user profile automatically. For 804 Stack Overflow users, we could identify 319 GitHub profiles.

Crypto Contributors

We used the GitHub repository API and collected a total of 19 633 public repositories associated with the 319 GitHub users. We selected the top seven programming languages used in the repositories, *i.e.*, Python, Ruby, C, C++, Rust, Java, and C#.

To understand which crypto libraries are popular in the selected languages, we consulted with two crypto experts. Among the suggested names, there are some candidates that come with the languages, such as *Java.security* in Java, or the libraries that are widely accepted and well-known, such as *Bouncy Castle* for Java and C#. Afterward, to ensure the rest of the suggested libraries are largely accepted in developer community, we checked how popular (*i.e.*, star and fork) the suggested open-source crypto libraries are on GitHub, *e.g.*, *libsodium* for the C language had 9.2k stars and 1.4k forks. The crypto libraries had on average 1844 stars and 346 forks, and the median numbers were 1105 and 245, respectively.

Using the compiled list of crypto libraries in Table 16, we employed the GitHub Code Search API and a custom regex script to identify in which files crypto namespaces, *e.g.*, "System.Security", were used. At the time of writing this study, the GitHub Code Search API could not perform the exact keyword search for the crypto namespaces. Therefore, we relied on a supplementary regex script to ensure the identified code snippets contain the namespaces. We retrieved a total of 2 404 crypto files in 812 repositories.

In the last step, we used *git blame* to identify the contributors who had committed to the 2 404 crypto files. To do so, we cloned the 812 crypto repositories and extracted authors and committers of crypto files by git blame. We then fetched the developers' email addresses, usernames, and full names by GitHub user API in order to check whether they are among the contributors of the 812 crypto repositories. Of the 319 top crypto responders on Stack Overflow, we found that 189 developers had crypto contributions on GitHub. They had a mean of 14 and median of three.

Manual Investigation

To address the research question, we performed a manual analysis to observe to what extent users employ cryptography in practice. To this end, we checked two aspects of their contribution, *(1) the programming language*

Table 16: The selected crypto libraries in the seven programming languages

Python	Ruby	Java	C	C++	C#	Rust
passlib	bcrypt-ruby	Java.security	libgcrypt	Botan	Bouncy Castle	octavo
pynacl	Ruby Themis	Javax.crypto	NaCl	Cryptlib	libsodium-net	rustls
hashlib	digest	Bouncy Castle	crypto-algorithms	Cryptopp	security.cryptography	rust-crypto
pythemis	RbNaCl		Themis	HElib	PCLCrypto	sodiumoxide
PyElliptic			wolfSSL			crypto
bcrypt			libsodium			Ring
			S2N-tls			

used for crypto purposes on both platforms, (2) crypto concepts used on both platforms.

Identifying detailed crypto concepts in various crypto libraries as well as crypto discussions can be an arduous task. Therefore, we deduced the concepts used in this study from recent work on the categorization of developers' crypto challenges on Stack Overflow [77]. The researchers' findings revealed that de-

velopers mostly encounter challenges concerning hashing, symmetric/asymmetric, and digital signature. Accordingly, we assumed that developers commonly use three high-level crypto concepts, which are *(1) hashing, (2) symmetric/asymmetric, and (3) signing/verification.*

In our manual analysis, we attempted to find commonalities in the programming languages (*i.e.*, the seven languages) and crypto concepts that are used by a developer on both platforms.

To compute the sample size for studying 189 users on GitHub, we defined a confidence level of 95% and 9% as the margin of error, which yields 74 for our sample size. We then randomly selected 74 users from the population. Writing queries on the Stack Exchange Data Explorer platform, we automatically retrieved all the posts (*i.e.*, titles, question and answer body) wherein the 74 developers were involved on Stack Overflow.

Two reviewers manually reviewed all the posts to extract the programming languages used in the discussions, *i.e.*, question and answer body. Afterward, they also checked the title and question body to understand to which concept or concepts a particular discussion can be assigned. They checked the crypto codes of the 74 users on GitHub, and extracted the crypto concept(s), and recorded the programming language of the crypto files. To understanding the crypto concepts, they looked for the APIs used in the crypto files. For instance, if the MessageDigest API was used in a Java crypto file, they assumed that the developer encountered the hashing topic in practice. In cases where they had doubts about the APIs, they referred to the API documentation of the library. They had several sessions in order to compare the results of their investigations and build a unified list. Ultimately, they checked for commonalities of the languages and the crypto concepts that the users used across the two platforms.

5.2 Results and discussions

In the following, we present and discuss our results regarding the three facets of this chapter.

5.2.1 Developer performance

We first present the state of cryptography uses in open-source Java projects, and then present our study of the factors that may influence developers' performance.

The State of Cryptography Uses

We analyzed a total of 2,324 projects, *i.e.*, 2,141 plus the 183 initial projects. These projects consisted of a total of 2,652 unique files containing JCA APIs, which means, on average, 1.4 files per project.

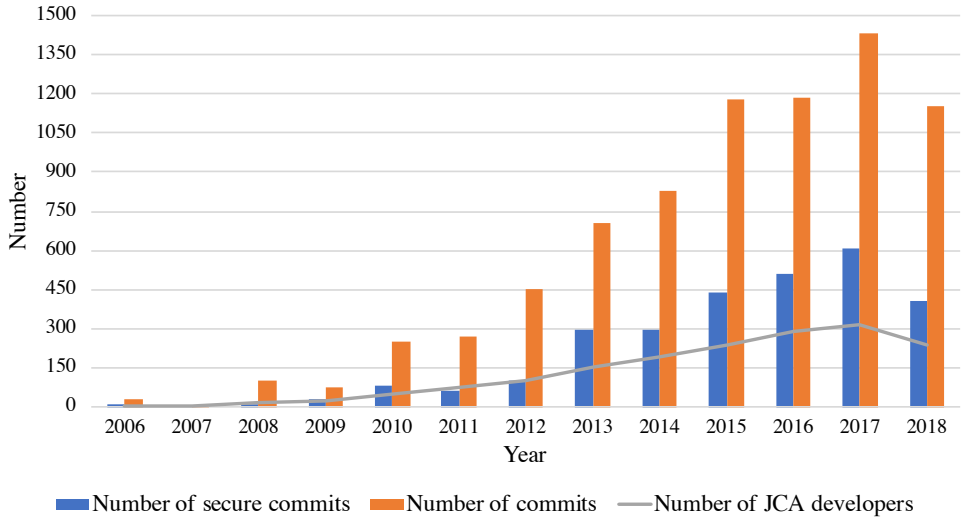

Figure 12: The number of secure, total number of commits, and number of JCA developers in each year

Figure 12 depicts the number of commits and developers in each year. As expected, the increase in the number of developers, the number of commits increases as well. This increase may be due to the emergence of having more cryptographic features in software systems. However, we assume developers do not necessarily learn how to properly use crypto APIs as the number of secure commits is much lower than the total number of commits in each year.

Table 17 depicts the numbers of secure and buggy projects and commits, and their totals. We found an average of 1.7 distinct API usages per project. There is an average number of 3.9 commits per project, of which an average of 1.4 are secure commits and 2.5 are buggy.

Table 17: The status of projects and commits

	Secure	**Buggy**	**Total**
Projects	642	1,682	2,324
Commits (LoC)	3,263	5,897	9,160

Table 18 presents the numbers of distinct developers who always committed secure code, always buggy code, or both secure and buggy code, as well as the total numbers of commits made by each group of JCA developers. We observed that 27.41% of developers consistently used the JCA correctly. The mean value of commits by these developers was 2. About 42% of developers had mistakes in every commit. These developers on average made 3.4 commits. The remaining 491 developers had a mean value of 12.13 commits. These developers did not perform well either: the chance of a secure commit is above 45% only for about half of them (*i.e.*, 52.34%).

Table 18: The status of developers and their commits

	Developer	Commits (LoC)
Always secure commits	440	909
Always buggy commits	647	2,293
Secure and buggy commits	491	5,958

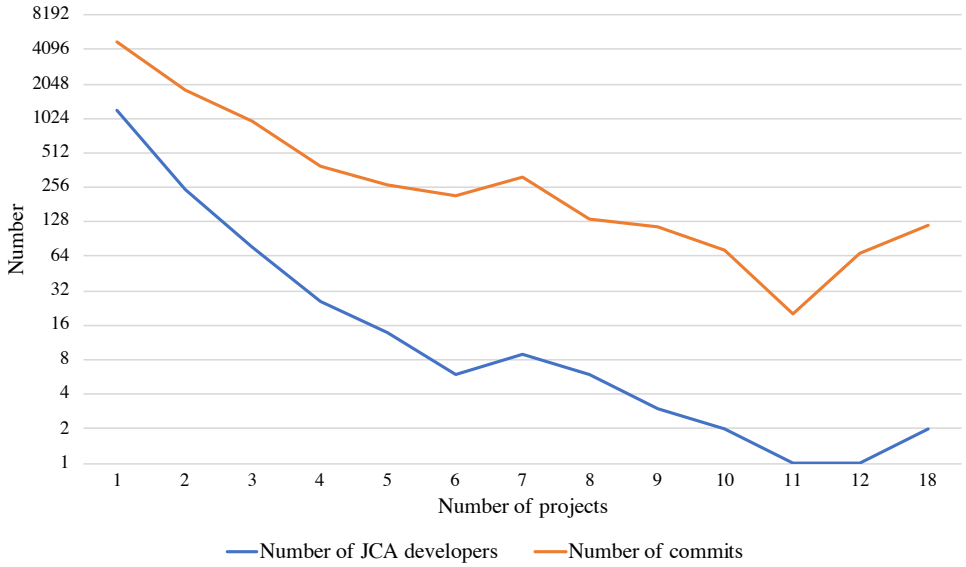

Figure 13: The distribution of developers and their commits in different projects

We also explored how many distinct developers have contributed to different numbers of projects (see Figure 13). We can observe a slight decrease in the number of developers and their total number of commits as the number of involved projects increases. On average 1.18 JCA developers contributed to each project, and each developer contributes to an average of 1.4 projects. The highest rate of contributions to JCA projects belongs to two developers who have contributed to 18 projects and produced 118 commits. In our dataset, 98.13% of the population of developers and 88.48% of commits are involved in five or fewer projects.

Finally, we found that, among the 19 cryptography APIs, *MessageDigest* was used the most *i.e.*, 2,859 times by 790 developers. The *Cipher*, *SecureRandom*, and *SecretKeySpec* are the next top APIs that were used more than 1,000 times. Figure 14 presents the distribution of each API use. We can see that *DSAParameterSpec*, *DHParameterSpec*, *SecretKey*, and *SecureRandom* were almost always used correctly. Similarly, developers showed promising performance in using the *SecretKeyFactory*, and *MAC* APIs, *i.e.*, at least 87% usages were correct. Developers seem to have severe difficulties in using about half of the APIs whose correct usages were less than 25%. The correct usages

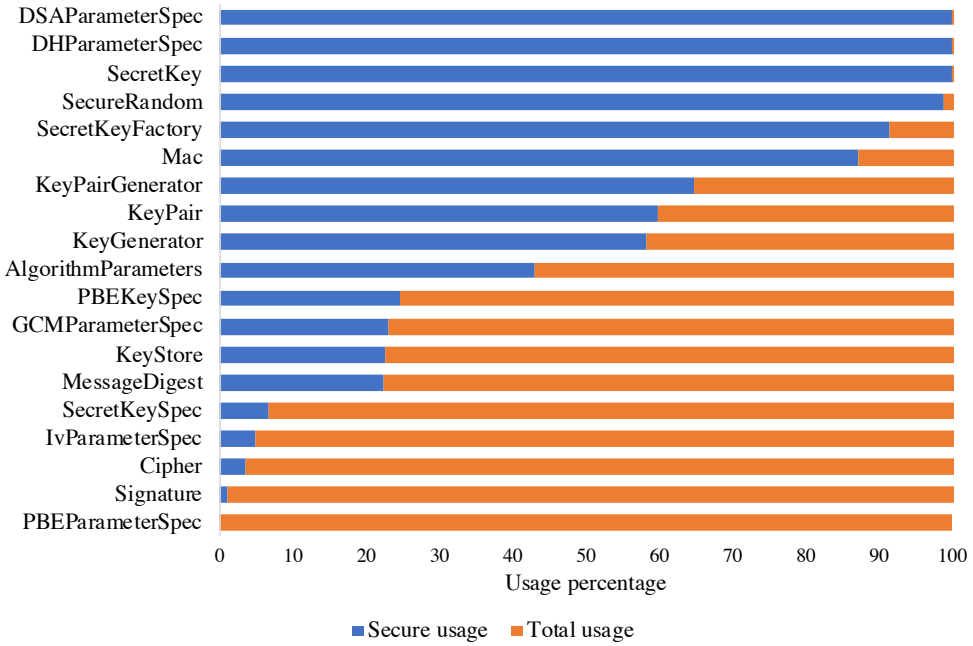

Figure 14: The secure versus total number of each API use in percentage

of five APIs namely, *SecretKeySpec*, *IvParameterSpec*, *Cipher*, *Signature*, and *PBEParameterSpec* were at most 6.58%.

We found that, on average, of 3.9 crypto uses in each project, 2.5 are not secure. Developers have difficulties in using certain APIs, which require further investigations at API level to understand the reasons underpinning this dilemma.

Factors Influencing Developer Performance

We define a *performant developer* as one who makes more secure than buggy commits. Our assumption is that by making more commits, contributing to more projects, working with a wide range of crypto APIs to accomplish different scenarios, and finally being engaged more days with cryptographic APIs can perhaps increase the experience of developers. We therefore collected four factors, namely the numbers of (1) JCA commits, (2) APIs used, (3) projects, and (4) days a developer committed, and studied whether these factors, which we assumed they account for developer experience, have an impact on the performance (number of buggy and secure commits) of developers in using cryptography.

We studied the correlation only between each of the four aforementioned factors, and the numbers of secure and buggy commits so as to find factors highly correlated with developer performance. Both visual inspection and application of the Shapiro-Wilk test confirmed that our data are not nor-

Table 19: The Spearman correlation matrix

	# JCA commits	# Project	# Days	# API used	# Secure	# Buggy
# JCA commits		0.42	0.47	0.75	0.53	0.74
# Project	0.42		0.55	0.28	0.3	0.27
# Days	0.47	0.55		0.43	0.34	0.31
# API used	0.75	0.28	0.43		0.6	0.51
# Secure	0.53	0.3	0.34	0.6		-0.051
# Buggy	0.74	0.27	0.31	0.51	-0.051	

mally distributed. We therefore used the Spearman correlation which does not assume that the data follow a normal distribution [158]. For every two variables, it generates a number between 1 and -1 depending on whether the relationship is positive or negative, respectively.

Table 19 depicts the correlation matrix between pairs of variables. The variables with a correlation score greater than 0.5 or less than -0.5 with secure and buggy commits are the number of JCA commits and the number of JCA APIs used. In the following we study whether these two factors really account for developer performance.

Number of commits

We grouped developers by quartiles of a boxplot which reflects the distribution of developers based on their numbers of commits. Correspondingly, the numbers of their commits can be grouped into three categories ranging from 2 to 4, 5 to 8, or 9 to more number of commits.

Figure 15 presents these groups, and the numbers of secure and buggy commits in each group. Visually, it is clear that all three groups exhibit an increase in the numbers of secure and buggy commits as the total number of commits increases while the median and the average numbers of buggy commits are always more than those of secure commits. In the first group, the median of secure commits is zero while it improved in the latter groups. In the last two groups with more commits, the upper whiskers of secure commits show that the maximum number of secure commits is less than the maximum number of buggy commits.

To establish statistical evidence, we pose the following null hypothesis: *the groups of commits are from identical populations*. To avoid the dispersion of the absolute numbers of secure and buggy commits, we evaluated this hypothesis from the *developer performance* perspective *i.e.*, the number of secure commits divided by the total number of commits by each developer. Therefore, we checked whether the performance of developers in the different groups is the same.

We used a rank-based nonparametric Kruskal-Wallis test since the three groups did not have a normal distribution and this violates one of the assumptions of the one-way ANOVA test. The test resulted a p-value of 0.001315 (chi-square 13.269, and df 2), which indicates strong evidence that at least one of the groups is different.

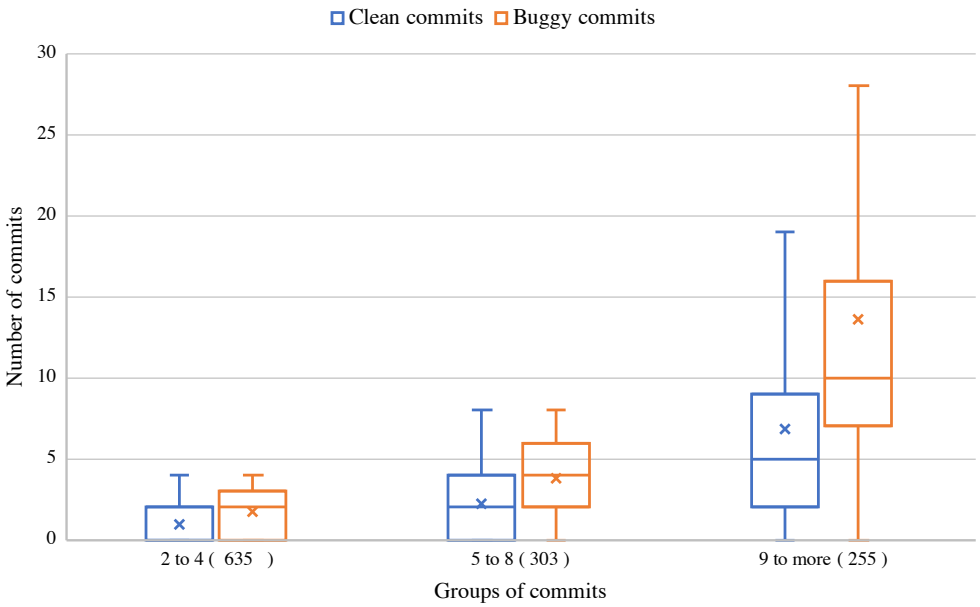

Figure 15: Secure and buggy commits based on the number of JCA commits grouping

Table 20: The Wilcoxon signed rank test result for the performance vs. counts

	First group	Second group
Second group	0.0273	
Third group	0.0018	1.000

We carried out post hoc analysis to find out which pairs of groups are significantly different. We used pairwise Wilcoxon signed rank comparison with the Bonferroni adjustment for the p-value. Table 20 presents the result. We can verify that the first group has a significant difference compared to the second and third group (the p-values are considerably less than 0.05). Nevertheless, the second group comes from an identical population as the third group.

Figure 16 presents the performance of developers versus the number of commits they have made. In the plot, we can see that many developers with a low number of commits have a performance close to either zero or one. The reason for this is clearly that, with few commits, every secure or buggy commit has a much greater impact on the performance than for developers with many commits. For this reason the first group, with only few commits, has a different profile than the others. The trend line shows that with the increase in the number of commits the performance slightly decreases; we believe that with more commits, more APIs are used, which consequently increases the chance of bugs. Therefore, if we do not consider the first group, we can accept the null hypothesis, and conclude that the performance of developers in the remaining groups is identical. In other words, we cannot conclude that

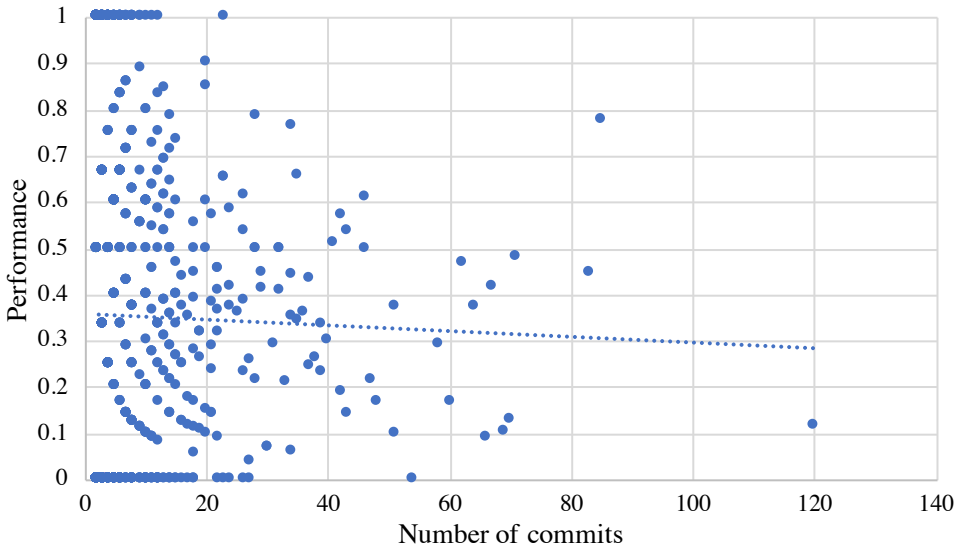

Figure 16: Performance vs. number of JCA commits

developer experience, as measured by number of commits, clearly influences their performance.

Although the number of commits is correlated with the number *of secure and buggy commits, we did not observe any correlation between the number of commits and developer performance,* i.e., *the* fraction *of secure commits.*

Number of APIs

We grouped developers by quartiles of a boxplot, which reflects their distribution based on the number of different JCA APIs that they use. Correspondingly, the numbers of their commits can be grouped into three categories ranging from 1 to 2, 3, or 3 to more APIs. Figure 17 presents these groups, and the numbers of secure and buggy commits in each group. Similar to the number of commits, the median number of secure commits in the first group is zero and it increases as the number of APIs increases. The mean value of buggy commits in each group is always higher than secure commits and the maximum value is notably higher than secure commits. To establish statistical evidence, we computed the performance of each developer (*i.e.*, secure commits divided by total number of commits), and used this metric instead of the absolute number of secure and buggy commits. We define the following null hypothesis: *The performance of developers is similar in groups that use a different number of APIs.* The Kruskal-Wallis test shows a significant difference in at least one of the groups. In particular, the p-value of 2.956e-06 (Chi-square 25.464) is not even close to the cutoff value (*i.e.*, 0.05), which shows a significant difference in the three groups.

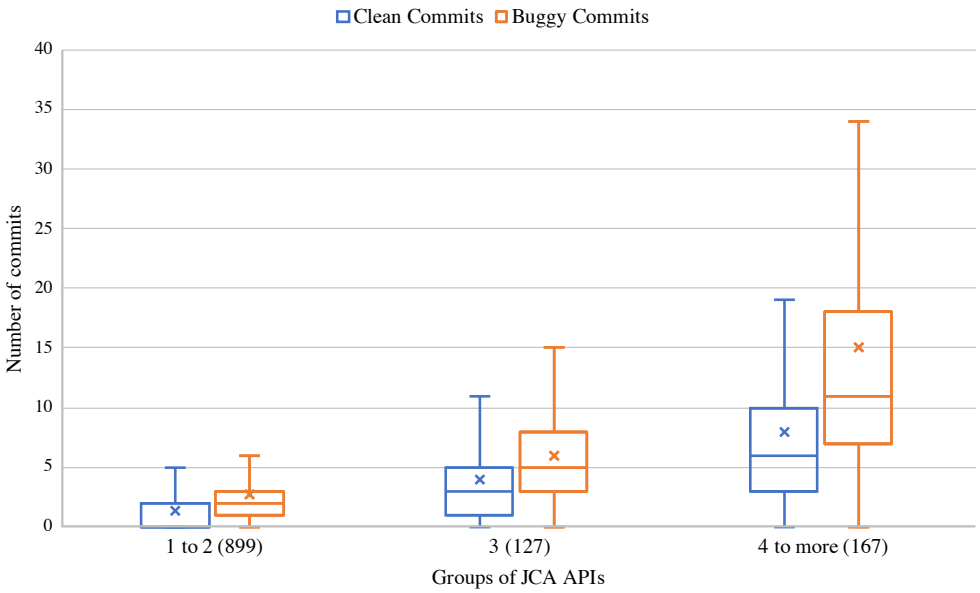

Figure 17: Secure and buggy commits based on API grouping

Table 21: The Wilcoxon signed rank test result for performance vs. API

	First group	Second group
Second group	0.00056	
Third group	0.00024	0.19557

The Wilcoxon signed rank test shows that the first group has a considerable difference with the other groups (See Table 21). Further investigation showed that developers in the first group mostly overlap with those with the lowest number of commits (*i.e.*, 2 to 4 commits). As in the previous subsection, we observe that every commit has a much higher impact on performance for these developers compared to other groups, leading to the difference of the first group with others. If we account for this phenomenon, we cannot reject the null hypothesis, and conclude that the performance of developers who use different APIs is identical. Figure 18 also shows the performance of developers versus the number of APIs that they used. Clearly, there is no sharp negative or positive trend in the performance of developers can be seen as the number of API increases.

Even though the number of used APIs are strongly correlated with the number *of secure and buggy commits, we did not observe any correlation between this factor and developer performance*

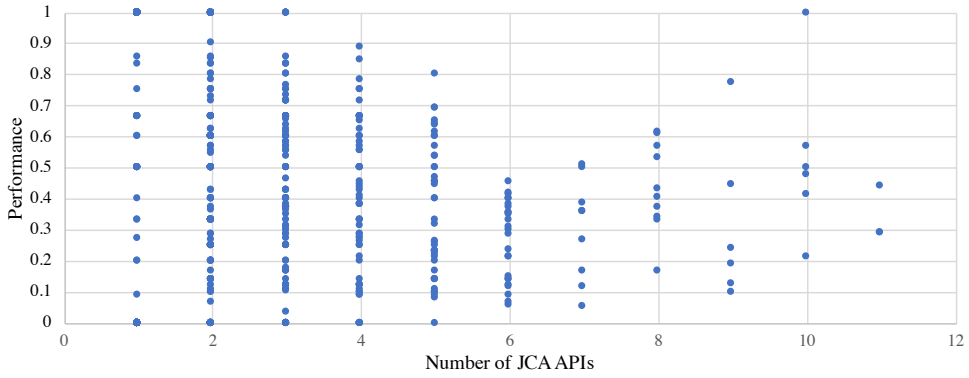

Figure 18: Performance vs. number of APIs

5.2.2 Developer feedback

Out of 216 repository maintainers, 76 did not respond, and 140 reacted to the issues within 20 days. Among all the reported misused APIs, *MessageDigest* had the greatest number of submitted and received responses. *KeyStore* and *SecretKeySpec* are respectively in the second and third place with 22 and 15 received responses from 32 and 31 submitted issues. Following that, *Cipher* and *Signature* were the last two APIs that had been reported in more than 10 submissions. The rest of the APIs had only a few submitted and received responses.

We evaluated all of the responses for 140 repositories in order to identify developer perceptions concerning cryptographic APIs. This widens our views on how knowledgeable developers are when they misuse a crypto API in their code.

We realized that only a tiny fraction of repository maintainers (*i.e.*, of seven repositories) agreed to fix the issues, and a large number of maintainers (*i.e.*, of 46 repositories) disagreed since the context where the misuse occurred was not considered to be security-sensitive. Fortunately, 32 repository maintainers were interested in starting a dialogue about exactly why a given issue can cause problems, and whether the associated risks can arise in practice. We present eight main categories of responses in the following and highlight the key findings of each category in a box.

Personal repository: We received three responses indicating that the target repository is for a personal use. A contributor said that *"the project is meant to be used for educational purposes and intentionally contains some vulnerable examples."* Another mentioned that *"the project is created for internal use and no issue will be addressed."*

People are not aware of the impact these issues could have on those who rely on online examples, as their repositories are publicly accessible. Another facet could be that they are not concerned about security when a program is being used on a very small scale.

Will fix later: Developers of seven repositories replied that they will fix the reported crypto misuses later without asking for any further explanation. Three of them replied that those misuses do not affect the functionality of the program and are not urgent to be fixed.

Developers often underestimate the impact of a crypto misuse.

Request for pull: Developers of 17 repositories suggested to create a pull request. For instance, a contributor responded that *"I'm not sure if I understand the problem. I am not a cryptologist."* We believe a lack of knowledge in this area exists that may cause developers to blindly accept a pull request. The inevitable consequence of blindly accepting a pull request could adversely affect the security of the final software, for instance, an adversary may submit a downgrade to the existing security mechanisms in a project.

There is a risk that developers who lack security knowledge blindly accept security-related pull requests.

Refer to the main library: In five cases developers used an open-source library, and asked us to report the issue to the library's repository.

Unfortunately, developers seem not to be concerned about security risks associated with external libraries.

Repo is not maintained: We learned from the responses that 15 repositories are not maintained anymore.

Inactive projects are common in the open source community, for example due to a lack of financial support. However, as long as the code is available online, novice developers may rely on open-source projects irrespective of how active the projects are.

Consult documentation: Developers of 10 repositories were not completely certain about how the APIs should be used securely. They either asked us to read the API documentation or quoted a relevant part of the documentation in their responses. For instance, we suggested not to use *java.lang.String* as the second parameter for *KeyStore*. This parameter is the password parameter. Developers replied that *"according to the documentation, the parameter is a String, so why should it never be String?"* One responded that *"MD5 is still supported by java according to the Java documentation."* Another one asked us to provide him with the correct use of *Signature* API as he did not know how to fix it. A developer referred to the official Java documentation to express his trust in using the *SHA1withRSA* algorithm in the *Signature* API. Developers of two repositories were not convinced to stop using *NoPadding* in the *Cipher* API. They noted that using an *empty string* in Java 8 can cause a run-time error and they cited *Cipher*'s page in Java documentation.

Developers have confidence in official documentation, but security concerns are mainly absent in such resources.

Uncertainty: We found that many developers (*i.e.*, of 32 repositories) asked us to provide a clarification. Some developers referred to blog posts where the *KeyStore* API was misused by converting a string variable to an array of characters, *i.e.*, `password.toCharArray()`, and passing it as the second parameter to the API. A common skepticism was about which algorithm is safe to use in *SecretKeySpec* and *KeyPairGenerator*.

A few developers asked how the misuses can be exploited in real life. For instance, a contributor was not convinced about the nature of the "misuse" as it was not clear to him how a wrong transformation mode in the *Cipher* API can be exploited in his application.

As expected, developer uncertainty regarding the correct way of using an API securely is related to either the right method call or the secure algorithm name

Consider the context and disagreement: The majority of responses (46) are connected with the context of the code. A large number of these responses were mainly related to the *MessageDigest* API since *MessageDigest* was used more frequently than any other crypto API in the analyzed projects. This is because *MessageDigest* can be used in many different scenarios such as authentication, checksumming, archiving, or in combination with other algorithms.

One common complaint was that MD5 or SHA1 were not being used for security purposes. They had been used for archiving or producing hashes for non-security use cases. For instance, one developer mentioned that the Redis API needs to generate checksums using SHA1. As another example of non-security usage, one repository used SHA1 for opening a handshake in WebSocket and the contributor referred to the RFC 6455 section 1.3 for further information. Moreover, three repository maintainers replied that instead of using `String.hashCode()`, they used SHA1/MD5 as an internal identifier, *i.e.*, generating a normalized document ID based on the URL of the given document. Another developer stressed that they use MD5 in order to track if the template source has been changed or not. A group of developers used MD5 to get a hash of an email address to produce the avatar URL of the user. SHA1 was also used in a *for* loop to generate fake data to be stored in a file in a repository. Contributors to a repository pointed to a code comment preceding the SHA1 usage that clearly says that *SHA1 was used only to generate a single hash for the entire contents of a folder and it is absolutely sufficient.*

Some contributors complained that the critiqued code is very old and the context is not security-focused, *i.e.*, a decade old, and running a static analysis is not a good measure to find misuses. In some responses, they did not exactly mention what the context was and only replied that the context is not security-sensitive. For example, we manually checked the codes of the repositories and

found that *MessageDigest* was used for purposes such as hashing parameters, *i.e.*, album name, in the URL or to cache the unpacked ZIP file and avoid multiple extractions.

In one repository, developers indicated that there was a code comment explaining why the *Signature* API had been used in an insecure way. A developer mentioned that although *KeyPairGenerator* accepts many algorithms such as RSA, DSA, and Diffie-Hellman, in this project the APNs protocol demands *KeyPairGenerator* to generate key pairs for the Elliptic Curve algorithm.

One contributor cited a blog post where the blogger discussed that SHA1 is still usable regardless of the existing collision vulnerability. He insisted that they will continue using the algorithm until a serious security problem is raised by using SHA1. Another developer stressed that MySQL authentication plugins do not support the usage of SHA-256 and accordingly, SHA1 was used in their project. In contrast, official MySQL documentation added that since MySQL 5.6, the sha256_password authentication plugin is supported. With regard to the misuse of MD5, a developer replied that he cannot change the algorithm name as the remote endpoint requires an MD5 hash and he is not able to change it on the remote endpoint.

Developers mainly argued that their context is not related to security. The use of security APIs to produce hashes was the most common non-security related usage.

We witnessed that some APIs were more prevalent compared to others in our reports. For example, the *MessageDigest* API was seen more than other APIs in all response categories except for "Personal repository." In *MessageDigest*, the most common misuse type is *constraint error* in which developers used the MD5 or SHA1 algorithms to compute hashes. As hashing algorithms could be used for non-security purposes, many maintainers were expected to note that the context is not relevant. *KeyStore* and *Cipher* are the second most frequently seen APIs in the responses. Repository maintainers mostly asked for clarification or referred to the documentation of Java for the misuses of *KeyStore*. The *Cipher* API had the majority of misuses linked to its first argument. This occurred due to the diversity of options in the transformation string. A transformation string includes the name of a cryptographic algorithm (e.g., AES or DES), and may be followed by a feedback mode and padding scheme, *e.g.*, *algorithm/mode/padding*. On the whole, developers had difficulty in understanding what constraint they should pass to the crypto APIs or how to create an object securely in order to pass it to another crypto API.

The responses of maintainers highlight the fact that some developers are fully aware of what they are doing, whereas others have doubts concerning the correct way of using such APIs. As a result, blaming only developers for the found crypto API misuses cannot be correct. To highlight the leading causes of crypto misuses, we identify various influential factors that must be taken into account. The *age of a project* can be an essential factor as security

standards can evolve over time. Another factor is to blindly rely on the *use of third-party libraries*. Developers need to spend more time and select libraries with higher credibility and support. *Official documentation* and *unofficial online documentation*, such as Stack Overflow, can have pernicious effects on developer choice. This impact can be ruinous when the developer lacks the minimum knowledge in the domain of cryptography, *e.g.*, choosing ECB mode from the examples provided by the official documentation. Furthermore, such developers may not be able to make a good use of static analysis tools to resolve their crypto problems. On the other hand, *statically checking the developer's code without considering the context* yields misleading results and could not be a dependable measure for developer performance in this domain. For instance, recent program analysis tools consider SHA-1 to be insecure. Such tools produce alerts if a developer cautiously uses three nested SHA-1 to produce a hash while the application is not, indeed, exposed to security threats. Lastly, even though some developers were aware of the right crypto usage, the widespread use of such examples on open-source projects may have profound implications on inexperience developers.

5.2.3 Developer survey

We discuss our findings from the developer self-reported knowledge perspective. We then compare variables related to each participant characteristic with the reported knowledge level. The participants rated their knowledge in cryptography as 21% (*i.e.*, 21) *somewhat knowledgeable*, 60% (*i.e.*, 58) *knowledgeable*, and 19% (*i.e.*, 18) *very knowledgeable*. We respectively assign these participants to low (21), medium (58), and high-profile (18) groups in this domain.

Developer demographics

We analyzed whether age, experience in programming, or education are correlated to knowledge in cryptography. Unsurprisingly, the older participants are, the more experienced they are in programming. Although experienced participants exist in every knowledge group, there is a clear distinction between high to medium-profile developers and low-profile developers (See Figure 19). All participants from the high-profile group have more than 10 years of experience in programming, and there are no participants from high or medium-profile groups with fewer than 5 years of experience in programming. The same pattern was seen among the groups for years of programming experience in Java. The education level is almost evenly distributed among the three groups. Of the seven Ph.D. participants, five belong to the medium-profile group.

Developers with a high or medium level of knowledge in cryptography have more years of experience in programming and Java.

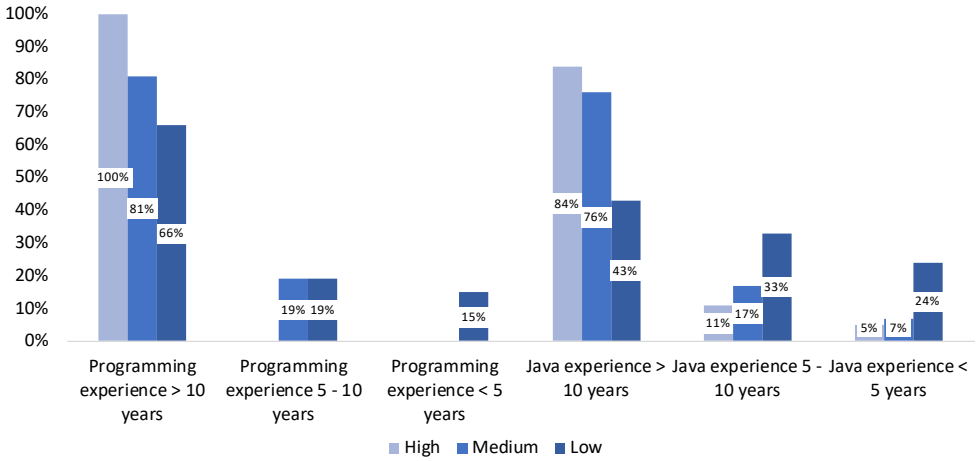

Figure 19: Years of experience in programming and Java

Developer characteristic

To have a glance at developer characteristic, we examined the relationship between developer knowledge and any of the following: security course attendance, experience in using crypto libraries, background in IT-security, developer security concern, ways of solving crypto problems and evaluating a crypto code, and working as a source code auditor.

With regards to security or cryptography course attendance, the more knowledge developers had, the more courses they attended. The high group has the highest number of participants (22%) attending *both courses*, while there are 9% and 14% such participants of the medium and low groups, respectively. The number of participants who attended such courses is below 40% in all groups.

Although several participants (*i.e.*, 15) responded that they worked as a secure code auditor, they belong to different knowledge groups, *i.e.*, high (3), medium (10), and low (2). Just over a half (*i.e.*, 53%) of such participants never attended a security or cryptography course, whereas the rest (*i.e.*, 7) attended security and cryptography courses.

In total, 38% of participants (*i.e.*, 37) reported that they had background in IT security. We observed that a very large proportion (86% and 61%) of the low and medium-profile participants had no background in IT security. In contrast, 61% of the high-profile developers expressed their security-relevant background, achieved through various methods *e.g.*, bachelor and master thesis on software security, or personal enthusiasm and self-study in software security.

Notably, all respondents from the high-profile group stated that they have at least two years of experience in using crypto libraries, whereas 52% and 21% of the low and medium-profile developers are not highly experienced (*i.e.*, <=

2 years) with such crypto libraries.

We received 47 responses regarding the hindrances developers encounter when dealing with cryptographic tasks. The majority of them mentioned two key obstacles: the first was the high complexity of using crypto APIs. For instance, a developer mentioned that *"Wide variety of configuration options"* is troublesome. The second obstacle concerned unreliable sources and lack of security experts in teams. For example, one participant blamed *"Poor Java docs"*. More than half of the 47 respondents (63%) were from medium-profile developers, and 17% of them were from the high-profile group. This means that developers who still feel confident about their knowledge in cryptography struggle to use them in the wild.

A large proportion of low-profile developers have no IT security background, while more than half of the high-profile developers do have such backgrounds. High-profile developers have slightly better records in the security/cryptography course attendance and considerably more years of experience in using crypto libraries than other developers, whereas medium and low-profile developers are almost similar. Developers from all groups mainly complained about the complexity of crypto APIs and insufficient documentation.

With regards to developer security concerns, 81% of developers rated their concern as *important* or *very important*. Remarkably, 61% high-profile developers are *very* concerned with security while 33% and 43% from low and medium groups reported the same level of concern (See Figure 20). Only one high-profile developer is *somewhat* concerned with security while 19% of low- and 17% of medium-profile developers reported the same level.

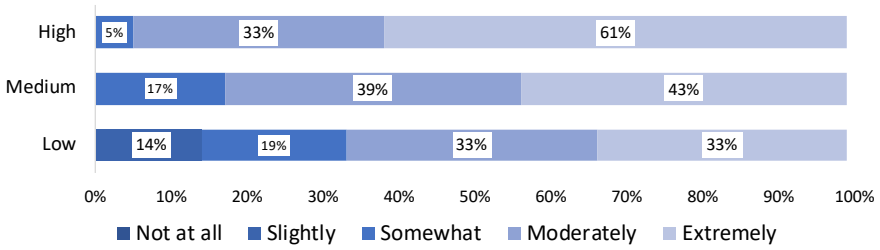

Figure 20: Security concern by participants

Participants reported various information sources to solve the challenges in a cryptography-related task (presented in Table 22). In particular, the primary information sources seem to be online. In each of the three groups, *websites found on search engines* and *Stack Overflow* are among the top three preferred approaches. It is noteworthy that the role of information security experts is not as commonly cited as other approaches. Participants only from high- and medium-profile groups consult with a security expert, and none of them are from the low-profile group. High-profile participants prefer discussion with

colleagues, security consultants, and crypto stack exchange more than other participants.

Just over half of the developers in all groups *only* evaluate their code manually (See Figure 21). Remarkably, the total number of participants who use a static analysis tool is fewer than one-fifth (*i.e.*, 18) of the total number of participants. In more detail, high-profile developers had the highest usage of static analysis tools (28%), while low-profile developers had the lowest usage of such tools (5%). Of ten developers who do not evaluate the security of their code, only one belongs to a high-profile group.

*Unlike others, nearly all (*i.e., *17) high-profile developers are* extremely *or* moderately *concerned about security. Developers mainly solve their crypto problems on Stack Overflow or websites returned by search engines. The high-profile group benefits more from security consultants, discussions with colleagues, and crypto Stack Exchange to resolve crypto problems. Developers mainly evaluate crypto-related issues manually rather than using analysis tools. All high-profile developers evaluate their crypto code and they use static analysis more than others. In contrast, low-profile developers tend to use static analysis tools less than others, and 16% do not evaluate their code.*

Table 22: The information sources that developers use - bold items are the highest in each row

	High	**Medium**	**Low**
Websites on search engines	83%	88%	**86%**
Stack Overflow	72%	%74	**%80**
Crypto Stack Exchange	**34%**	33%	19%
Security consultant	**28%**	10%	0%
Books	33%	**41%**	14%
Discussion with colleagues	**77%**	38%	57%

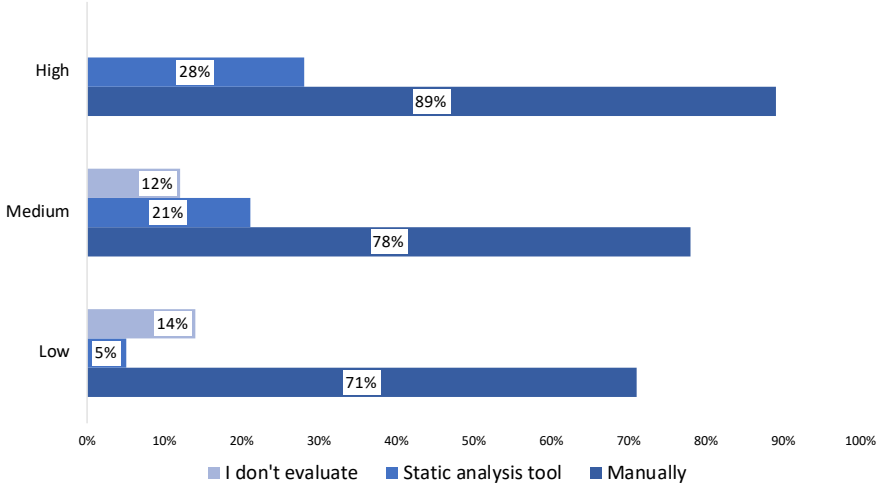

Figure 21: How developers evaluate a crypto copy-pasted code

Company characteristic

Developers were asked to respond to how concerned their companies are with security (See Figure 22). More than half of the companies (*i.e.*, 72) were concerned (important and very important) with security. Furthermore, 71% (*i.e.*, 69) of the companies do not have any security consultant in their team and 70% of the companies have no or fewer than 30% of developers responsible for secure development. Disappointingly, we learned that 57% (*i.e.*, 53) of respondents do not receive security training at workplace. A yearly training interval is the most common approach (27%), and a two-year training interval is the least common approach (6%) among companies.

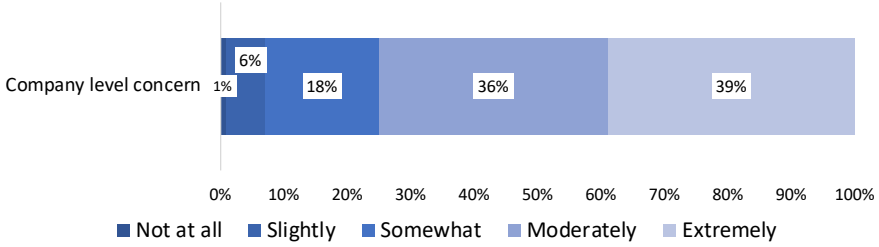

Figure 22: Security concerns by companies

We observed the mapping of "in-site consultant", "regular training", and "responsible developers" with developer knowledge (See Figure 23). As expected, in all three factors, high-profile developers reported positive responses slightly more than the medium-profile group, and noticeably more than the low-profile group. However, the medium and low-profile developers are very similar concerning the in-site security consultant.

The majority of companies are concerned with security but the lack of security consultants, regular security training, and security developers are inevitable. In particular, high-profile developers benefit more from the three factors.

Discussion

Developer background: Acar *et al.* assigned 307 active GitHub users to complete several security-relevant programming tasks and surprisingly, found no statistically significant differences concerning functional correctness and security perception among the participants who registered their status as a student, professional developer, or those who had security background [3]. Interestingly, years of experience was not an effective factor for security perception. Oliveira *et al.* designed a study for 109 developers to use some APIs that had some blind spots, *i.e.*, containing underlying causes to misuse an API, and some easy to use ones [115]. The results show that developer expertise and experience did not predict their ability to identify blind spots. In

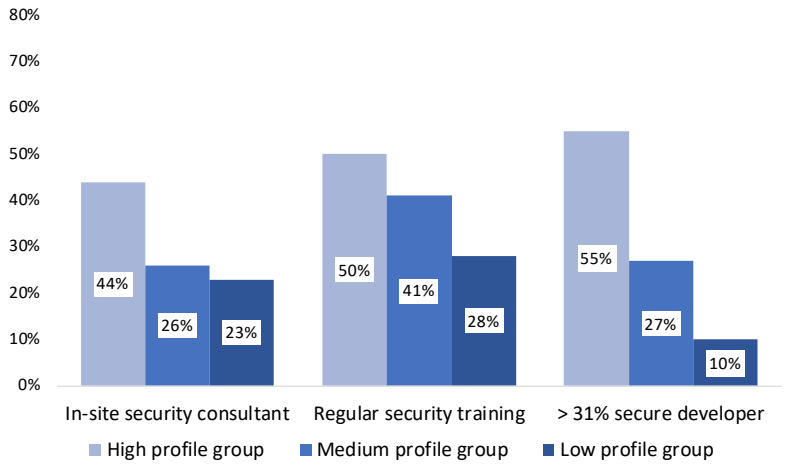

Figure 23: Security support provided by participant companies

another study, the outcome of an experiment with 54 professional and inexperienced developers for writing security-related code explains that development experience is not a decisive factor for code security [2]. Nadi *et al.* conducted two surveys and asked developers about the issues they face when working with crypto tasks [112]. The participants mentioned several types of issues including lack of documentation, difficulty in API use, and indirection between the APIs and the underlying implementation. The authors also realized that developers from various knowledge level groups still face the same types of issues in cryptography. Robillard *et al.* conducted surveys and interviews with Microsoft developers, and realized that poor documentation is a major learning obstacle for learning APIs [130]. To alleviate unsafe coding practices, security training courses, *e.g.*, secure programming, are more effective compared to general security training [111].

Security tool adoption: Johnson *et al.* conducted interviews with 20 developers to understand the determinant factors why static analysis tools were not adopted by many developers [86]. Participants mentioned reasons such as the high rate of false positives, the way that warnings are displayed, faulty integration of the tool into the development process, lack of detailed explanation of bugs with automatic fixes, and not including understandable configuration options in the tool for all levels of developers. Other researchers investigated the reasons for a low rate of security tool adoption [13] [14]. They found that organization and team policies affect the usage of security-related tools and larger organizations use security tools more than small ones. The greater adoption of security tools can be influenced by factors such as the culture of the company, security concerns, training, and dedicated security and testing teams.

Ways of solving crypto problems: Even though using Stack Overflow might help the functional correctness, it leads to more insecure copy-pasted

code snippets [2]. Ye *et al.* worked on a system called insecure code snippet detection (ICSD) to detect the imminent insecure code snippets on Stack Overflow [157]. In a survey with 87 Stack Overflow visitors, they reported outdated answers, wrong solutions, and buggy code. Their results cast light on the choice for finding programming solutions, and how often they reused a prepared solution. Stack Overflow had the first rank in finding solutions. Acar *et al.* conducted a comprehensive study by surveying 295 application developers, and a lab study with 54 Android developers (professionals and students) in which they were allowed to resolve coding issues with one of the following four means: any resources, Stack Overflow only, official Android documentation only, or books only [2]. Their findings suggest that developers use Stack Overflow as a major source. Interestingly, developers who could use any resources had similar performance (functional and security correctness) to those who were assigned to use Stack Overflow only. The lack of an official role in organizations as security champions/consultants is evident, and oftentimes this role is given to someone on the development team with limited security knowledge. By hiring security consultants, managers can gain positive impacts from the resulting security level of products, and security testers would largely benefit from the presence of such consultants [140] [123].

Security concern: Witschey *et al.* conducted a study to understand what factors affect the usage of security tools [150]. Strangely enough, being more concerned about security did not lead to greater security tool usage while having training or academic background in the security field did. Research indicates that some organizations use external resources, *e.g.*, penetration testers, to encourage developers to pay extra attention to security in development, however, without strong support, the motives tend to lose priority compared to the important functional requirements [151]. Likewise, managers sometimes are obliged to make vital decisions, such as releasing the code with some known problems, due to business forces [140].

Security training: The need for regular information security training is undeniable in companies [143]. From the training frequency viewpoint, quarterly security awareness training is recommended to renew employee knowledge concerning the latest threats and trends, and in case some difficulties exist, biannual training could be the minimum required time frame [57]. Puhakainen *et al.* stressed that information security trainings and communication efforts should be continuous and integrated into the organization's usual communication efforts otherwise security policies lose their efficacy [125]. According to the SANS Institute, a security awareness program should consider who is going to be in the training course, which topics are suitable for the audience, and ultimately how participants engage in order to identify how frequent security training should take place.[7]

[7] `https://www.sans.org/security-awareness-training/blog/wrong-question-how-long-should-security-awareness-training-be`

By studying the literature we found clear evidence to corroborate the findings of this study. Each of the discussed studies either solely explored one factor and obtained similar results or they emphasized the importance of the studied factor to improve the state of developers in security or cryptography. We believe that even though 81% of the participants, as well as 75% of their companies, are utterly concerned, *i.e.*, *important* or *very important*, about security, the practices of the participants and companies do not accord with their grave security concern. However, conducting a survey has some inherent limitations. To profoundly investigate this matter, conducting interviews with some of the participants who provided their email addresses can be beneficial to inspect organizational policies, project-level limitations, and objectives.

5.2.4 Experts' practices

We explore the usage of crypto responders' programming languages and crypto concepts on Stack Overflow and GitHub.

Stack Overflow

We extracted 804 top crypto responders in which 319 users shared their GitHub profile on Stack Overflow. We fetched the crypto discussions of the 74 users (the sample size), extracted their provided answers, and stored the names of the programming languages involved in the discussions. In total, 55% of discussions were about Java. A user could have participated in various discussions wherein different programming languages were involved. We therefore considered all those languages as being the areas of the user's crypto knowledge. The median and mean numbers of programming languages used on Stack Overflow are 3 and 2.7, respectively.

More than four-fifths of the developers (*i.e.*, 65) participated in discussions where the three crypto-concepts were discussed. Similar to programming languages, a user can provide answers for a discussion in which the knowledge of a concept or mixed concepts are required. For instance, we considered *(1) hashing (2) sign/verification* for the discussion (ID:33305800) on Stack Overflow since a user was confused about the differences between hashing with SHA256 and signing with SHA256withRSA.

GitHub

Of 319 users with GitHub profiles, 189 had made crypto contributions to public repositories on GitHub. To conduct our manual analysis, we randomly selected 74 users from the 189 crypto developers. We extracted the names of programming languages where crypto APIs were used. The median number of programming languages used on GitHub is 1 and the mean is 1.4. In all 74 cases, the number of programming languages and crypto concepts on Stack Overflow was higher than or equal to the same groups of data on GitHub. For

instance, developer A participated in discussions where three languages (*i.e.*, C++, C#, Java) were involved as well as the three crypto concepts while the same developer only used Java crypto APIs for hashing purposes on GitHub.

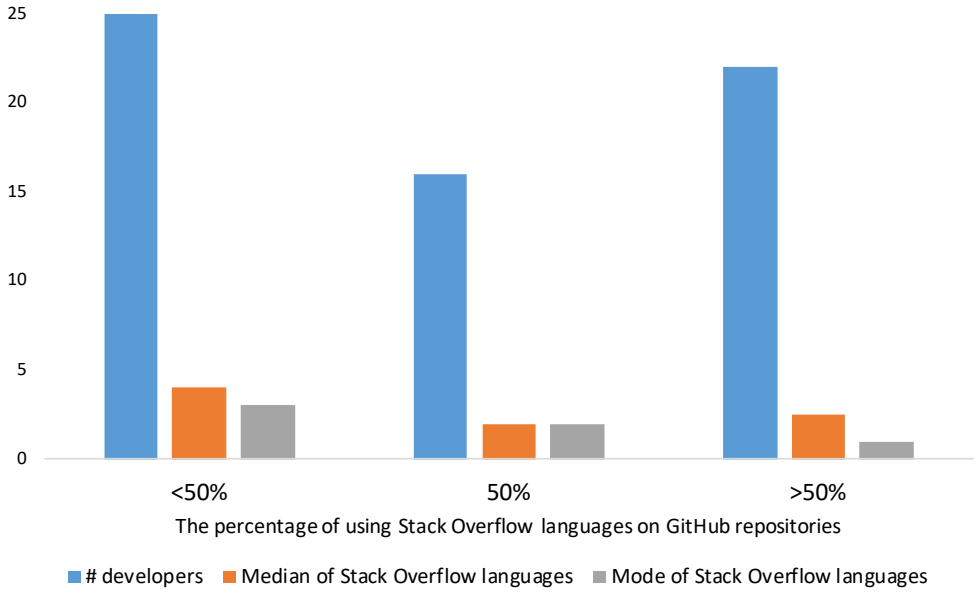

Figure 24: The number of developers based on their percentage of Stack Overflow programming languages usage in GitHub repositories

Mapping result

Interestingly, we realized that 63 (*i.e.*, 85%) of such users had used at least one language that matches their crypto activity on Stack Overflow. Such agreement implies that the users are confident in those languages. We split the 63 developers into three groups: those who used fewer than 50% of the languages in their GitHub open-source projects (*i.e.*, 25), those who used half of the languages (*i.e.*, 16), and those using more than 50% of the languages (*i.e.*, 22) (See Figure 24). In particular, more than half of the developers (*i.e.*, 38) had crypto contributions for either half or more than half of the languages that they prefer to provide crypto help for on Stack Overflow. The developers who used fewer than 50% of their Stack Overflow languages in open-source projects constitute 39% (*i.e.*, 25) of the whole.

With regard to crypto concepts, there are six developers who used APIs on GitHub which are related to the three crypto concepts (See Figure 25). There are seven developers who used *signing/verification* and *hashing*, five developers who employed *hashing* and *symmetric/asymmetric*, and only two developers used *signing/verfication* and *symmetric/asymmetric*. The rest of developers only used one of the concepts in the identified projects. They might have

a broader contribution to cryptography in open-source projects, however, it may be due to the limitation of our obtained knowledge concerning their practices on GitHub. On the other side, the manual investigation revealed that, on Stack Overflow, 65 developers participated in all three concepts, seven developers only in symmetric/asymmetric, and only two in signing/verification. Checking the labels of 74 developers, we uncovered that almost all of the developers (*i.e.*, 67 or 90%) worked with at least one common crypto concept on both platforms. Of the 67 users, 30% of them had more than one concept shared on both platforms. The findings imply that developers are confident in programming languages and the crypto concepts as they had relevant experience in practice. Likewise, user satisfaction, such as high upvotes for the responses on Stack Overflow, confirm that the users' guidance is practical and effective in the domain of cryptography.

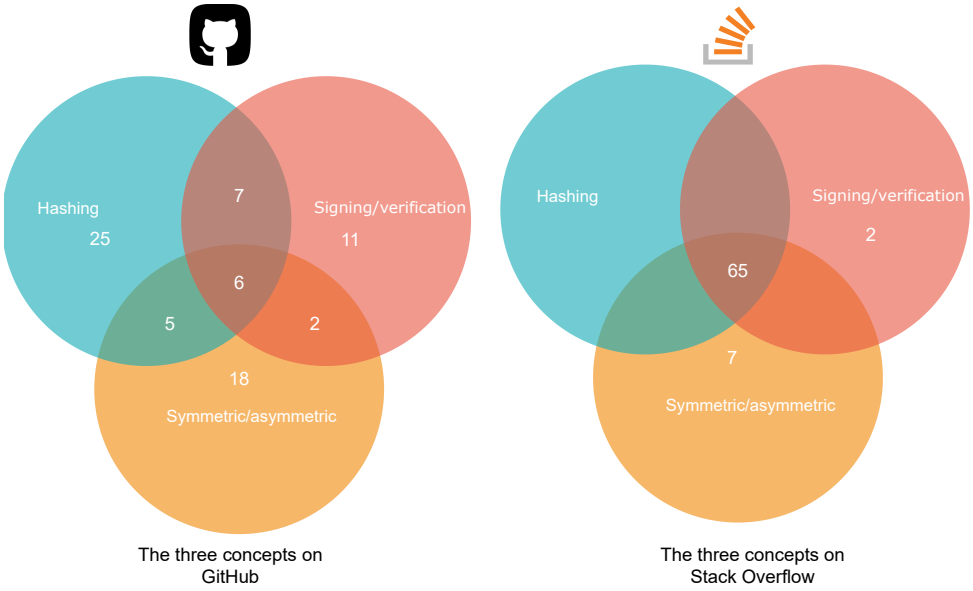

Figure 25: The numbers of developers with experience in each crypto concept on Stack Overflow and GitHub

5.3 Threats to validity

In the following, we describe the threats to validity of each of the explored facets.

5.3.1 Developer performance

We analyzed 2,324 Java projects that use JCA APIs. We found these projects through the contributors of some initial projects that were identified in previ-

ous work. We did not define any criteria for selecting the remaining projects, but they may not represent the whole Java ecosystem. Also, we only focused on Maven projects and did not look for Gradle or ANT projects. We relied on CogniCrypt for misuse detection as it provides state-of-the art static analyses and covers a wide range of API misuses. Whereas certain limitations are inherent in static analysis in general, we did not check for the existence of false positives in the results as the authors have done so and found the tool to be fairly precise. We aborted the analysis of each project after 15 minutes. Increase in the timeout may yield more projects to be analyzed. We used the *git blame* command in order to identify the last developer who has committed a change to that line of code, and did not look into the commit history. Though the last commit can be due to some refactoring or maintenance purposes, we studied several cases and did not find any other changes concerning crypto APIs. Moreover, commits can be an outcome of several works by different people locally where the resulting commit is the only visible one.

5.3.2 Developer feedback

We contacted the maintainers of analyzed repositories and received 140 responses. However, we did not check whether the contributors who replied to the issues are the ones who used the crypto APIs. We only relied on their first reply and did not further pursue the conversation in each issue. In the initial phase, we did not take into consideration how active the repositories are, therefore we had some responses concerning "this repository is not maintained anymore". Nevertheless, such abandoned code repositories could have a negative impact on inexperienced developers.

5.3.3 Developer survey

Our sample of software developers using crypto APIs on GitHub is limited in size. To increase the number of such developers, more crypto open-source projects need to be identified, and associated developers must be extracted. In the survey, there was no participant who reported *no knowledge of cryptography*, and we did not exclude any participant from our analysis. All the participants had used crypto APIs in Java open-source projects, and it was unexpected to receive responses indicating no knowledge of cryptography. Developers, in general, may have different viewpoints on how to evaluate their knowledge in a specific area, such as cryptography. To lower the risk of assumption bias, we provided the participants in the survey with an extra definition of what each level of knowledge is intended to mean, and the four-level knowledge used in a previous study [112]. To grasp the real knowledge of developers, we need to judge developer knowledge based on their real performance, *i.e.*, in a controlled experiment. We only asked the participants about their companies' practices since we did not intend to ask about the names

or web addresses. Even though we received 97 responses from 1103 potential participants, it may be possible that more than one participant refers to the same company.

5.3.4 Experts' practices

We identified 804 developers who were among the top 1% of responders to 64 crypto tags on Stack Overflow. However, we were only able to find 319 of these developers on GitHub, and did not perform any exhaustive search on Google to find more users. A developer may have multiple accounts on GitHub for various purposes but we only consider one account per user. Some users may have private repositories and make more significant contributions to crypto-related projects, nevertheless, such contributions cannot be assessed. We looked into the repositories written in seven programming languages, and did not analyze the remaining repositories. Even though we included popular and default crypto libraries in each programming language, adding more crypto libraries in each programming language can allow a more realistic conclusion to be drawn. This is important, considering that the diversity of crypto libraries in each language is debatable. We used the git blame command to fetch a crypto file's contributors. Consequently, there is a likelihood that the developers who contributed to crypto files had committed to other parts of the file but not to the cryptography parts.

5.4 Summary and conclusion

We investigated 2,324 open-source Java projects whose code contains usages of Java Cryptography Architecture (JCA). We discovered that, on average, of 3.9 crypto uses in each project, 2.5 are not secure, and that developers have great difficulties in using more than half of the APIs. We also studied four factors that rationalize developer experience namely the number of JCA commits, API diversity, the number of projects developers are involved in, and the frequency of committed lines of code. We found that none of these factors influence developer performance in this domain.

We investigated the influential factors in misusing such APIs. We contacted the maintainers of the projects to understand the reasons behind the misuse of crypto APIs, and we classified their responses into eight main categories. The results demonstrate that security hints in API documentation are scarce, misuses are rooted in third-party libraries, or the code context plays a crucial role in using crypto APIs incorrectly. Finally, to support the research community, we publicly share the CryptoMine dataset, including the analysis results, and information about each project such as its metadata information, the precise locations of API use, and the safety status of these APIs, to name but a few.

We surveyed 97 developers, who used cryptography in open-source projects, and studied their security and cryptography practices. Our analyses demonstrate that high-profile developers reported better to the developer- and company-level questions, *e.g.*, security tool usage, and background in IT security. It should be recalled that over 70% of the participants and their companies are utterly concerned about security. Nevertheless, a number of worrisome patterns, *e.g.*, lack of regular security training, security consultants, and low rate of security tool usage, were observed in other participants' responses. The results provide corroborative evidence supporting the outcome suggested by prior research. To further understand the root causes of developer practice in this area, future studies should consider organizational policies, project-level limitations and objectives, and developer expertise in practice.

We conducted a study of the top 1% of crypto responders on Stack Overflow to shed some light onto the adoption of cryptography on GitHub by the top crypto responders on Stack Overflow. In particular, to the best of our knowledge, no previous study has profiled crypto developers across online communities. We found 189 users who used cryptography in open-source projects on GitHub and studied 74 of this population. The results indicate that the majority of analyzed users (*i.e.*, 85%) use the same programming languages for participating in crypto discussions on Stack Overflow and crypto contributions on GitHub. Closer inspection of three areas in cryptography (*i.e.*, hashing, symmetric/asymmetric, or signing/verification) revealed that 90% of the analyzed users had practical experience with at least one of the crypto concepts that they had discussed on Stack Overflow. Collectively, the results demonstrate that top crypto users are consistent with their crypto activity on both platforms, and this provides a basis for further research to investigate the quality of their practical experience.

Chapter 6

Root causes and Remedies

In the last two chapters, we discussed the problems of crypto APIs and common practices of developers in the field of cryptography. We believe that utilizing the experience of crypto experts can substantially contribute to comprehending the problems, providing insights into secure usage of cryptography. Furthermore, thanks to their rich experience in perhaps industry or open-source communities, the crypto experts might offer invaluable recommendations to improve the problem of misusing crypto APIs. To the best of our knowledge, no similar attempts have been made to collect the suggestions of crypto experts with regard to cryptography misuses.

Aside from collecting experts' opinions, another approach to alleviating the problem of misusing crypto APIs is to support developers with security or educational tools. Security tools can be beneficial for developers to understand a number of intricate subjects in order to securely use crypto APIs. However, this may not be possible on a large scale due to multiple reasons, *e.g.*, organizational policies or developer unwillingness. There exist several tools to assist developers in using crypto APIs correctly [127, 43]. These tools nevertheless need to be installed, are dependent on specific programming environment and are relatively complex to learn. More importantly, not all developers are aware of such tools.

We found that an educational pathway might be more intriguing to explore. In this regard, Adamovi *et al.* attempted to lessen the learning curve of cryptography by proposing an interactive tool, called CrypTool [4]. In their experiments, they obtained positive feedback concerning using the tool in teaching cryptography in comparison to traditional techniques. Aydin attentively linked the basics between relevant subjects, *e.g.*, algebra and number theory, that are taught in a mathematics course and several related areas from computer science and cryptography without requiring a lot of prerequisite knowledge for undergraduates [12]. However, we believe that areas in the educational part of cryptography are still not sufficiently explored, and proposing more practical solutions will enormously help address the lack of

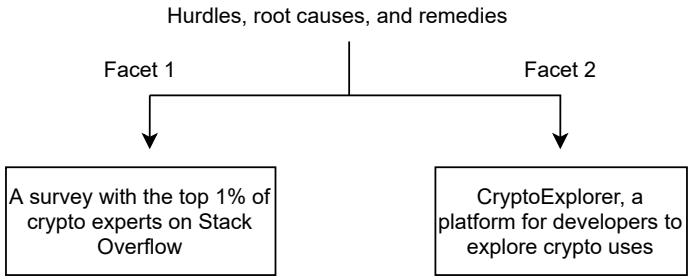

Figure 26: The methodology followed to answer the fourth research question

knowledge of developers in cryptography.

6.1 Study design

The aim of this chapter is to collect and analyze the opinions of crypto experts concerning the current state of problems in cryptography as well as devising an educational tool for easing the learning curve of crypto APIs. To that end, we address the fourth research question of our study, *i.e.*, *What are the feasible remedies to alleviate the issue of misusing crypto APIs?*, from two facets, illustrated in Figure 26.

Facet 1 → We surveyed the top 1% of crypto responders on Stack Overflow to collect their suggestions on how to ease the use of cryptography for inexperienced, mid-range developers. We contacted 247 users whose email addresses were available and received 26 detailed responses from the top crypto responders. These developers mentioned 36 crypto hurdles (*e.g.*, validating certificate chains) and the root causes (*e.g.*, outdated tutorials) indicating why cryptography is difficult for developers. Furthermore, they recommended 29 strategies, *e.g.*, employing misuse-resistant crypto libraries, by which cryptography can be facilitated for developers as well as for crypto designers. The suggestions reported by the participants mainly targeted the human-related, crypto APIs, and learning resources aspects. The current findings can provide an exhaustive list of guidelines for inexperienced developers to become reasonably informed about common challenges and solutions in the cryptography domain.

Facet 2 → We developed a web platform, named CryptoExplorer, stocked with numerous real-world secure and insecure examples that developers can explore to learn how to use cryptographic APIs properly. This platform currently provides 3,263 secure uses and 5,897 insecure uses of Java Cryptography Architecture mined from 2,324 Java projects on GitHub. A preliminary usability study study showed that CryptoExplorer instantly provides developers with secure crypto API use examples, that

developers can save time compared to searching on the internet for such examples, and that they learn to avoid using certain algorithms in APIs by studying misused API examples.

In the following, we explain how we conduct the data collection and analysis phases for each of the aforementioned facets.

6.1.1 Experts' opinions

In this section, we describe how we choose crypto questions on Stack Overflow, the approach to fetch the top 1% of crypto responders, and the structure of our survey. Figure 27 shows the steps taken in our methodology and how they are linked together and what data is passed to the next step.

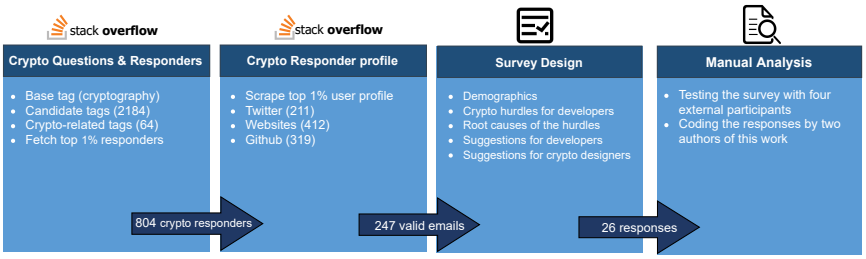

Figure 27: The steps taken in this study to contact top 1% of crypto responders on Stack Overflow

Crypto Questions and Responders

In chapter 5, we assessed the practical experience of crypto experts based on their contribution on Stack Overflow [74]. We recap here what we have done in subsection 5.1.4. We were inspired by the Stack Overflow feature that presents top responders according to a specific tag. We based the *cryptography* tag to extract 11,130 posts, and subsequently analyzed what other tags appeared together with the *cryptography* tag in the posts, providing us with 64 crypto tags resulted from a manual analysis of the tags (See Table 15).[1] Eventually, 804 unique developers constituted the list of top 1% crypto responders of the 64 crypto-related tags.

Crypto Responder Profile

Stack Overflow offers the ability to its users to share their social media addresses (*e.g.*, Twitter, GitHub, and personal websites) on their profile. Nevertheless, the aforementioned information is not accessible on Stack Exchange

[1] `http://crypto-explorer.com/mapping_data/`

Data Explorer. Hence, to find the selected users' GitHub and Twitter profiles, we automatically scraped profiles of the 804 Stack Overflow top crypto responders. Using the BeautifulSoup library in Python, we parsed each user profile automatically. We found 319 users' GitHub profiles, 211 users' Twitter addresses, and 412 personal website addresses. We looked for the personal website/blog of such users on their Twitter and GitHub bio. Finally, we checked the personal websites to find contact details of the users. This approach armed us with valid 247 email addresses of such users.

Survey Design

We conducted an anonymous survey, which has two parts, namely (1) developer background, (2) hurdles and solutions. The questions and the responses of the survey are available online.[2] In the first section, we asked for their (1) education, (2) level of knowledge in cryptography, (3) ways of obtaining knowledge in cryptography, (4) the crypto library(s) they use most and their drawbacks or advantages. Concerning the knowledge level in cryptography, Nadi *et al.* surveyed 48 developers and introduced a four-level classification for developer crypto knowledge [112]. To assist participants, we added extra explanations to each level so that participants can choose the right knowledge level.

In the second section, we asked four open-ended questions: (1) hurdles in cryptography for developers, (2) the root causes of the hurdles, (3) tools or resources to help developers, (4) potential remedies for crypto libraries. In the survey, we asked the users to read the statement of agreement before participating. Moreover, we did not collect any personal information except for the information explicitly collected by the survey instrument.

To publish the survey, we employed Google Forms to build our online survey. A time-consuming survey is commonly not completed on the internet [18]. Therefore, we evaluated the completion time of the survey with four colleagues who had experience in using cryptography. The assessment phase with the four potential participants revealed that the survey takes approximately 8 minutes to be completed, and two questions in the second part lack clarity for the potential participants. Finally, we fixed the problematic questions in order to increase the lucidity of the questions.

Manual Analysis

To tackle the research question, the explanations of respondents to the open-ended questions commonly consisted of fewer than 60 words. However, to minimize human errors, two reviewers separately coded the responses and cross-checked the consistency of their results. To check the consistency between the reviewers, we calculated Cohen's kappa and the agreement for the

[2] `http://185.94.98.132/~crypto/top_crypto/`

two raters was 81%, which indicates almost perfect agreement between the two reviewers. To arrive at a consensus, the reviewers had multiple sessions together to consolidate their lists of coding. Detecting similar or duplicated codes was a major issue causing discrepancies in their coding.

6.1.2 CryptoExplorer

In this section we present the CryptoExplorer web platform that developers can use to explore real-world crypto API uses mined from open-source projects. Developers can search crypto APIs, explore secure and insecure API uses, and compare them.

We explain the workflow supported by the tool as presented in Figure 28, and explain the current use cases.

Workflow

CryptoExplorer automatically grows its database of cryptographic examples by finding cryptographic-related projects and analyzing and storing the cryptographic API examples. The pipeline consists of five major steps to add one cryptographic example. We use the cron scheduling daemon to automatically schedule and execute bash scripts.

Search and filter: To add more cryptographic API usages, we look for current open source projects hosted on GitHub. First, we look for Java projects using GitHub's repository search API. We exclude forked projects to avoid cloning duplicated projects. We then use GitHub's code search API to search for JCA APIs inside Java projects. Finally, we store the addresses of Java projects whose code contains cryptographic APIs.

Clone and compile: We clone and compile projects to perform static analysis. We clone identified Java projects containing JCA APIs. To build the projects, we check for the presence of the Project Object Model (POM) file in the project's path, and if it exists, we proceed with compilation. The POM file is an eXtensible Markup Language (XML) representation of a Maven project. We use the Maven build tool for the compilation process and we skip projects in which dependencies cannot be resolved. Lastly, we download neither the forked version of a project nor a project twice for analysis.

Analyse: We currently employ CogniCrypt, a static-analysis tool tailored to find a wide range of misuses of JCA APIs [92]. The tool returns secure and buggy API usages. We specified a time period, *i.e.*, 15 minutes, for CogniCrypt to run the analysis to limit time and resources used on the server.

In future, we plan to add more analysis tools and present their results in CryptoExplorer.

Parse and inform: We extracted information from the analysis report to present to developers via CryptoExplorer. In particular, for every project, we extracted which cryptographic API was (mis)used, the reason for being

misused, the corresponding file name, and the line number of each detected (mis)use. With the help of the *git blame* command we also identified the last developer who committed the code associated with each API use, as well as the commit time.

Afterwards, we created issues on the GitHub page of projects to report the potential misuses. In case the project owner decided to disable issues for the project, we sent an email to the developers who committed the API misuse. Each issue report includes help instructions related to the type of misuse, line number, and file path.

In order to curate a healthy dataset, we checked responses to each issue and, in case of a false positive, we adapted the entry in CryptoExplorer's dataset. As we needed to manually analyze the responses, we mined 100 projects weekly at the moment. We did not re-examine the repositories automatically once we notified them concerning the crypto misuses.

Store: We stored the analysis results in a database, tracking elements such as the filename, the crypto API name, the API call line number, the user-defined function where JCA APIs were used, and whether the API use is secure or not. CryptoExplorer is designed to support multiple languages: it must be configured with the API host language, build tools, and static analysis tools. If a specific crypto algorithm is found to be vulnerable (*e.g.*, `SHA-256`), it is feasible to mark the specific crypto algorithm's usages as buggy in the database.

Table 23 presents the current numbers of secure and buggy projects and commits, and their totals. There is an average of 1.7 distinct API usages per project with a standard deviation of 1.3 and an average number of 3.9 commits per project with a standard deviation of 7.4.

Table 23: The status of projects and commits

	Secure	**Buggy**	**Total**
Projects	642	1,682	2,324
Commits (LoC)	3,263	5,897	9,160

Usage Scenario

The user interface of CryptoExplorer, shown in Figure 29, is simple to use, and just a few options need to be adjusted to tune the search query. Developers can either directly visit the publicly available website of the CryptoExplorer, or use an Eclipse plugin that we have developed to interact with CryptoExplorer from within the IDE. The simplicity of the plugin only requires developers to select either an entire Java file or part of one and click on the plugin's icon to open CryptoExplorer in a web browser.

CryptoExplorer allows developers to search for example usages of a particular cryptographic API. For instance, a user might input the *MessageDigest*

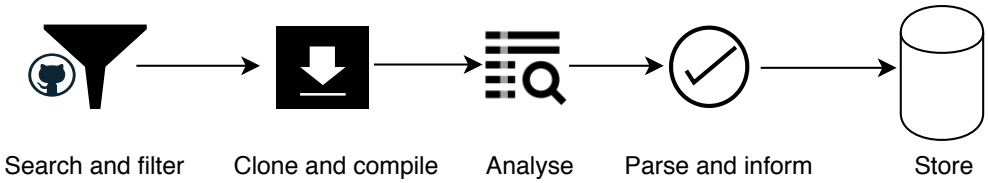

Figure 28: The workflow of CryptoExplorer

API name to see how this API is (mis)used in different examples in order to circumvent the lack of usage examples and security hints in the official Java documentation. Developers may also input a piece of cryptography code that they would like to evaluate, or to use as a query to find similar crypto usages. They can choose to explore only secure uses or buggy uses. In case CryptoExplorer does not find any example, it suggests examples where both secure and buggy uses of the queried APIs exist, distinguishable by green and red colors, respectively.

When developers input sample code to be used to search for usages of cryptographic APIs, CryptoExplorer identifies the cryptographic APIs in the code, and presents developers with code examples that use the same APIs. For better readability, we rank those files higher where crypto API usages are close to each other in a file and are in the same user-defined method. We use standard deviation to compute how close API usages are in a file. As each API could be misused in a few ways, CryptoExplorer does not return hundreds of examples to users with several identical key messages. In case users are interested to study more APIs examples, they can navigate more examples.

For instance, in Figure 29, a user has entered a piece of code that aims to conduct file encryption. CryptoExplorer recognizes the `Cipher` and `Mac` cryptographic APIs in the code (1). Once the user hits the "Search" button, CryptoExplorer returns code examples that simultaneously use the same APIs listed in the search code (2). In this example, CryptoExplorer could not find any secure example where both APIs were used securely, and it suggested an example where the APIs were used in both secure and buggy ways. The misused API is highlighted by a red linear gradient color and the secure API usage is highlighted by a green linear gradient color. In the case of navigating large files, there are two buttons that assist users to jump to the highlighted lines quickly. Under each returned example, users can read related information regarding the example's misuses (3). Finally, users can navigate to more examples (4).

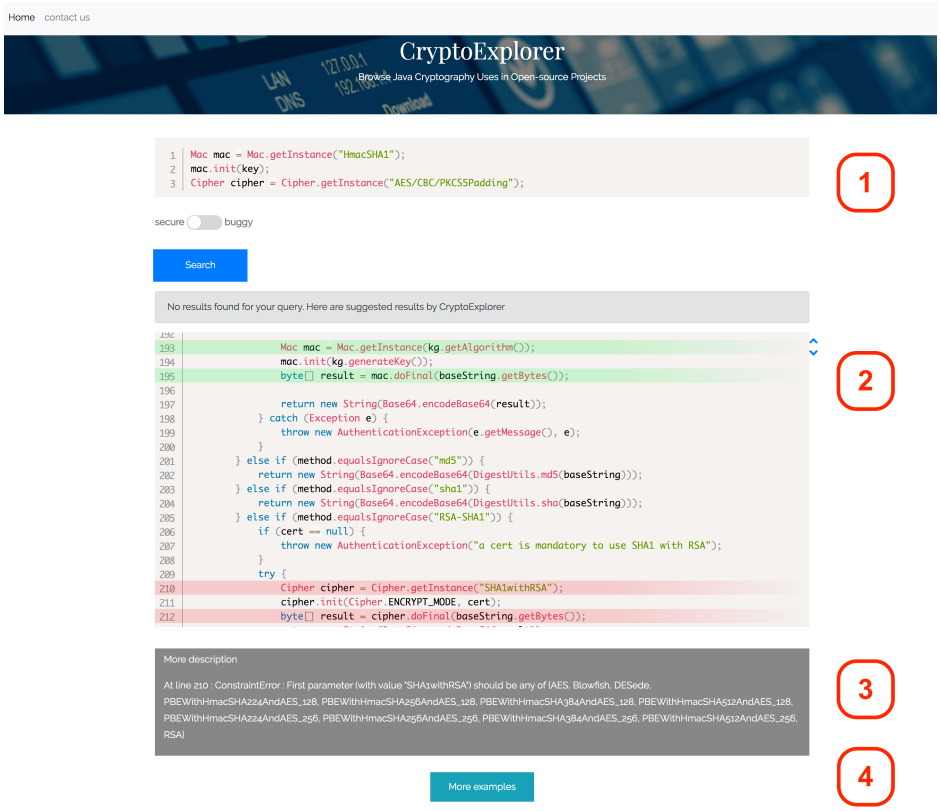

Figure 29: Exploring code examples based on a given code snippet

6.2 Results and discussions

In the following, we present and discuss our results regarding the two facets of this chapter.

6.2.1 Experts' opinions

We received 26 detailed responses from the top 1% of crypto responders on Stack Overflow. We received responses from 10% of the contacted users. We believe that the number of participants is not an essential factor as our accessible target audience is difficult to reach. In particular, we acknowledge the importance of how detailed the responses are and the quality of the responses which we capitalize to draw impactful conclusions.

Nearly half of the developers (42%) reported their educational level as either Master or PhD. With respect to developer knowledge in cryptography, 46% reported they are *very knowledgable*, and the rest responded *knowledgeable*. The participants reported their level of knowledge neither as *somewhat knowledgable* nor *not knowledgable*. Approximately all of the participants (*i.e.*, 23) responded to what crypto libraries they primarily used (See Figure 30).

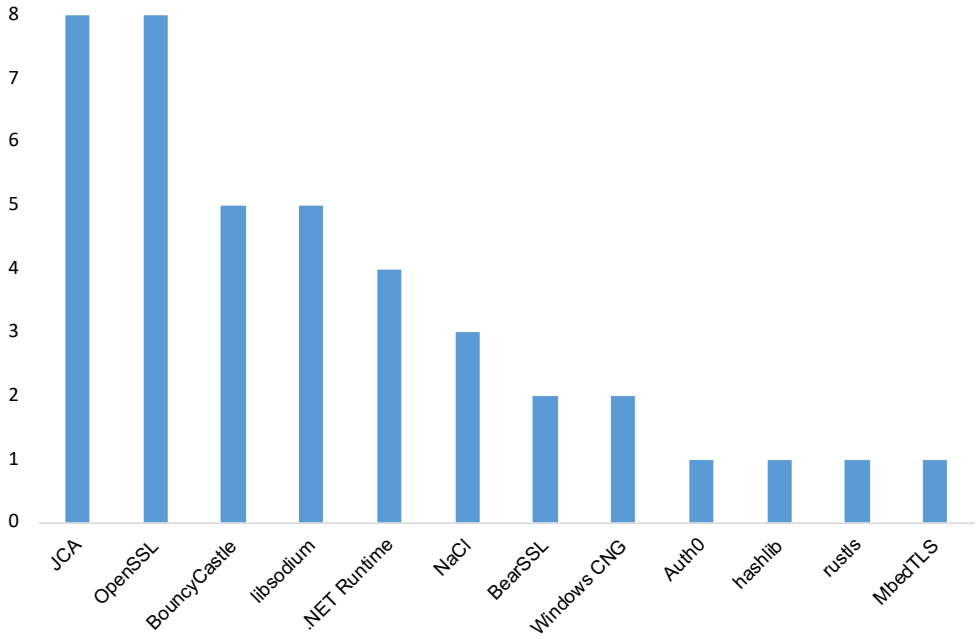

Figure 30: Crypto libraries with which the participants mainly work

JCA and OpenSSL each were selected eight times, and Bouncy Castle alongside libsodium are in second place of participants' choices.

Crypto Hurdles

In this section of the survey, we asked participants based on their experience what areas of cryptography are considered to be highly problematic for inexperienced developers and received 26 responses. We extracted 21 key areas from the participants' responses and described them in the following:

(1) differences between password hashing and encryption, (2) use cases for symmetric/asymmetric encryption, (3) what salt, IV, and nonce are (4) pros and cons of various modes of encryption (ECB or GCM), (5) confusion in picking an algorithm for a scenario, (6) safely signing data or what data should be signed to prevent replay attacks, (7) why not use random number generators, (8) crypto random numbers vs random numbers of other forms, (9) encryption vs integrity, (10) management of private keys (storage, reusing keys, and transmission), (11) what are side-channel attacks,(12) the likelihood of attack vectors, (13) basics of PKI, (14) implementing authentication systems, (15) elliptic curve concept, (16) giving up when working with low-level APIs, (17) certificate authority and X.509, (18) what are PKCS standards, (19) how to share a private key across devices, (20) how to validate certificate chains, (21) hash-based message authentication code (or HMAC).

We also asked them about the advantages and disadvantages of their library of choice and received 15 responses. We aggregated their comments

concerning specific crypto libraries including advantages and drawbacks as follows:

> **JCA:** It is comprehensive but the API is clunky. Requires a good notion of several cryptographic concepts and is not friendly for novice developers. **Bouncy Castle:** Seems to be widely accepted as a safe implementation, problem with finding reliable examples, it has grown too large and documentation is not helpful. **Libsodium:** Set a new bar of modern algorithms combined with ease of use, which hides the complexity from the developer. Hard to use cross platform. **BearSSL:** very low in resource consumption (RAM) and fully constant-time. **.NET runtime:** Not sure if the crypto algorithms are NIST approved. Does not present a comprehensive list of algorithms, but the existing ones are easy to use. **OpenSSL:** No security against accidental misuse and presents opportunities to make mistakes. The library tends to be difficult-to-configure options and APIs. However, it is rich in features and it is open source and enables others to profoundly analyze the library. **Windows CNG:** it is closed source, not always correctly documented.

Discussion and related work: The participants pointed out many technical issues (*i.e.*, 21 items) in cryptography which are regarded as complicated areas for inexperienced developers. In chapter 5, we conducted a large-scale study on crypto-related questions on Stack Overflow and figured out that there are three main themes in the questions [77], which can be mapped to the 21 problematic areas reported by the participants of this study. For instance, they found digital certificates is to be one of the three topics in which developers mainly ask questions about areas such as X.509, validating certificate chains, signing and verifying, generating self-signed certificates, and certificate authority. In the second topic, they found developers struggle with different encryption modes (*e.g.*, CBC and ECB) and key sizes (*e.g.*, 128, 192, and 256-bit), and padding options, which are similar to what crypto experts suggested from experience. Lastly, the third found topic in the developer question implies that developers are uncertain which hashing algorithms (*e.g.*, MD5, SHA-1) can furnish a higher level of reliability, the relevance between password length and the resulting hash, to what degree salt can maximize the security of a hash, and the differences between key stretching algorithms, *e.g.*, PBKDF2, bcrypt, and scrypt.

Developers oftentimes are not informed of the undesirable repercussions and the catastrophic harm of weak random number generators as it makes adversaries able to predict future outputs of a cryptosystem [156]. Notwithstanding the significance of deterministic random bit generators (DRBGs), they may not have received adequate scrutiny, leading to several disastrous mistakes in software systems, *e.g.*, weakening the DRBG's seeding process in Debian DRBG [3], producing weak RSA keys by improper seed generators [20] [80], and enabling the theft of $5 700 bitcoin due to a bug in entropy use in the Android DRBG.[4]

[3] `https://www.hdm.io/tools/debian-openssl/`
[4] `https://tinyurl.com/mr45kcvn`

With respect to security concern, we also reported in subsection 5.1.3 that all the participants in the survey are concerned about the potential attack vectors and security, whereas the participants rarely employ security tools in development. Similarly, recent studies exhibited that user data can be easily susceptible to Man in the Middle attacks provided that developers inadvertently employ trust managers in Android that accept all certificates, trust all hostnames, and ignore SSL errors [117]. This poses security hazards on authentication systems and eliminates the burden of certificate validation.

Authentication is the first line of defense to every software. In microservices, JSON Web Token (JWT) plays an important role in token-based user-to-service authentication where each service relies on tokens in a secure and decentralized manner [155]. Nonetheless, the JWT tokens are prone to some attacks, such as failing to verify the signature, KID parameter injection and allowing the None algorithm.[5] The "Public-Key Cryptography Standards," or "PKCS" standards consist of a number of components, called PKCS #1, #3, and #5-15. PKCS is defined for both binary and ASCII data types, describing the syntax for messages in an abstract manner. For instance, PKCS #8 is a private-key information syntax standard., defining a method to store private key information [148]. However, there exist some security caveats for PKCS #11 and PKCS #1 1.5 [27] [84].

Initialization vectors (IV) have been seen as a problematic part in cryptography for developers on Stack Overflow. Developers should be aware of IV attacks as a security risk of the CBC encryption mode in block ciphers that can be applied also in IPsec [105]. In such attacks, an unauthenticated IV in CBC encryption is used so that the adversary can control the first block of the decrypted plaintext. IV attacks also exposed the security of WEP protocol in wireless networks [139].

Root Causes

The participants also mentioned 15 factors in three major categories that are regarded as the root causes of developer difficulty in cryptography. The influential causes are as follows:

[5] `https://www.netsparker.com/blog/web-security/json-web-token-jwt-attacks-vulnerabilities/` The None algorithm specifies that the token is not signed

Human-related: Lack of knowledge or interest about what happens under the hood, a few people are responsible for operations security in small or mid-range companies, economical/availability hinders hiring experienced developers, excuses for using deprecated algorithms (*e.g.*, MD5), lack of knowledge in mathematics (*e.g.*, finite fields) and statistics, inadequate attention toward security testing in software development, developers have the "functional" mindset, while security can be achieved only with the "hostile" mindset. **Learning resources:** Lack of training in companies, incorrect and outdated tutorials, lack of comprehensive, understandable documentation for developers from any levels, no lesson dedicated to crypto in major bachelor or master programs. **Crypto APIs:** Outdated algorithms in libraries and examples, too many options in APIs, lack of misuse-resistance APIs, the complex sequences of API calls needed to get to the results, accepting weak key lengths.

Discussion and Related Literature: The knowledge of developers varies in terms of security and therefore, companies attempt to hire security consultants to fill the knowledge gap. However, hiring such experts can be expensive and may not be feasible for some new or resource limited businesses. In a survey, out of 97 participants, only 38% reported the existence of a security consultant at their workplace [76]. Another issue is sometimes developers use deprecated algorithms, *e.g.*, MD5, due to various reasons, such as project limitation, irrelevancy to security context, abandonment of the project, and security through obscurity [72].

Developers often struggle to find practical examples on documentation and therefore, they refer to online forums to resolve their crypto issues. Moreover, the level of explanation in documentations may not match the level of developers. Acar *et al.* pointed out that official API documentation commonly lacks security-related hints, crypto libraries should offer a simple, convenient interface and accessible documentation with secure, easy-to-use code examples [1].

As cryptography is a core component of many software systems, having a dedicated course in bachelor or master programs can benefit students in their future careers. However, the cryptography course must be carefully adapted to the target audience since they can be from computer science, mathematics, computer engineering, and information security fields [6]. In a study, a researcher focused on design thinking (DT) activities for teaching cryptography to students [9]. He devised a cryptography curriculum for a semester and the pilot study with students showed a high level of student understanding and improvement in solving complex cryptographic problems. Hamdani *et al.* addressed the issue of creating a stand-alone cryptography course independent of other subjects, such as discrete mathematics, statistics, number theory, and modern algebra[7]. They proposed two detailed curricula for the undergraduate and graduate students. Adamovi *et al.* eased the learning curve of cryptography by introducing a different, interactive approach. open-source tool, called CrypTool [4].

There is corroborative evidence indicating that large numbers of options

while working with crypto APIs coupled with poor documentation oftentimes caused the developers to fail [76] [1]. Writing tests can improve the security of the software and reduce the risk of expensive embarrassments later on. However, writing tests is regarded as an arduous task for developers. Ghafari *et al.* realized that testing is not genuinely embraced by developers and there is a tendency to abandon writing tests when high effort is necessary [58]. Besides that, other factors such as ongoing changes in requirements and the urgent need to add new features can also exacerbate the problem of writing tests.

Ultimately, in order for developers to think like hackers, it is worthy to discover how blackhat hackers proceed with the exploration and exploitation phases [48] so that developers can reasonably determine the most effective risk assessment and management strategies. Most professional security experts acquire prominent security certifications, such as OSCP/OSEP [6], or CISSP [7], and practice other methods, such as thinking outside of the box, continuous study, attending conferences, *e.g.*, Black Hat, OWASP, and SANS [33].

Suggestions

The participants were asked to suggest any resources or tools to help inexperienced developers. They mentioned 15 suggestions in three major categories which are described as follows:

Crypto APIs: Use good libraries (*e.g.*, Tink and libsodium), don't roll your own crypto (*e.g.*, use TLS), use a library that has the best support, use static analysis tools, to build a custom crypto protocol, use the Noise framework, understanding SHA and key based SSH-login then RSA, understanding OpenSSL helps learning about different crypto concepts. **Learning resources:** For beginners "Network Security" chapter of Andrew Tanenbaum's Computer Networks book, https://doc.libsodium.org/ https://crackstation.net/ https://hashcat.net/hashcat/, handbook of Applied Cryptography by Menzenes, security.stackexchange.com and Wikipedia, Cryptography courses on MOOC (*e.g.*, Coursera), OWASP - https://cheatsheetseries.owasp.org/, https://cryptobook.nakov.com/ and security blogs of companies *e.g.*, microsoft IBM, read Singh's The Code Book. **Human-related:** Hiring an expert to review the code.

We also asked the participants what tools and resources they used to obtain their crypto knowledge/experience. We extracted the following means from 21 responses:

[6] `https://www.offensive-security.com/courses-and-certifications/`
[7] `https://www.isc2.org/Certifications`

Books: Cryptography Engineering by Schneier Ferguson and Kohno, Serious Cryptography by J.P. Aumasson, Intro to Modern Cryptography by Katz & Lindell, An Introduction to Mathematical Cryptography by Hoffstein - Piper - & Silverman, Simon Singh's "the code book", Schneier's Secrets and Lies, Sterling's Hacker Crackdown, A Book of Abstract Algebra, Practical Cryptography and Applied Cryptography by Bruce Scheier, Finite Fields for Computer Scientists and Engineers. **Learning resources:** Coursera class by Dan Boneh, Thawte training, training at companies, https://www.schneier.com, crypto security blogs/forums (crypto stackexchange), RFCs, IACR ePrint papers, official documentation, tutorials on youtube (Computerphile), college education, Ph.D. studies. **Practical experience:** Implementations of crypto algorithms in the C# runtime, hands-on experience with Java crypto APIs

We asked the participants what improvements they recommend to increase the usability of crypto libraries. We received 23 responses and extracted the following key points from participants' suggestions:

Crypto APIs: (1) Generate random nonces internally! One less argument to deal with, (2) implement APIs with friendly and fewer arguments, (3) like libsodium, few functions, with good defaults, only one algorithm per use case, clear documentation, stable/similar API across languages and platforms, (4) give developers a way to verify it's done right, (5) the library should handle the salt, (6) Add a middle level library that build totally upon the base crypto library but simplify usage for the most common case uses. A middle level library can implement the most common use cases setting of some of the arguments in a basic but secure way, (7) not provide endlessly tunable functions with which to shoot oneself in the foot, (8) In Java the APIs force the users to consider buffered values. Better integration with java.io would help. In addition, the implementation factory pattern is 15 years out of date in Java, (9) make the APIs as feature complete as possible, (10) documentation explaining how to use one API is useless, it must show how the API can be used in a context, (11) point out to the most common programming mistakes in this area, (12) more friendly fun video explaining the concepts, (13) documentation should embody a detailed list of vulnerabilities known to date, (14) support all the common needs such as cert formats, signing, JWT, etc.

Discussion and Related Literature: It is highly recommended that security through obscurity can bring about negative consequences and avoidance of such methods is the best choice. Moreover, engineers without thorough knowledge in cryptography oftentimes attempt to build crypto protocols, which is a very challenging problem. However, such protocols must be evaluated from various facets such as mutual and optional authentication, identity hiding, forward secrecy, and other advanced features by tools and frameworks *e.g.*, Noise, ProVerif, and AVISPA [121] [23] [10]. Similarly, there exists a number of static analysis tools that are specifically designed for checking crypto APIs, *e.g.*, CHIRON, CogniCrypt [128] [92].

There is a considerable number of security books available on the internet. However, grasping all the necessary content from books can be cumbersome for developers, and therefore they prefer other types of information sources, which can be more engaging and rapidly accessible[76]. For instance, Massive Open Online Courses (MOOCs) are free online courses available for anyone

around the world to study. There are two major MOOCs platforms: (1) Coursera (offered by Stanford University) and (2) edX (offered by MITx) that present a large number of computer science courses. In particular, there are cryptography courses, *e.g.*, "Cryptography 1" [8] and "Cryptography" [9], available on the Coursera platform for interested students to learn [95]. There exists a large number of online cybersecurity courses, including cryptography, on other platforms as well, *e.g.*, Udemy and Khan Academy [60]. [10]

A prime factor in successful information security management is effective compliance of security policies and suitable integration of people, processes, and technologies. Security awareness training of employees is one of the mechanisms in which the effectiveness of integration can be enhanced. Emina *et al.* conducted a security awareness program in a company with 2900 employees and subsequent auditing of how effective and successful the program was [46]. The results had positive outcomes on the impact of human awareness on the success of information security management programs. Siponen *et al.* conducted a field survey to comprehend which factors assist employees' compliance with security policies [135]. The results demonstrated that the visibility of information security policies has a considerable impact on employees' adherence to the policies. Furthermore, employees comply with information security policies if they are fully aware of how vulnerable their organization is to security hazards and the severity of these threats. Kweon *et al.* attempted to discover a relationship between cybersecurity training and the number of incidents of organizations [96]. They realized the role of security training and education has a positive association with reducing the number of incidents in organizations from the quantitative aspect.

As discussed in the hurdles and root causes section, developers lack adequate knowledge to determine what algorithms and additional options render their cryptosystem insecure. To fully eradicate such unintentional mistakes, an effective approach is to expose developers to a few sets of secure options so that they are not inundated with various options in crypto APIs. In the same vein, Kafader *et al.* developed a fluent API, called FluentCrypto, to facilitate the secure and correct use of cryptography in Node.js [88]. Their solution provides a task-based approach and hides the low-level complexities from the developer in which the APIs become misuse-resistant. Google also proposed a library called Tink in which the APIs are secure, ease of use is considered, and the APIs are misuse resistant.[11] Tink is currently available for Java, Python, C++, and Go. Mindermann *et al.* conducted a controlled experiment on two major crypto libraries in Rust with 28 student participants [108]. They figured out that libraries vary in terms of misuse resistance and usability and proposed a list of 15 suggestions including technical and non-

[8] `https://www.coursera.org/learn/crypto`
[9] `https://www.coursera.org/learn/cryptography`
[10] `https://www.khanacademy.org/`
[11] `https://github.com/google/tink`

technical ways to address the issues. For instance, one of the suggestions is to hide low-level APIs in a separate API layer called "hazardous materials" to inform the developers about the risk of working with this type of API.

6.2.2 CryptoExplorer

We investigated the experience of users interacting with CryptoExplorer to understand to which extent it supports developers to properly use crypto APIs.

Experiments

We conducted semi-structured interviews with four participants who used this platform. They willingly chose to participate without being paid and all had an academic background in computer science (*i.e.*, two bachelors, and two Ph.D. students). The participants had at least 2 years of experience in Java programming. They all used `MessageDigest` to produce hashes, and they were familiar with cryptography concepts. However, it was not their daily job to write cryptographic-related code in Java or any other languages.

We presented CryptoExplorer to the participants, and explained its features. Then we asked them to accomplish the following two tasks using CryptoExplorer, while they could consult official JCA documentation as well:

- **Task1.** Tell us of two security concerns that one should consider when using the `MessageDigest` API to generate a hash.
- **Task2.** Find security issues in a given crypto code snippet that uses the `Cipher` API to conduct file encryption, and explain how to resolve them. This task first must be done by using any resources on the internet and then with the help of CryptoExplorer.

We asked participants to think aloud while working on each task. In the end, we interviewed them regarding the difficulties that they experienced.

To accomplish the first task, participants had to search the `MessageDigest` API, and explore 20 similar code examples. Every participant succeeded to complete this task, on average within seven minutes. They stated that they had to read all of the returned examples as the API was used in a different ways (*e.g.*, initialized with different hashing algorithms). A participant suggested adding an option to decide whether to exclude an example due to false positives. Such examples may present different scenarios that are not cryptography-related or tool's mistake. When we asked whether they know why a particular security issue exists, they stated that sometimes this was not directly evident from a misuse itself. They had to read the information below each example, and in a few cases, they stated the information is not sufficiently expressive. For instance, one participant did not know why using

`SHA-1` is not secure. One participant said that providing external links for each misuse could help demystify the reason behind each misuse. A participant explored more examples by clicking on the more examples button, and we realized that he could not completely benefit from examples that had variables whose definitions were missing in the provided Java file. He suggested excluding examples whose information spans more than one Java file and providing a button to report such examples. All in all, all participants figured out what hashing algorithms, *e.g.*, `MD5` or `SHA1`, can be problematic or the importance of calling the `Digest` and `Update` methods in `MessageDigest`.

To accomplish the second task, participants first used the internet for 20 minutes to find out the misuses in the code snippet. Only one participant could realize where the problem lies. Then, they used CryptoExplorer and completed the task on average in eight minutes. They all had to go through six similar examples to learn the correct and wrong way of using the API. They found that mainly because the *algorithm/mode/padding* string in the `getInstance` method of the `Cipher` API accepts several algorithms, mode, and padding modes. Participants stated that the buttons for jumping to affected lines help immensely as sometimes files contain hundreds of lines of codes. They also suggested that a search feature for each example would ease the problem of finding variables. We also discovered that different types of misuses have different levels of difficulty for users to understand. For instance, the incomplete operation error type required developers to carefully read a misuse example, while a constraint error was often clear to participants only after looking at a misuse example.

Finally, we asked participants about their experience with CryptoExplorer, and how much easier it is compared to searching for misuses on the internet. They all agreed that CryptoExplorer facilitates learning how a crypto API should be used correctly by providing real-world analyzed examples, highlighting the lines, and presenting information related to each example. They also pointed out that it is extremely hard to find topics on Stack Overflow or a particular website regarding misuses of a specific cryptographic API and that they cannot trust the provided cryptographic code snippets.

6.3 Threats to validity

In the following, we describe the threats to validity of each of explored facets.

6.3.1 Experts' opinions

We employed an existing work's data in order to conduct our study [74]. They identified 804 developers who were among the top 1% responders in 64 crypto tags on Stack Overflow. The number of crypto-related tags may not represent the entire crypto-related tags on Stack Overflow but certainly covers a large number of crypto questions on Stack Overflow (>100,000 questions).

Moreover, there is a high likelihood that the missing crypto-related tags may have been used together with one of the major crypto-related tags used in this study. Interestingly, the unique number of top crypto responders was not increased by adding more crypto-related tags, which can predict to some degree that adding more crypto tags may not certainly result in having more unique top crypto responders. We were only able to find 319 GitHub profiles and did not perform any exhaustive search on Google to collect more users. We only examined their personal blog or website in order to identify their contact details. We could only contact 247 top crypto responders. Further investigation on the internet may increase the number of identified contact details of such users.

6.3.2 CryptoExplorer

One of the inevitable limitations of CryptoExplorer is its maintainability. At the time of building the tool, we used the rental server to keep the pipeline up and running. Nevertheless, due to financial constraints, the process of adding more examples could have been interrupted. The other issue is the reliability of crypto static analysis tools incorporated into the pipeline. In case of producing false positives, this might add inaccurate examples to the database. Hence, a manual inspection on a subset of examples is necessary.

6.4 Summary and conclusion

We conducted a study on the top 1% of crypto responders on Stack Overflow to shed some light on how cryptography can be eased for developers. We were interested in perceiving the experience of top crypto users in terms of hurdles and improvements in cryptography. While the participants reported various technical subjects that are problematic for developers, the mentioned root causes for the problematic areas can be categorized into three major issues, namely training and learning resources, crypto APIs, and human-related. They pointed out suggestions such as the usage of libraries that are misuse-resistant and the usage of various online sources, *e.g.*, Coursera, where developers can find reliable and updated cryptography tutorials.

We have presented CryptoExplorer, a web platform to search for real-world crypto API (mis)uses in open-source projects. It currently provides hundreds of code examples mined from 2 324 Java projects on GitHub. A preliminary usability study showed that CryptoExplorer helps developers to find secure crypto examples, and learn how to properly use crypto APIs by examining examples of correct uses and misuses. Moreover, as the dataset of CryptoExplorer is growing, it can be useful for researchers to conduct other related studies.

Chapter 7

Conclusions

Given the high significance of the right usage of cryptography, we explored many facets of inadvertent misuse of crypto APIs. There are temporary or limited solutions to this matter. Some studies introduced some educational tools while others proposed some specialized static analysis tools for cryptography. In essence, such academic tools, if available to use, lack long-term support, maintenance, and promotion. Furthermore, they commonly target a limited audience, *e.g.*, Java developers, or technologies, *e.g.*, JCA APIs, and extending such tools can be a daunting task as they commonly lack well-written documentation.

According to our observations, we learned that lack of detailed knowledge and the high complexity of tools are inextricably intertwined. To discern which one outweighs the other, we explored several directions of each factor. To minimize the bias of our personal judgment, we are not able to determine which one is more influential, despite the fact that each factor alone can be very impactful. Speaking of the developer perspective, developers are highly concerned with security; however, they commonly do not adopt the security standards in their development cycle. This issue has been observed both in open-source as well as industrial projects. It might be due to the fact that they have not had any formal, technical, and customized education in this regard. Such technical training, unlike general security training, can brighten and broaden a developer's viewpoint toward security, specifically cryptography. More importantly, cryptography itself contains many topics and cannot be delivered in one security training session. It is troubling that a large percentage of computer science graduates do not pass a dedicated course on cryptography, not to mention secure development. One more possible factor could be the importance of security with regard to project-level and company-level constraints. Even though developers' companies are highly concerned with security, such companies do not utilize security consultants or security developers. In our investigations, we did not notice a remarkable difference between having more experience and being fresh to programming. This could

be interpreted as cryptography has not been significantly noticeable for such developers over years of working with programming languages. However, there is a group of developers in our observations who have a distinct knowledge of cryptography. Contacting developers on GitHub, we understood that there are some contexts where specific crypto had to be used, some misuses were rooted in the used third-party library, or the repository had been intended to be archived. Experts, in this field, assisted a large number of developers on Stack Overflow. The interesting part is that such expert users use the same programming languages and crypto concepts both on Stack Overflow discussions and GitHub, showing they participate in the discussions where they have experience.

As for crypto APIs, complicated and unknown parameters seem to be the black hole for developers. Therefore, they choose a parameter based on either chance or other sources, which are insecure or outdated. There are latent complexities using OpenSSL, crypto APIs from various libraries, more than a crypto library in a project, different hashes and passwords, and the secure way to deal with cryptographic keys with relation to generating and storing such keys.

Future investigations are necessary for various directions to clarify the reasons behind misusing crypto APIs. For instance, future research on how computer science students in different countries perceive security at university might provide corroborative evidence why such insufficient knowledge still prevails. We call for not only producing more entertaining video tutorials that take developers of various levels into account but also promoting free educational resources for those who are not from a computer science background. Furthermore, the analysis of project- and company-level constraints with respect to the level of security could also assist to achieve a better, fairly comprehensive conclusion. It is also interesting to observe how developers from various crypto/security knowledge groups can successfully complete cryptographic tasks in a lab environment, and subsequently, compare their level of security concern, educational background, years of experience, and any other potential impactful factors. We believe that there should be a principle for designing crypto libraries mandating the usage of secure options in order to enable developers to update such libraries' rules from a reliable source to apply the latest security updates. This effectively bridges the gap of inadequate knowledge of developers and inadvertent misuse of crypto APIs.

Bibliography

[1] Yasemin Acar, Michael Backes, Sascha Fahl, Simson Garfinkel, Doowon Kim, Michelle L Mazurek, and Christian Stransky. Comparing the usability of cryptographic APIs. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 154–171. IEEE, 2017.

[2] Yasemin Acar, Michael Backes, Sascha Fahl, Doowon Kim, Michelle L Mazurek, and Christian Stransky. You get where you're looking for: The impact of information sources on code security. In *2016 IEEE Symposium on Security and Privacy (SP)*, pages 289–305. IEEE, 2016.

[3] Yasemin Acar, Christian Stransky, Dominik Wermke, Michelle L Mazurek, and Sascha Fahl. Security developer studies with GitHub users: Exploring a convenience sample. In *Thirteenth Symposium on Usable Privacy and Security ({SOUPS} 2017)*, pages 81–95, 2017.

[4] Saša Adamović, Irina Branović, Dejan Živković, Violeta Tomašević, and Milan Milosavljević. Teaching interactive cryptography: the case for CrypTool. In *IEEE Conference, ICEST*, 2011.

[5] David Adrian, Karthikeyan Bhargavan, Zakir Durumeric, Pierrick Gaudry, Matthew Green, J Alex Halderman, Nadia Heninger, Drew Springall, Emmanuel Thomé, Luke Valenta, et al. Imperfect forward secrecy: How Diffie-Hellman fails in practice. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 5–17, 2015.

[6] Wasim A Al-Hamdani. Missing factors in teaching cryptography algorithms for information security tracks. In *2009 Information Security Curriculum Development Conference*, pages 15–20, 2009.

[7] Wasim A Al-Hamdani and Ivory J Griskell. A proposed curriculum of cryptography courses. In *Proceedings of the 2nd annual conference on Information security curriculum development*, pages 4–11, 2005.

[8] Arash Ale Ebrahim, Mohammadreza Hazhirpasand, Oscar Nierstrasz, and Mohammad Ghafari. FuzzingDriver: the missing dictionary to increase code coverage in fuzzers. In *IEEE 29th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 2022.

[9] Wasim A Alhamdani. Teaching cryptography using design thinking approach. *Journal of Applied Security Research*, 11(1):78–89, 2016.

[10] Alessandro Armando, David Basin, Yohan Boichut, Yannick Chevalier, Luca Compagna, Jorge Cuéllar, P Hankes Drielsma, Pierre-Cyrille Héam, Olga Kouchnarenko, Jacopo Mantovani, et al. The AVISPA tool for the automated validation of internet security protocols and applications. In *International conference on computer aided verification*, pages 281–285. Springer, 2005.

[11] Nimrod Aviram, Sebastian Schinzel, Juraj Somorovsky, Nadia Heninger, Maik Dankel, Jens Steube, Luke Valenta, David Adrian, J Alex Halderman, Viktor Dukhovni, et al. {DROWN}: Breaking {TLS} using sslv2. In *25th {USENIX} Security Symposium ({USENIX} Security 16)*, pages 689–706, 2016.

[12] Nuh Aydin. Enhancing undergraduate mathematics curriculum via coding theory and cryptography. *Primus*, 19(3):296–309, 2009.

[13] Nathaniel Ayewah and William Pugh. A report on a survey and study of static analysis users. In *Proceedings of the 2008 workshop on Defects in large software systems*, pages 1–5. ACM, 2008.

[14] Nathaniel Ayewah, William Pugh, David Hovemeyer, J David Morgenthaler, and John Penix. Using static analysis to find bugs. *IEEE software*, 25(5):22–29, 2008.

[15] Ali Sajedi Badashian, Afsaneh Esteki, Ameneh Gholipour, Abram Hindle, and Eleni Stroulia. Involvement, contribution and influence in GitHub and Stack Overflow. In *CASCON*, volume 14, pages 19–33, 2014.

[16] Biju Bajracharya and David Hua. Importance of integrating cryptography, steganography, and digital watermarking for undergraduate curriculum. *CTE Journal*, 5(2), 2017.

[17] Abdul Ali Bangash, Hareem Sahar, Shaiful Chowdhury, Alexander William Wong, Abram Hindle, and Karim Ali. What do developers know about machine learning: a study of ML discussions on StackOverflow. In *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*, pages 260–264. IEEE, 2019.

[18] Bemad Batinic and Michael Bosnjak. 11 fragebogenuntersuchungen im internet. *Internet für Psychologen*, page 287, 2000.

[19] Mihir Bellare, Anand Desai, Eron Jokipii, and Phillip Rogaway. A concrete security treatment of symmetric encryption. In *Proceedings 38th Annual Symposium on Foundations of Computer Science*, pages 394–403. IEEE, 1997.

[20] Daniel J Bernstein, Yun-An Chang, Chen-Mou Cheng, Li-Ping Chou, Nadia Heninger, Tanja Lange, and Nicko Van Someren. Factoring RSA keys from certified smart cards: Coppersmith in the wild. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 341–360. Springer, 2013.

[21] Benjamin Beurdouche, Karthikeyan Bhargavan, Antoine Delignat-Lavaud, Cédric Fournet, Markulf Kohlweiss, Alfredo Pironti, Pierre-Yves Strub, and Jean Karim Zinzindohoue. A messy state of the union: Taming the composite state machines of TLS. In *2015 IEEE Symposium on Security and Privacy*, pages 535–552. IEEE, 2015.

[22] Karthikeyan Bhargavan and Gaëtan Leurent. On the practical (in-) security of 64-bit block ciphers: Collision attacks on HTTP over TLS and OpenVPN. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 456–467, 2016.

[23] Bruno Blanchet, Ben Smyth, Vincent Cheval, and Marc Sylvestre. ProVerif 2.00: automatic cryptographic protocol verifier, user manual and tutorial. *Version from*, pages 05–16, 2018.

[24] David M Blei, Andrew Y Ng, and Michael I Jordan. Latent dirichlet allocation. *the Journal of machine Learning research*, 3:993–1022, 2003.

[25] Daniel Bleichenbacher. Chosen ciphertext attacks against protocols based on the RSA encryption standard PKCS# 1. In *Annual International Cryptology Conference*, pages 1–12. Springer, 1998.

[26] Mohammad Ubaidullah Bokhari and Qahtan Makki Shallal. A review on symmetric key encryption techniques in cryptography. *International Journal of Computer Applications*, 147(10), 2016.

[27] Matteo Bortolozzo, Matteo Centenaro, Riccardo Focardi, and Graham Steel. Attacking and fixing PKCS# 11 security tokens. In *Proceedings of the 17th ACM conference on Computer and communications security*, pages 260–269, 2010.

[28] Mohamed Bouguessa, Benoît Dumoulin, and Shengrui Wang. Identifying authoritative actors in question-answering forums: the case of yahoo!

answers. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 866–874, 2008.

[29] Alexandre Braga and Ricardo Dahab. Mining cryptography misuse in online forums. In *2016 IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C)*, pages 143–150. IEEE, 2016.

[30] Virginia Braun and Victoria Clarke. Using thematic analysis in psychology. *Qualitative research in psychology*, 3(2):77–101, 2006.

[31] Michele Bugliesi, Stefano Calzavara, Riccardo Focardi, and Wilayat Khan. Automatic and robust client-side protection for cookie-based sessions. In *International Symposium on Engineering Secure Software and Systems*, pages 161–178. Springer, 2014.

[32] Kelsey Cairns, Harry Halpin, and Graham Steel. Security analysis of the W3C web cryptography API. In *International Conference on Research in Security Standardisation*, pages 112–140. Springer, 2016.

[33] Tracey Caldwell. Ethical hackers: putting on the white hat. *Network Security*, 2011(7):10–13, 2011.

[34] Jonathan Chang, Sean Gerrish, Chong Wang, Jordan L Boyd-Graber, and David M Blei. Reading tea leaves: How humans interpret topic models. In *Advances in neural information processing systems*, pages 288–296, 2009.

[35] Alexia Chatzikonstantinou, Christoforos Ntantogian, Georgios Karopoulos, and Christos Xenakis. Evaluation of cryptography usage in Android applications. In *Proceedings of the 9th EAI International Conference on Bio-inspired Information and Communications Technologies (formerly BIONETICS)*, pages 83–90, 2016.

[36] Mengsu Chen, Felix Fischer, Na Meng, Xiaoyin Wang, and Jens Grossklags. How reliable is the crowdsourced knowledge of security implementation? In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, pages 536–547. IEEE, 2019.

[37] Ping Chen, Nick Nikiforakis, Christophe Huygens, and Lieven Desmet. A dangerous mix: Large-scale analysis of mixed-content websites. In *Information Security*, pages 354–363. Springer, 2015.

[38] Kefei Cheng, Meng Gao, and Ruijie Guo. Analysis and research on HTTPS hijacking attacks. In *2010 Second International Conference on Networks Security, Wireless Communications and Trusted Computing*, volume 2, pages 223–226. IEEE, 2010.

[39] Jacob Cohen. A coefficient of agreement for nominal scales. *Educational and psychological measurement*, 20(1):37–46, 1960.

[40] Dan Michael A Cortez, Ariel M Sison, and Ruji P Medina. Cryptographic randomness test of the modified hashing function of SHA256 to address Length Extension Attack. In *Proceedings of the 2020 8th International Conference on Communications and Broadband Networking*, pages 24–28, 2020.

[41] Anupam Datta, Ante Derek, John C Mitchell, and Arnab Roy. Protocol composition logic (PCL). *Electronic Notes in Theoretical Computer Science*, 172:311–358, 2007.

[42] Morris Dworkin. Recommendation for block cipher modes of operation. methods and techniques. Technical report, National Inst of Standards and Technology Gaithersburg MD Computer security Div, 2001.

[43] Manuel Egele, David Brumley, Yanick Fratantonio, and Christopher Kruegel. An empirical study of cryptographic misuse in Android applications. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*, CCS '13, pages 73–84, New York, NY, USA, 2013. ACM.

[44] Manuel Egele, David Brumley, Yanick Fratantonio, and Christopher 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, 2013.

[45] Abeer EW Eldewahi, Tasneem MH Sharfi, Abdelhamid A Mansor, Nashwa AF Mohamed, and Samah MH Alwahbani. SSL/TLS attacks: Analysis and evaluation. In *2015 International Conference on Computing, Control, Networking, Electronics and Embedded Systems Engineering (ICCNEEE)*, pages 203–208. IEEE, 2015.

[46] Mete Eminağaoğlu, Erdem Uçar, and Şaban Eren. The positive outcomes of information security awareness training in companies–a case study. *information security technical report*, 14(4):223–229, 2009.

[47] Levent Erkök and John Matthews. Pragmatic equivalence and safety checking in Cryptol. In *Proceedings of the 3rd workshop on Programming Languages meets Program Verification*, pages 73–82, 2009.

[48] Jose Esteves, Elisabeth Ramalho, and Guillermo De Haro. To improve cybersecurity, think like a hacker. *MIT Sloan Management Review*, 58(3):71, 2017.

[49] Sascha Fahl, Marian Harbach, Thomas Muders, Lars Baumgärtner, Bernd Freisleben, and Matthew 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, 2012.

[50] Sascha Fahl, Marian Harbach, Henning Perl, Markus Koetter, and Matthew Smith. Rethinking SSL development in an appified world. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 49–60, 2013.

[51] Dávid János Fehér and Barnabás Sandor. Effects of the WPA2 KRACK attack in real environment. In *2018 IEEE 16th international symposium on intelligent systems and informatics (SISY)*, pages 000239–000242. IEEE, 2018.

[52] Ian Fette and Alexey Melnikov. The websocket protocol, 2011.

[53] Felix Fischer, Konstantin Böttinger, Huang Xiao, Christian Stransky, Yasemin Acar, Michael Backes, and Sascha Fahl. Stack overflow considered harmful? the impact of copy&paste on Android application security. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 121–136. IEEE, 2017.

[54] Felix Fischer, Huang Xiao, Ching-Yu Kao, Yannick Stachelscheid, Benjamin Johnson, Danial Razar, Paul Fawkesley, Nat Buckley, Konstantin Böttinger, Paul Muntean, et al. Stack overflow considered helpful! deep learning security nudges towards stronger cryptography. In *28th {USENIX} Security Symposium ({USENIX} Security 19)*, pages 339–356, 2019.

[55] S Galbraith. Mathematics of public key cryptography, version 0.9. 2011.

[56] Jun Gao, Pingfan Kong, Li Li, Tegawendé F Bissyandé, and Jacques Klein. Negative results on mining crypto-API usage rules in Android apps. In *Proceedings of the 16th International Conference on Mining Software Repositories*, pages 388–398. IEEE Press, 2019.

[57] Bill Gardner and Valerie Thomas. *Building an information security awareness program: Defending against social engineering and technical threats*. Elsevier, 2014.

[58] Mohammad Ghafari, Markus Eggiman, and Oscar Nierstrasz. Testability first! In *2019 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pages 1–6. IEEE, 2019.

[59] James Giles and Bruce Hajek. An information-theoretic and game-theoretic study of timing channels. *IEEE Transactions on information Theory*, 48(9):2455–2477, 2002.

[60] Lorena González-Manzano and Jose M de Fuentes. Design recommendations for online cybersecurity courses. *Computers & Security*, 80:238–256, 2019.

[61] Peter Leo Gorski, Luigi Lo Iacono, Dominik Wermke, Christian Stransky, Sebastian Möller, Yasemin Acar, and Sascha Fahl. Developers deserve security warnings, too: On the effect of integrated security advice on cryptographic {API} misuse. In *Fourteenth Symposium on Usable Privacy and Security ({SOUPS} 2018)*, pages 265–281, 2018.

[62] Matthew Green and Matthew Smith. Developers are not the enemy!: The need for usable security APIs. *IEEE Security & Privacy*, 14(5):40–46, 2016.

[63] Gustavo Grieco, Will Song, Artur Cygan, Josselin Feist, and Alex Groce. Echidna: effective, usable, and fast fuzzing for smart contracts. In *Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis*, pages 557–560, 2020.

[64] Thomas L Griffiths and Mark Steyvers. Finding scientific topics. *Proceedings of the National academy of Sciences*, 101(suppl 1):5228–5235, 2004.

[65] Zvi Gutterman and Dahlia Malkhi. Hold your sessions: An attack on Java session-id generation. In *Cryptographers' Track at the RSA Conference*, pages 44–57. Springer, 2005.

[66] Phillip Hallam-Baker, Rob Stradling, and B Laurie. DNS certification authority authorization (CAA) resource record. *Internet Engineering Task Force*, 6844, 2013.

[67] Mohammadreza Hazhirpasand, Arash Ale Ebrahim, and Oscar Nierstrasz. Stopping DNS rebinding attacks in the browser. In *ICISSP*, pages 596–603, 2021.

[68] Mohammadreza Hazhirpasand and Mohammad Ghafari. One leak is enough to expose them all. In *International Symposium on Engineering Secure Software and Systems*, pages 61–76. Springer, 2018.

[69] Mohammadreza Hazhirpasand and Mohammad Ghafari. Cryptography vulnerabilities on HackerOne. In *IEEE International Conference on Software Quality, Reliability and Security*. IEEE, 2021.

[70] Mohammadreza Hazhirpasand, Mohammad Ghafari, Stefan Krüger, Eric Bodden, and Oscar Nierstrasz. The impact of developer experience in using Java cryptography. In *2019 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pages 1–6. IEEE, 2019.

[71] Mohammadreza Hazhirpasand, Mohammad Ghafari, and Oscar Nierstrasz. CryptoExplorer: An interactive web platform supporting secure use of cryptography APIs. In *2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 632–636. IEEE, 2020.

[72] Mohammadreza Hazhirpasand, Mohammad Ghafari, and Oscar Nierstrasz. Java cryptography uses in the wild. In *Proceedings of the 14th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pages 1–6, 2020.

[73] Mohammadreza Hazhirpasand, Mohammad Ghafari, and Oscar Nierstrasz. Tricking Johnny into granting web permissions. In *Proceedings of the Evaluation and Assessment in Software Engineering*, pages 276–281, 2020.

[74] Mohammadreza Hazhirpasand, Oscar Nierstrasz, and Mohammad Ghafari. Crypto experts advise what they adopt. In *Proceedings of the 36th IEEE/ACM International Conference on Automated Software Engineering Workshops*, 2021.

[75] Mohammadreza Hazhirpasand, Oscar Nierstrasz, and Mohammad Ghafari. Dazed and confused: What's wrong with crypto libraries? In *18th Annual Conference on Privacy, Security and Trust (PST)*. IEEE, 2021.

[76] Mohammadreza Hazhirpasand, Oscar Nierstrasz, and Mohammad Ghafari. Worrisome patterns in developers: A survey in cryptography. In *Proceedings of the 36th IEEE/ACM International Conference on Automated Software Engineering Workshops*, 2021.

[77] Mohammadreza Hazhirpasand, Oscar Nierstrasz, Mohammadhossein Shabani, and Mohammad Ghafari. Hurdles for developers in cryptography. In *37th International Conference on Software Maintenance and Evolution (ICSME)*, 2021.

[78] Mohammadreza Hazhirpasand, Oscar Nierstrasz, Mohammadhossein Shabani, and Mohammad Ghafari. Crypto heroes: Views and recommendations. In *Proceedings of the 37th ACM/SIGAPP Symposium on Applied Computing*, 2022.

[79] Juhani Heikka. A constructive approach to information systems security training: An action research experience. *AMCIS 2008 Proceedings*, page 319, 2008.

[80] Nadia Heninger, Zakir Durumeric, Eric Wustrow, and J Alex Halderman. Mining your Ps and Qs: Detection of widespread weak keys in network devices. In *21st {USENIX} Security Symposium ({USENIX} Security 12)*, pages 205–220, 2012.

[81] Jeff Hodges, Collin Jackson, and Adam Barth. HTTP strict transport security (hsts). *URL: http://tools. ietf. org/html/draft-ietf-websec-strict-transport-sec-04*, 2012.

[82] Md Shohrab Hossain, Arnob Paul, Md Hasanul Islam, and Mohammed Atiquzzaman. Survey of the protection mechanisms to the SSL-based session hijacking attacks. *Netw. Protoc. Algorithms*, 10(1):83–108, 2018.

[83] Marieke Huisman, Pratik Worah, and Kim Sunesen. A temporal logic characterisation of observational determinism. In *19th IEEE Computer Security Foundations Workshop (CSFW'06)*, pages 13–pp. IEEE, 2006.

[84] Tibor Jager, Saqib A Kakvi, and Alexander May. On the security of the PKCS# 1 v1. 5 signature scheme. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 1195–1208, 2018.

[85] Sifat E Jahan, Mehjabin Rahman, Anindya Iqbal, and Tishna Sabrina. An exploratory analysis of security on data transmission on relevant software engineering discussion sites. In *2017 4th International Conference on Networking, Systems and Security (NSysS)*, pages 1–9. IEEE, 2017.

[86] Brittany Johnson, Yoonki Song, Emerson Murphy-Hill, and Robert Bowdidge. Why don't software developers use static analysis tools to find bugs? In *Proceedings of the 2013 International Conference on Software Engineering*, pages 672–681. IEEE Press, 2013.

[87] Antoine Joux. Authentication failures in nist version of gcm. *NIST Comment*, page 3, 2006.

[88] Simon Kafader and Mohammad Ghafari. Fluentcrypto: Cryptography in easy mode. *arXiv preprint arXiv:2108.07211*, 2021.

[89] Jonathan Katz and Yehuda Lindell. *Introduction to modern cryptography*. CRC press, 2020.

[90] Jinhan Kim, Sanghoon Lee, Seung-won Hwang, and Sunghun Kim. Towards an intelligent code search engine. In *Twenty-Fourth AAAI Conference on Artificial Intelligence*, 2010.

[91] Erik Kouters, Bogdan Vasilescu, Alexander Serebrenik, and Mark GJ Van Den Brand. Who's who in gnome: Using LSA to merge software repository identities. In *2012 28th IEEE International Conference on Software Maintenance (ICSM)*, pages 592–595. IEEE, 2012.

[92] Stefan Krüger, Sarah Nadi, Michael Reif, Karim Ali, Mira Mezini, Eric Bodden, Florian Göpfert, Felix Günther, Christian Weinert, Daniel Demmler, et al. Cognicrypt: Supporting developers in using cryptography. In *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 931–936. IEEE, 2017.

[93] Stefan Krüger, Johannes Späth, Karim Ali, Eric Bodden, and Mira Mezini. Crysl: An extensible approach to validating the correct usage of cryptographic APIs. In *32nd European Conference on Object-Oriented Programming, ECOOP 2018, July 16-21, 2018, Amsterdam, The Netherlands*, pages 10:1–10:27, 2018.

[94] Stefan Krüger, Johannes Späth, Karim Ali, Eric Bodden, and Mira Mezini. Crysl: An extensible approach to validating the correct usage of cryptographic APIs. *IEEE Transactions on Software Engineering*, 2019.

[95] Alptekin Küpçü. White paper on self study cryptography course.

[96] Eunkyung Kweon, Hansol Lee, Sangmi Chai, and Kyeongwon Yoo. The utility of information security training and education on cybersecurity incidents: an empirical evidence. *Information Systems Frontiers*, 23(2):361–373, 2021.

[97] David Lacey. Understanding and transforming organizational security culture. *Information Management & Computer Security*, 2010.

[98] David Lazar, Haogang Chen, Xi Wang, and Nickolai Zeldovich. Why does cryptographic software fail? a case study and open problems. In *Proceedings of 5th Asia-Pacific Workshop on Systems*, pages 1–7, 2014.

[99] Sarah Lewis. Qualitative inquiry and research design: Choosing among five approaches. *Health promotion practice*, 16(4):473–475, 2015.

[100] Yong Li, Yuanyuan Zhang, Juanru Li, and Dawu Gu. iCryptoTracer: Dynamic analysis on misuse of cryptography functions in iOS applications. In Man Ho Au, Barbara Carminati, and C.-C. Jay Kuo, editors, *Network and System Security*, pages 349–362, Cham, 2014. Springer International Publishing.

[101] Helger Lipmaa, Phillip Rogaway, and David Wagner. Comments to nist concerning aes modes of operations: Ctr-mode encryption. In *National Institute of Standards and Technologies*. Citeseer, 2000.

[102] Yang Liu, Minghui Qiu, Swapna GOTTIPATI, Feida Zhu, Jing Jiang, Huiping Sun, and Zhong Chen. Cqarank: Jointly model topics and expertise in community question answering.(2013). research collection school of information systems.

[103] Gang Luo. A review of automatic selection methods for machine learning algorithms and hyper-parameter values. *Network Modeling Analysis in Health Informatics and Bioinformatics*, 5(1):1–16, 2016.

[104] Philipp Mayring. *Qualitative Inhaltsanalyse : Grundlagen und Techniken*. Beltz, Weinheim, 12. edition, 2015.

[105] Christopher B McCubbin, Ali Aydin Selçuk, and Deepinder Sidhu. Initialization vector attacks on the IPsec protocol suite. In *Proceedings IEEE 9th International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WET ICE 2000)*, pages 171–175. IEEE, 2000.

[106] Alfred J Menezes, Paul C Van Oorschot, and Scott A Vanstone. *Handbook of applied cryptography*. CRC press, 2018.

[107] Na Meng, Stefan Nagy, Danfeng Yao, Wenjie Zhuang, and Gustavo Arango Argoty. Secure coding practices in Java: Challenges and vulnerabilities. In *Proceedings of the 40th International Conference on Software Engineering*, pages 372–383, 2018.

[108] Kai Mindermann, Philipp Keck, and Stefan Wagner. How usable are Rust cryptography APIs? In *2018 IEEE International Conference on Software Quality, Reliability and Security (QRS)*, pages 143–154. IEEE, 2018.

[109] Bodo Möller, Thai Duong, and Krzysztof Kotowicz. This POODLE bites: exploiting the SSL 3.0 fallback. *Security Advisory*, 21:34–58, 2014.

[110] Ildar Muslukhov, Yazan Boshmaf, and Konstantin Beznosov. Source attribution of cryptographic API misuse in Android applications. In *Proceedings of the 2018 on Asia Conference on Computer and Communications Security*, pages 133–146, 2018.

[111] Muhammad Nadeem, Edward B Allen, and Byron J Williams. Computer security training recommender for developers. In *RecSys Posters*, 2014.

[112] Sarah Nadi, Stefan Krüger, Mira Mezini, and Eric Bodden. Jumping through hoops: Why do Java developers struggle with cryptography APIs? In *Proceedings of the 38th International Conference on Software Engineering*, pages 935–946, 2016.

[113] Eric Neidhardt. Asymmetric cryptography for mobile devices. *Service-centric Networking*, pages 1–12, 2011.

[114] Nick Nikiforakis, Yves Younan, and Wouter Joosen. Hproxy: Client-side detection of SSL stripping attacks. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 200–218. Springer, 2010.

[115] Daniela Seabra Oliveira, Tian Lin, Muhammad Sajidur Rahman, Rad Akefirad, Donovan Ellis, Eliany Perez, Rahul Bobhate, Lois A DeLong, Justin Cappos, and Yuriy Brun. API blindspots: Why experienced developers write vulnerable code. In *Fourteenth Symposium on Usable Privacy and Security SOUPS 2018)*, pages 315–328, 2018.

[116] Marten Oltrogge, Yasemin Acar, Sergej Dechand, Matthew Smith, and Sascha Fahl. To pin or not to pin—helping app developers bullet proof their TLS connections. In *24th {USENIX} Security Symposium ({USENIX} Security 15)*, pages 239–254, 2015.

[117] Marten Oltrogge, Nicolas Huaman, Sabrina Amft, Yasemin Acar, Michael Backes, and Sascha Fahl. Why Eve and Mallory still love android: Revisiting TLS (in) security in android applications. In *30th {USENIX} Security Symposium ({USENIX} Security 21)*, 2021.

[118] Haidar Osman, Mohammad Ghafari, and Oscar Nierstrasz. Hyperparameter optimization to improve bug prediction accuracy. In *2017 IEEE Workshop on Machine Learning Techniques for Software Quality Evaluation (MaLTeSQuE)*, pages 33–38. IEEE, 2017.

[119] Priyadarshini Patil, Prashant Narayankar, DG Narayan, and S Md Meena. A comprehensive evaluation of cryptographic algorithms: DES, 3DES, AES, RSA and Blowfish. *Procedia Computer Science*, 78:617–624, 2016.

[120] Nikhil Patnaik, Joseph Hallett, and Awais Rashid. Usability smells: An analysis of developers' struggle with crypto libraries. In *Fifteenth Symposium on Usable Privacy and Security ({SOUPS} 2019)*, pages 245–257, 2019.

[121] Trevor Perrin. The noise protocol framework. *PowerPoint Presentation*, 2018.

[122] Luca Piccolboni, Giuseppe Di Guglielmo, Luca P Carloni, and Simha Sethumadhavan. Crylogger: Detecting crypto misuses dynamically. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 1972–1989. IEEE, 2021.

[123] Andreas Poller, Laura Kocksch, Sven Türpe, Felix Anand Epp, and Katharina Kinder-Kurlanda. Can security become a routine? A study of organizational change in an agile software development group. In *Proceedings of the 2017 ACM conference on computer supported cooperative work and social computing*, pages 2489–2503, 2017.

[124] Angelo Prado, Neal Harris, and Yoel Gluck. SSL, gone in 30 seconds. *Breach attack*, 2013.

[125] Petri Puhakainen and Mikko Siponen. Improving employees' compliance through information systems security training: an action research study. *MIS quarterly*, pages 757–778, 2010.

[126] Chaiyong Ragkhitwetsagul, Jens Krinke, Matheus Paixao, Giuseppe Bianco, and Rocco Oliveto. Toxic code snippets on Stack Overflow. *IEEE Transactions on Software Engineering*, 2019.

[127] Sazzadur Rahaman, Ya Xiao, Ke Tian, Fahad Shaon, Murat Kantarcioglu, and Danfeng Yao. CHIRON: deployment-quality detection of Java cryptographic vulnerabilities. *CoRR*, abs/1806.06881, 2018.

[128] Sazzadur Rahaman, Ya Xiao, Ke Tian, Fahad Shaon, Murat Kantarcioglu, and Danfeng Yao. Chiron: Deployment-quality detection of Java cryptographic vulnerabilities. *arXiv preprint arXiv:1806.06881*, 2018.

[129] Martin P Robillard. What makes APIs hard to learn? answers from developers. *IEEE software*, 26(6):27–34, 2009.

[130] Martin P Robillard and Robert DeLine. A field study of API learning obstacles. *Empirical Software Engineering*, 16(6):703–732, 2011.

[131] Christoffer Rosen and Emad Shihab. What are mobile developers asking about? a large scale study using Stack Overflow. *Empirical Software Engineering*, 21(3):1192–1223, 2016.

[132] Pratik Guha Sarkar and Shawn Fitzgerald. Attacks on SSL a comprehensive study of beast, crime, time, breach, lucky 13 & RC4 biases. *Internet: https://www. isecpartners. com/media/106031/ssl_attacks_survey. pdf [June, 2014]*, 2013.

[133] Bruce Schneier. Cryptographic design vulnerabilities. *Computer*, 31(9):29–33, 1998.

[134] Shao Shuai, Dong Guowei, Guo Tao, Yang Tianchang, and Shi Chenjie. Modelling analysis and auto-detection of cryptographic misuse in Android applications. In *2014 IEEE 12th International Conference on Dependable, Autonomic and Secure Computing*, pages 75–80. IEEE, 2014.

[135] Mikko Siponen, M Adam Mahmood, and Seppo Pahnila. Technical opinion are employees putting your company at risk by not following information security policies? *Communications of the ACM*, 52(12):145–147, 2009.

[136] Geoffrey Smith and Dennis Volpano. Secure information flow in a multi-threaded imperative language. In *Proceedings of the 25th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 355–364, 1998.

[137] Ian Sommerville. *Software Engineering*. Pearson, 9th edition, 2011.

[138] Juraj Somorovsky. Systematic fuzzing and testing of TLS libraries. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 1492–1504, 2016.

[139] Adam Stubblefield, John Ioannidis, Aviel D Rubin, et al. Using the Fluhrer, Mantin, and Shamir attack to break WEP. In *NDSS*, 2002.

[140] Tyler W Thomas, Madiha Tabassum, Bill Chu, and Heather Lipford. Security during application development: An application security expert perspective. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, pages 1–12, 2018.

[141] Yijun Tian, Waii Ng, Jialiang Cao, and Suzanne McIntosh. Geek talents: Who are the top experts on GitHub and Stack Overflow? *CMC-COMPUTERS MATERIALS & CONTINUA*, 61(2):465–479, 2019.

[142] Sri Lakshmi Vadlamani and Olga Baysal. Studying software developer expertise and contributions in Stack Overflow and GitHub. In *2020 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 312–323. IEEE, 2020.

[143] J Andrew Valentine. Enhancing the employee security awareness model. *Computer Fraud & Security*, 2006(6):17–19, 2006.

[144] Dirk Van Der Linden, Emma Williams, Joseph Hallett, and Awais Rashid. The impact of surface features on choice of (in) secure answers by stackoverflow readers. *IEEE Transactions on Software Engineering*, (01):1–1, 2020.

[145] Mathy Vanhoef and Frank Piessens. Release the kraken: new kracks in the 802.11 standard. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 299–314, 2018.

[146] Bogdan Vasilescu, Vladimir Filkov, and Alexander Serebrenik. Stackoverflow and GitHub: Associations between software development and crowdsourced knowledge. In *2013 International Conference on Social Computing*, pages 188–195. IEEE, 2013.

[147] Carmine Vassallo, Sebastiano Panichella, Fabio Palomba, Sebastian Proksch, Harald C Gall, and Andy Zaidman. How developers engage with static analysis tools in different contexts. *Empirical Software Engineering*, 25(2):1419–1457, 2020.

[148] Yongge Wang. Public key cryptography standards: Pkcs. *arXiv preprint arXiv:1207.5446*, 2012.

[149] Anna-Katharina Wickert, Lars Baumgärtner, Florian Breitfelder, and Mira Mezini. Python crypto misuses in the wild. In *Proceedings of the 15th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pages 1–6, 2021.

[150] Jim Witschey, Olga Zielinska, Allaire Welk, Emerson Murphy-Hill, Chris Mayhorn, and Thomas Zimmermann. Quantifying developers' adoption of security tools. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, pages 260–271. ACM, 2015.

[151] Shundan Xiao, Jim Witschey, and Emerson Murphy-Hill. Social influences on secure development tool adoption: why security tools spread. In *Proceedings of the 17th ACM conference on Computer supported cooperative work & social computing*, pages 1095–1106. ACM, 2014.

[152] Jiafei Yan, Hailong Sun, Xu Wang, Xudong Liu, and Xiaotao Song. Profiling developer expertise across software communities with heterogeneous information network analysis. In *Proceedings of the Tenth Asia-Pacific Symposium on Internetware*, pages 1–9, 2018.

[153] Xin-Li Yang, David Lo, Xin Xia, Zhi-Yuan Wan, and Jian-Ling Sun. What security questions do developers ask? a large-scale study of Stack Overflow posts. *Journal of Computer Science and Technology*, 31(5):910–924, 2016.

[154] Yuval Yarom, Daniel Genkin, and Nadia Heninger. Cachebleed: a timing attack on OpenSSL constant-time RSA. *Journal of Cryptographic Engineering*, 7(2):99–112, 2017.

[155] Tetiana Yarygina and Anya Helene Bagge. Overcoming security challenges in microservice architectures. In *2018 IEEE Symposium on Service-Oriented System Engineering (SOSE)*, pages 11–20. IEEE, 2018.

[156] Katherine Q Ye, Matthew Green, Naphat Sanguansin, Lennart Beringer, Adam Petcher, and Andrew W Appel. Verified correctness and security of mbedtls hmac-drbg. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 2007–2020, 2017.

[157] Yanfang Ye, Shifu Hou, Lingwei Chen, Xin Li, Liang Zhao, Shouhuai Xu, Jiabin Wang, and Qi Xiong. Icsd: An automatic system for insecure code snippet detection in stack overflow over heterogeneous information network. In *Proceedings of the 34th Annual Computer Security Applications Conference*, pages 542–552, 2018.

[158] Jerrold H Zar. Spearman rank correlation. *Encyclopedia of Biostatistics*, 7, 2005.

[159] Xunhui Zhang, Tao Wang, Gang Yin, Cheng Yang, Yue Yu, and Huaimin Wang. Devrec: a developer recommendation system for open source repositories. In *International Conference on Software Reuse*, pages 3–11. Springer, 2017.

[160] Mingyi Zhao, Jens Grossklags, and Peng Liu. An empirical study of web vulnerability discovery ecosystems. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 1105–1117, 2015.

[161] Weizhong Zhao, James J Chen, Roger Perkins, Zhichao Liu, Weigong Ge, Yijun Ding, and Wen Zou. A heuristic approach to determine an appropriate number of topics in topic modeling. In *BMC bioinformatics*, volume 16, page S8. Springer, 2015.

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Declaration of originality

The signed declaration of originality is a component of every semester paper, Bachelor's thesis, Master's thesis and any other degree paper undertaken during the course of studies, including the respective electronic versions.

Lecturers may also require a declaration of originality for other written papers compiled for their courses.

I hereby confirm that I am the sole author of the written work here enclosed and that I have compiled it in my own words. Parts excepted are corrections of form and content by the supervisor.

Title of work (in block letters):

Authored by (in block letters):
For papers written by groups the names of all authors are required.

Name(s):	**First name(s):**

With my signature I confirm that

- I have committed none of the forms of plagiarism described in the 'Citation etiquette' information sheet.
- I have documented all methods, data and processes truthfully.
- I have not manipulated any data.
- I have mentioned all persons who were significant facilitators of the work.

I am aware that the work may be screened electronically for plagiarism.

Place, date	**Signature(s)**

For papers written by groups the names of all authors are required. Their signatures collectively guarantee the entire content of the written paper.