"Jumping Through Hoops": Why do Java Developers Struggle With Cryptography APIs?

Sarah Nadi[†] Stefan Krüger[†] Mira Mezini[†§] Eric Bodden[‡]

Technische Universität Darmstadt[†] Universität Paderborn & Fraunhofer IEM[‡] Lancaster University[§]

{nadi, mezini}@cs.tu-darmstadt.de, stefan.krueger@cased.de, eric.bodden@uni-paderborn.de

ABSTRACT

To protect sensitive data processed by current applications, developers, whether security experts or not, have to rely on cryptography. While cryptography algorithms have become increasingly advanced, many data breaches occur because developers do not correctly use the corresponding APIs. To guide future research into practical solutions to this problem, we perform an empirical investigation into the obstacles developers face while using the Java cryptography APIs, the tasks they use the APIs for, and the kind of (tool) support they desire. We triangulate data from four separate studies that include the analysis of 100 StackOverflow posts, 100 GitHub repositories, and survey input from 48 developers. We find that while developers find it difficult to use certain cryptographic algorithms correctly, they feel surprisingly confident in selecting the right cryptography concepts (e.g., encryption vs. signatures). We also find that the APIs are generally perceived to be too low-level and that developers prefer more task-based solutions.

Categories and Subject Descriptors

D.2.2 [**Design Tools and Techniques**]: Software libraries

Keywords

Cryptography, API misuse, empirical software engineering

1. INTRODUCTION

As software applications collect more and more sensitive data, it is becoming increasingly essential for developers to rely on cryptography to protect this data. However, this is easier said than done. Application developers are not necessarily cryptography experts and the offered cryptography Application Programming Interfaces (APIs) are often rather complex and not easy to use.

For instance, consider the Java Cryptography Architecture (JCA) [4], the official framework for working with cryptography in Java. The JCA is designed to allow Java application developers to easily use cryptography by separating the APIs developers use from the underlying implementations that can be supplied by any *provider* (e.g., Java's default implementation or BouncyCastle [2]). However, JCA APIs offer a broad variety of different algorithms that in turn support many modes and configuration options. While any JCA provider must support a certain list of cryptography algorithms, it may also support additional algorithms, or even provide different default values for the same JCA API call. As a result, the task to use and compose these API components may be challenging.

In fact, the misuse of cryptography APIs has already been established as a common cause of many security vulnerabilities [14–16]. Solutions that effectively support application developers in correctly and easily incorporating cryptography into their applications are thus urgently needed. To provide such solutions, it is essential to understand the root causes behind the phenomenon and the kind of support developers would perceive as useful. Eliciting these causes and requirements in the specific context of Java-based software development is the goal of this paper. Specifically, this paper addresses the following research questions.

RQ1 *What obstacles, if any, do developers face during the use of Java cryptography APIs?* While existing studies established, after the fact, that APIs are misused [14–16], they give no insights into the underlying reasons. If developers do face obstacles while using the APIs, understanding the nature of these obstacles is essential for deriving guidance towards useful solutions.

RQ2 *What are the common cryptography tasks developers perform?* Robillard [25] concluded that a main obstacle to learning an API is the lack of information on how to use this API to accomplish a specific task. To the best of our knowledge, there is no data that identifies common cryptography tasks developers perform, which would help in providing better support for them.

RQ3 *What tools/methods would help developers use cryptography more effectively?* To aid in developing solutions, we elicit developers' expectations and requirements.

To answer these research questions, we conduct an empirical investigation consisting of four separate studies. *Study 1* (S1) analyzes the top 100 Java cryptography questions asked on the popular question/answer site StackOverflow. *Study 2* (S2) is a pilot survey that gathers data from 11 developers who asked Java cryptography-related questions on StackOverflow. In *Study 3* (S3), we analyze 100 randomly selected public GitHub repositories that use Java's cryptog-

ICSE '16, May 14-22, 2016, Austin, TX, USA

DOI: http://dx.doi.org/10.1145/2884781.2884790

raphy APIs to identify the tasks developers usually need to accomplish. Finally, *Study 4* (S4) surveys 37 developers who use Java's cryptography APIs to confirm some of our findings from the previous three studies. The material used in all four studies is available on our online artifacts page [1].

We answer RQ1-3 by triangulating data from S1-S4. We find that developers do indeed have difficulties in using the Java cryptography APIs and that their main obstacle is the API complexity and lack of a proper level of abstraction. Our survey participants suggest several solutions and we recommend a list of tool features for guiding future tool builders in addressing the obstacles faced by developers.

2. S1: STACKOVERFLOW POSTS

We analyzed Java cryptography-related posts on StackOverflow (SO) to check if developers face problems using the APIs and to understand their obstacles (RQ1).

2.1 Study Design

To query SO, we used the online StackExchange Data Explorer [9]. The query results were then downloaded as a `CSV` file for further analysis. To search for relevant posts, we used the tags `java` and `cryptography` and explicitly excluded `javascript`. On June 25 2015, such a query returned 1,232 posts. To be able to manually analyze each of these posts, we selected only the top 100. To identify the top questions, we sorted posts by *view count* followed by *score* followed by *favorite count*. When faced with a problem, an application developer would search the web or SO and click on the relevant question link(s) from the search results to view it. We believe that a high view count suggests that more application developers face the same or a similar problem. Since only SO members are allowed to vote on questions and not everyone uses the voting system, we used the score count as the second rather than first sorting criterion. For completeness, we included the favorite count as the third sorting metric. The selected 100 posts had 16,124 views on average, while the 100th post already had only 5,291 views.

We manually analyzed all 100 questions, reading both the question and answers, to identify two factors: (1) the question topic and (2) the obstacle the poster is likely facing. To reduce subjectivity, the first two authors independently analyzed the questions and discussed any disagreements. The combined analysis time of the posts was around 33hrs.

Since our study is of exploratory nature, we did not use any pre-defined categories. Instead, we used open coding [12], where we assigned a short description code to each post for each of the two factors above, re-iterating and refining these codes as necessary. At the end, we grouped the identified topics and underlying obstacles into relevant categories. The researcher agreement, measured by the kappa score [35], for these two factors was 0.65 and 0.41 respectively.

2.2 Results

Question Topics. The fact that we found over 1,000 Java cryptography-related questions on SO suggests that developers do indeed face difficulties using cryptography. To understand these difficulties, we look at the topics posters ask about. We identify nine main topics shown in Table 1 and divide four of these main topics into further sub-categories (not shown in table for space limitations). The N/A category contains questions that are not cryptography-related even though they matched our query tags (e.g., general random-number generation or migrating code from Java to .NET).

Table 1: Question topics for the top 100 Java cryptography-related posts (questions) on StackOverflow

Main Topic	# Questions
Symmetric Encryption	37
Public-key Encryption	18
Provider	12
Signature	8
Hashing	7
Keystore	7
Random Numbers	4
N/A	7
Total	**100**

Table 1 shows that 37% of the top 100 Java cryptography questions on SO are related to symmetric encryption. Symmetric algorithms use the same key for encrypting and decrypting the data and are, on large data, more efficient than public-key algorithms. In terms of sub-categories, out of these 37 questions, 41 % are related to how to encrypt data in general. However, we also find more specific questions such as handling padding during encryption (11 %) or dealing with keys and their lengths (14 %).

The next most popular topic we found was *public-key encryption* with 18 questions (18%). We find that half of the posts dealing with public-key encryption ask about keys. It seems that many developers have difficulty correctly setting up and reading the required pairs of public and private keys.

We also found that 12% of the questions are about Java cryptography providers, including how to set them up, the default values of different providers, or whether some algorithm is supported by a specific provider or not. Apart from the seven N/A questions, the remaining identified categories include some questions about signatures, hashing, keystores, and random numbers for cryptographic applications.

S1-Obs.1: 37 % of top 100 Java cryptography related questions on SO are about symmetric encryption.

Obstacles. Table 2 shows the obstacles we identified for the same top 100 questions, after excluding the seven questions categorized as N/A. In 53 questions (57 %), we find that the problem is related to knowing how to use the API. For instance, users are not sure what method calls are needed to generate a key or why the sequence of method calls they use to perform some task (e.g., encrypt certain data) is not working properly. One example is post number 11065063. The poster uses a symmetric cipher and calls the `cipher.update()` method that encrypts part of the input data. However, she ignores the returned encrypted byte array, which causes any following decryption to be incorrect. Another example is post number 1785555 that asks how to generate an initialization vector (IV). From the answers, it seems that a developer can either use `SecureRandom` to generate the IV herself or she can get it through the cipher parameters, using the `IvParameterSpec` class, if the block mode requires an IV. The obstacle in these two examples is a matter of understanding how the API works and which methods should be used to achieve the given task.

In cases such as these two examples and the other 51 questions in this category, we realized that the question poster had some domain knowledge (i.e., she at least knows which algorithms to use) and knew the API to use, but the API

Table 2: Identified obstacles in the analyzed StackOverflow questions from Table 1 (excluding the 7 N/A posts)

Obstacle	# Questions
API use	53
Domain Knowledge	14
Provider & Setup Issues	14
Library Identification	6
Domain knowledge + API use	6
Total	93

complexity led to incorrect usage. This category also includes problems in understanding the underlying API implementation (e.g., knowing the encoding of the return value of an API call or understanding an unclear API error message).

S1-Obs.2: 53 (57 %) of question posters have some domain knowledge, but the API complexity or unclear underlying implementation prevents them from using the correct sequence of method calls & parameters.

However, we also find 14 (15 %) posts where lack of domain knowledge is the main obstacle. For example, we find three different posts (7735474, 9399400, and 9316437) where the question posters confuse encryption and hashing. The third post 9316437 asked how to decrypt a SHA-256 encrypted string. By looking at the answers, we infer that the poster lacks domain knowledge since SHA-256 is a hash function and hash functions are one-way and irreversable by definition. Similar issues include understanding the key sizes supported by different algorithms or what a keystore is. Such users cannot identify the correct algorithms to use in the first place and cannot differentiate between the vast amount of options available. Note that in six additional questions, the obstacle includes both API use and domain knowledge. Additionally, we find six posts (7 %) where the user does not know which Java library to use.

S1-Obs.3: 26 (28 %) of question posters lack domain knowledge (do not know the correct cryptographic algorithms to use) or cannot identify suitable libraries.

In 14 of the cases (15 %) we looked at, the obstacle is setting up the environment. The majority of these problems are related to setting up and using the correct provider. For example, in posts 285624 and 4895773, the same code caused exceptions only on certain machines, because the users did not set up the providers correctly on all the machines they tested on. Another common problem is that certain algorithms only work with a special "unlimited-strength" setup that needs an extra jar file to be installed [5].

S1-Obs.4: 15 % of question posters have problems setting up their environments and providers.

2.3 Intermediate Discussion

S1-Obs.1 and *S1-Obs.2* suggest that developers face problems even for simple tasks such as symmetric encryption. Even when they know the right algorithm to use, they may still need guidance on how to use the APIs properly. On the other hand, *S1-Obs.3* suggests that there are also users who lack the appropriate domain knowledge or are not even sure which libraries to use. Such users require significant guidance to accomplish their tasks. Finally, irrespective of the user's domain knowledge, *S1-Obs.4* suggests that users still run into setup issues frequently, and that JCA's provider architecture is sometimes confusing to developers. Such users may need support with selecting the right provider and setting up their environments correctly.

3. S2: PILOT SURVEY

S2 addresses RQ1-3 and also serves as a focused pilot study in preparation for S4. We use the opportunity that posters of Java cryptography-related questions on SO have actually faced problems themselves. Our goal is to understand the obstacles they faced, the kind of tasks they need to perform, and the solutions they might find useful.

3.1 Study Design

Survey Design. We asked the following questions:

(Q_1) *What is your current occupation?* undergraduate or graduate student, academic or industrial researcher, industrial or freelance developer, other.

(Q_2) *How many years of Java programming experience do you have?* <1 year, 1-2, 2-5, 6-10, 11+ years

(Q_3) *Rate your background/knowledge about cryptography concepts such as encryption, digests, signatures, etc. Not knowledgeable* - I do not know anything about cryptography, *Somewhat knowledgeable* - I have a vague idea about various areas of cryptography and what they are used for, *Knowledgeable* - I am familiar with various areas of cryptography and what they are used for, *Very knowledgeable* - I know all/most areas of cryptography, the different available algorithms, and what they are used for.

(Q_4) *How often do you need to use cryptography in your software applications? Never*, *Rarely* - I need cryptography for less than 33% of the software applications I develop, *Occasionally* - I use cryptography in more than 33% but less than 66% of the software applications I develop, *Frequently* - I need cryptography for more than 66% of the software applications I develop. A similar scale was used by Moreno et al. [22].

(Q_5) *What kind of cryptography-related tasks do you usually implement in your applications?* Free-text.

(Q_6) *How much effort did you go through before resorting to posting your question? Didn't spend much time to be honest* – Stackoverflow just seemed like a reasonable place to start, *I spent a few hours reading resources and trying a few things here and there*, *I had a quick look at some APIs*, but didn't really understand much, and *I spent nights trying to figure this thing out!*

(Q_7) *Describe the steps you took to try to solve the problem yourself before resorting to StackOverflow.* Free-text.

(Q_8) *What obstacles made it difficult for you to accomplish your tasks?* Free-text. We use the exact question format used by Robillard [25].

(Q_9) *Rank the following obstacles to using cryptography in your applications where 1 is the biggest obstacle.* Rank from 1 to 3 for three obstacles we believe might be relevant: *Identifying which concepts and algorithms to use*, *Identifying which Java libraries to use*, and *Identifying how to use the APIs of the identified library.* Question is provided on a separate page from previous question.

(Q_{10}) *What do you think would be a useful tool/technology/idea that can help you complete your cryptographic tasks*

Table 3: Background of Survey 1 participants (SO users). We assign a code to each participant.

Participant Code	Current Occupation	Java Exp. (in years)	Cryptography Knowledge	Cryptography Use
S2-P1	Industrial developer	2 - 5	Somewhat knowledgeable	Frequently
S2-P2	Industrial developer	11+	Very knowledgeable	Rarely
S2-P3	Industrial developer	2 - 5	Knowledgeable	Frequently
S2-P4	Industrial developer	6 - 10	Knowledgeable	Rarely
S2-P5	Undergrad. student	1 - 2	Somewhat knowledgeable	Rarely
S2-P6	Freelance developer	1 - 2	Somewhat knowledgeable	Rarely
S2-P7	Grad. student	2 - 5	Knowledgeable	Rarely
S2-P8	Other: Senior SW Eng.& Team Lead	6 - 10	Knowledgeable	Rarely
S2-P9	Freelance developer	6 - 10	Knowledgeable	Occasionally
S2-P10	Freelance developer	1 - 2	Somewhat knowledgeable	Rarely
S2-P11	Industrial developer	6 - 10	Knowledgeable	Frequently

more correctly, efficiently, etc.? Free-text.

Participant Recruitment. We identified 1,066 SO users who asked Java cryptography-related questions[1]. As SO provides no means to directly contact users, we identified email addresses of only 46 users by visiting their listed websites. We contacted these users, explaining the goal of our project and asking them to fill the survey. Out of these 46 users, eight responded to the survey. To increase the number of participants, we additionally identified users who asked questions with tags `Java` and `Encryption` since a quick analysis of such questions suggests that not all users tag their questions explicitly with `cryptography`. This allowed us to send invitations to 15 additional users, three of whom answered our survey. This resulted in a total of 11 participants.

3.2 Results

Participant Profiles. Table 3 shows the background of the 11 SO posters that participated in this survey. We can see that with the exception of two students (S1-P5 and S1-P7), most of the participants can be considered as professional developers. This provides us some reassurance that the questions they have asked on SO are related to real application development rather than student assignments. The level of Java experience varies between participants though, but is more skewed towards experienced developers. Eight participants have at least two years of Java development experience, with five having at least six years. Most of the participants (seven) say that they rarely need cryptography in the various software they develop. This suggests that they are typical application developers who only sometimes need to use cryptography in their applications. In terms of cryptography knowledge, all participants rated themselves as at least somewhat knowledgeable.

Cryptographic Tasks. We grouped similar free-text answers provided by participants, resulting in nine categories: *encrypt data*, *encrypt files*, *secure connections & communication*, *store/authenticate user login*, *encrypt cookies*, *generate/store secret keys*, *derive keys*, *transfer files securely*, and *protect sensitive data in a database*. Participants were able to provide more than one task. However, with the exception of three tasks, each task was only mentioned by one participant. The three tasks mentioned by more than one participant are *store/authenticate user login* (7 participants – 64 %), *secure connections & communication* (4 participants – 34%), and *encrypt files* (3 participants – 27 %).

[1]Tags `java` and `cryptography` (excluding `javascript`).

S2-Obs.1: The most common task needed by 64 % of participants is storing and authenticating user login.

Effort. In terms of effort, ten (91 %) participants said that they spent a few hours reading resources and trying out solutions on their own. The remaining participant points out that her problem was simply a typo that she kept overlooking. Only six participants described the steps they took before resorting to SO (this was an optional question). Most of these participants mentioned reading lots of API documentation, tutorials, and blogs. One participant even mentioned taking a Coursera cryptography course and reading two cryptography books. This points out the problem that application developers spend valuable time trying to understand the cryptography domain as well as its specific APIs.

S2-Obs.2: To accomplish their cryptography tasks, participants spend at least several hours reading through online resources.

Obstacles. Only six participants answered optional (Q_8). We grouped similar answers together, resulting in three main obstacle categories: (1) documentation (S2- P1, P2, P8, P9, P10), (2) API design (S2- P2, P8, P9, P10), and lack of cryptography knowledge (S2-P4). As expected, the most mentioned obstacle was lack of documentation, which has been previously noted as a general API usability problem [25]. Participants complained that there is no specific place to find answers (S2-P1), as well as about the lack of Java cryptography tutorials (S2-P8) and up-to-date API documentation (S2- P2, P8, P9, P10). In terms of API design, S2-P6 mentioned that it is generally difficult to implement key generation and secure transfer of public/private keys using the API. S2- P1, P4, and P8 discussed problems due to differences between API versions and unclear error messages. S2-P2 also mentioned the need to understand underlying implementation details due to the layer of indirection between the JCA APIs and the algorithm implementations (JCA is designed as a set of abstract classes/interfaces that are implemented by the specific providers). Since only three participants explicitly ranked their free-text obstacles, we could not infer any meaningful conclusions from the rankings.

S2-Obs.3: As obstacles, participants mention lack of documentation, difficulty in API use, and indirection between the APIs and the underlying implementation.

On the other hand, ten participants ranked our three predefined obstacles. With the exception of one participant who did not view identifying the cryptography concepts to use as an obstacle (i.e., did not provide a ranking for it), the

two other obstacles were ranked by all participants. Since the majority of participants ranked all three factors, this suggests that they are indeed obstacles developers face. We considered the one unranked choice as having a rank of ten since higher-ranks are reflected in smaller values. On average, the highest ranked obstacle was identifying how to use the APIs (avg. rank 2.1). This was followed by identifying which Java libraries to use (avg. rank 2.4), and identifying which concepts and algorithms to use (avg. rank 2.7).

S2-Obs.4: Participants consider how to correctly use the APIs as their biggest obstacle.

Developer Support. We received comments from five participants. Three participants asked for better documentation and/or examples while two ask for more high-level APIs that solve common tasks. It seems that developers find it hard to deal with intricate details and knowledge about the APIs and may prefer an input-based black-box kind of API.

S2-Obs.5: Participants ask for better API documentation and/or higher-level abstraction of APIs in the form of common tasks.

3.3 Intermediate Discussion

All our participants considered themselves as at least somewhat knowledgeable about cryptography and yet they all faced problems using cryptography. This suggests that even developers who have some cryptographic knowledge still face problems accomplishing cryptography-related tasks. The obstacles we observe here (*S2-Obs.3* and *S2-Obs.4*) match those identified in S1. Participants' desire for better documentation (*S2-Obs.5*) along with the valuable time they spend reading online resources (*S2-Obs.2*) suggest that improved documentation technology is needed.

4. S3: TASKS IN OPEN-SOURCE SOFTWARE

This study addresses RQ2 by identifying cryptography-related tasks implemented in 100 public GitHub repositories.

4.1 Study Design

To determine common cryptography tasks performed by application developers, we inspected the actual code they write. We used the GitHub Search APIs [3] through a Python wrapper library [8] to find open-source projects that use the Java cryptography APIs provided by JCA. We limited our search to recent Java projects created within the last five years that have more than 100 stars and use the package `javax.crypto` in the code. By filtering out repositories with less than 100 stars, we sought to ensure that the study only considers popular projects that are actually used by people, rather than small projects that nobody uses. In the end, we obtained a list of all repositories that contain matching code files and the links to these files. This resulted in 257 unique projects with 1,452 code files (a project may contain multiple matching code files).

Our goal was to understand what cryptographic tasks are performed in the identified code. Unfortunately, this is a manual task since any topic analysis techniques would not give us an accurate description. To make this task more feasible, we randomly selected 100 projects and split them between the first two authors. We then used open-coding to analyze the identified tasks. In total, we analyzed 301 Java files over 50hrs. While we did not intentionally select repositories based on their domain, our randomly chosen repositories included different types of projects such as web services, Android libraries and apps, content management systems, and web applications. We could identify 41 different types of projects, which shows that the random selection was not focused on a specific domain. The top three categories covered by the selection were Android apps (ten repositories), database or data-storage related projects (ten repositories), and communication projects (nine repositories).

Table 4: Tasks identified from analyzing 100 GitHub projects (a project may implement more than one task).

Task	# Projects
Symmetric encryption	64
Sign & verify	42
Generate secret key	23
Asymmetric encryption	13
Key storage	12
User authentication	10
Secure connection	6
Other	5
Hashing	3
Generate key pair	2

4.2 Results

Table 4 shows the distribution of tasks we identified. We found that most projects (64%) perform some form of symmetric encryption. The type of data being encrypted includes messages, keys, server tokens/response, cookies, documents, PIN codes, and session IDs. We even found passwords being encrypted, a practice that is often discouraged and should be replaced by hashing [10, Chapter 2.5.4].

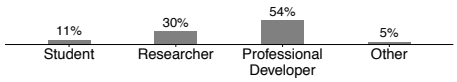
The next most-popular task that occurs in 42% of the projects is signing data and later verifying this signature. Similar to symmetric encryption, we found signatures for a variety of input types including URIs, messages, requests, files, and session IDs. Generating secret keys was also a very common task (23%). This is not surprising since generating a secret key is often a prerequisite to symmetric encryption.

S3-Obs.1: The most common tasks found in the analyzed repositories are symmetric encryption (64%), signing and verifying data (42%), and generating secret keys (23%).

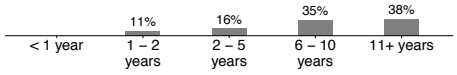
Other tasks we identified include asymmetric encryption, key storage, and user authentication, all occurring in at least 10% of the analyzed projects. The remaining tasks such as hashing, securing connections, or generating key pairs occurred in less than 10% of the projects we analyze. We also included an "Other" category that groups very infrequent tasks in our dataset, such as the checking of certificates.

4.3 Intermediate Discussion

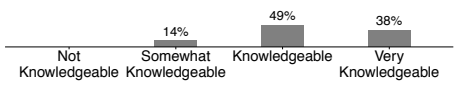
In S2, we observed that storing and authenticating user login is the most common task needed by participants (*S2-Obs.1*). On the other hand, we found that only 10% of the analyzed GitHub projects implement some form of user authentication. This discrepancy could be due to the domain of the projects analyzed. Additionally, it is hard to generalize based on the 11 participants from S2, which is why we use the survey in S4 for further investigation. On the other hand, the fact that 64% of the analyzed projects need some



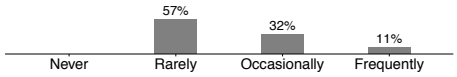
(a) Occupation



(b) Years of Java experience



(c) Cryptography knowledge



(d) Frequency of cryptography use

Figure 1: Background of Study 4 participants

form of symmetric encryption (*S3-Obs.1*) matches *S1-Obs.1* that 37% of the SO questions are about symmetric encryption. Since developers often need this task, it is likely they will have questions about how to accomplish it.

5. S4: GENERAL SURVEY

In our final study, we surveyed Java application developers who use cryptography APIs. This survey addresses all three research questions posed in the introduction and validates some of of our observations from the previous three studies.

5.1 Study Design

Survey Design. The following is the list of questions posed in the survey (all mandatory). During its design, we used our experience with the pilot survey from S2.

(Q_{1-3}) Same as Section 3.1(Q_1) – (Q_3)

(Q_4) Same as Section 3.1(Q_4), but using percentage of development tasks instead of percentage of projects to avoid corner situations where the participant only worked on one project. Answering *never* here ends the survey.

(Q_5) *What are the most common cryptography-related tasks you need in your applications? Rank the tasks ...* Tasks identified from S2: *Store/ authenticate user login*, *Encrypt files*, *Secure connections and communications*, and *Transfer files securely*. Additional tasks can be added through "Other" fields. Note that the timeline of S3 and S4 overlapped, which is why S3 tasks are not included here.

(Q_6) *Did you use Java Cryptography APIs before?* Yes/No. Answering No ends the survey.

(Q_7) *Please rank the Java cryptographic libraries/APIs below according to your frequency of use where 1 is most used.* This question examines if most developers actually use the JCA APIs. Choices include *Java Cryptography Architecture (JCA) APIs (irrespective of provider)* and *Lightweight Bouncy Castle APIs* [2]. Additional APIs can be added through "Other".

(Q_8) *Thinking of your most-used API, <(Q_7) rank#1 choice>, how would you rate its ease of use in terms of accomplishing your tasks correctly and securely? very hard to use*, *hard to use*, *easy to use*, and *very easy to use.*

(Q_9) *What [obstacles make it hard for you to learn and use|features make it easy to use] <(Q_7) rank#1 choice>?* Free-text.

(Q_{10}) *Thinking of when you have a new cryptography-related task to implement in your software (e.g., ...) do you ever have difficulties with the following?* Short-form of obstacle choices is shown in Fig. 3, each with choices *never*, *rarely*, *occasionally*, *frequently*, and *don't know.* Note that we divide the obstacle previously used in S2, "Identifying which concepts and algorithms to use", into two obstacles "Identify the correct algorithm (e.g., AES vs DES) to use" and "Identify which concepts (e.g., encryption vs hashing) to use".

(Q_{11}) *S*ame as Section 3.1(Q_{10})

Participant Recruitment. Since there is no way of determining the whole population of developers who use Java cryptography APIs, we used non-probabilistic sampling [19]. We advertised the survey on our Social Media accounts and also asked colleagues to forward to any relevant Java developers they know (*snowball sampling* [17]). Additionally, we emailed developers who have committed to Java files that use cryptography APIs on GitHub. However, we made sure that these developers do not belong to the 100 projects we analyzed in Study 3.

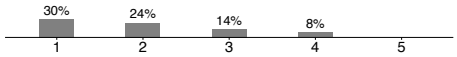
Data Analysis. Fourty-three developers completed the survey. However, during data analysis, we removed six participants who indicated that they do not need cryptography in any of their development tasks (Q_4) or have not used Java cryptography APIs before (Q_6). We base our findings below on the remaining 37 participants. We used quantitative analysis for multiple choice and rating questions and qualitative analysis (mainly open coding) for free-text answers. Each participant is assigned a code, *S4-P#*.

5.2 Results

Participant background. Fig. 1 shows participants' backgrounds. For simplicity, we group undergraduate and graduate students as *Student*, academic and industry researchers as *Researcher*, and freelance developer and industrial developer as *Professional Developer*. Most participants are professional developers (54 %) and only 11 % are students. The majority of participants have at least 6 years of Java development experience (73 %). Additionally, the majority of participants rated themselves as at least knowledgeable about cryptography (86 %). However, most participants (57 %) rarely need cryptography in their development tasks.

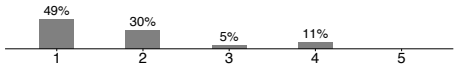
Common cryptography tasks. Fig. 2 shows the most common cryptography tasks as ranked by participants. Figures 2a–2d show the four tasks we provide while Fig. 2e shows the only free-text task (through the "Other" fields) that is mentioned by multiple participants. Note that percentages do not add up to 100% since participants can leave a task without a rank if they do not encounter it in their development tasks. However, they have to mark at least one task (i.e., rank 1).

Fig. 2 shows that securing connections and communications was the highest-ranked task (49 % of participants ranked it as #1, Fig. 2c). This is followed by authenticating users that is ranked #1 by 30 % (Fig. 2a). We also determined the average rank per task, again assigning rank 10 to any un-

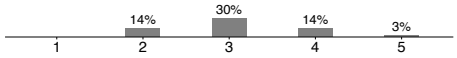


(a) Store/authenticate usernames & passwords

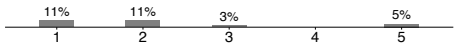
(b) Encrypt file



(c) Secure connections & communications



(d) Transfer files securely



(e) Other: Signing & verifying data

Figure 2: Task ranking by Study 4 participants (Q_5)

ranked tasks. This also gives preference to tasks that were ranked by more participants. This results in the following ordered task ranking: securing connections and communications (avg. rank 2.22, 35 participants), authenticating users (avg. rank 3.95, 28 participants), encrypting files (avg. rank 5.03, 26 participants), transferring files (avg. rank 5.89, 22 participants), and signing & verifyinig data (avg. rank 7.70, 11 participants).

S4-Obs.1: The top three tasks needed by participants are securing connections and communications, authenticating user logins, and encrypting files.

Additional tasks mentioned by participants in the "Other" fields include checking licenses, hashing, generating and managing keys, and handling OpenPGP emails. However, each of these tasks was mentioned only by one participant.

Used Java APIs. When ranking their most-used Java APIs (Q_7), 57 % of participants ranked `JCA` as #1 while 35 % ranked `BouncyCastle` as #1. Apart from `SpongyCastle` that was mentioned by two participants in the "Other" field (rank #1 and rank #2), additional APIs listed by participants include `J2ME Crypto APIs` (ranked #1), `FlexiProvider` (ranked #3), `Keyczar` (ranked #3), `BCMail` (ranked #2), and `BCOpenPGP` (ranked #3) – each mentioned only once.

S4-Obs.2: JCA is the most used API (rated #1 by 57%), but developers also use other APIs.

To analyze the ease of use ratings (Q_8), we aggregated *very hard* and *hard* as *hard* and *very easy* and *easy* as *easy*. We found that 65 % of participants rate their commonly used library as hard to use, including those who provided their own API at rank #1. We could not detect any statistically significant relationship between the most-used library and the ease of use rating (Chi-squared test p-value=0.479). We also did not find a correlation between participants' backgrounds and their ease of use ratings. The occupation and level of cryptography knowledge of the participants who found the APIs hard to use was very diverse. There were 8 researchers, 14 professional developers, 1 CEO, and 1 technical evangelist. Three of them were somewhat knowledgeable, 12 knowledgeable, and 9 very knowledgeable about cryptography.

S4-Obs.3: Irrespective of the library used and background, 65% of participants find the APIs hard to use.

Obstacles in hard-to-use APIs. We elicited obstacles from 21 participants who find their most-used library hard to use (three participants put 'none' or '.'). Since we could not find a relationship between the library being used and the ease of use rating, we collectively analyzed all listed obstacles.

Out of the 21 participants who provided feedback, ten (48 %) participants said that the API is not high-level enough (S4- P8, P12, P16, P19-20, P24-25, P27, P33-34), eight (38 %) of which list it as their #1 obstacle. Participants mentioned that the APIs are too complex even for basic tasks. For example, S4-P24 says, *"It's way too low level and modular. Most developers need simple high-level abstractions, not the complete toolbox with a wide variety of implementation."*

Nine (43 %) participants mention poor documentation as one of their obstacles (S4- P3, P7, P10, P16, P19-20, P23, P25, P31), with four (19 %) rating it as #1. Most of these participants mentioned the lack of useful examples.

Seven participants (33%) mentioned specific API design obstacles, with three (14 %) as their #1 obstacle. Three of those complained that debugging client code using the API and understanding the API error messages is difficult (S4- P11, P23, P31). The others complained about not knowing which methods and parameters to use (S4-P1), especially since there are *"misleading default values for algorithms/methods (e.g., [insecure] ECB mode as default)"* (S4-P3) as well as the lack of proper constants for algorithm names (S4-P12). As one participant pointed out, such things make it *"easy to [create] dangerous errors"* (S4-P8). To illustrate these points, consider how a symmetric encryption cipher is created: `Cipher cipher = Cipher.getInstance("AES")`. Note that the algorithm name (i.e., the cipher algorithm in this case) is a string instead of a proper Java constant, as pointed out by S4-P12. Also note that an encryption cipher needs both a mode and a padding scheme to function properly. When no mode is provided as in the code above, the standard provider defaults to the insecure ECB mode [14] as discussed by S4-P3. Finally, as S4-P1 pointed out, developers must understand which modes and padding schemes are secure in order to correctly pass them as parameters to the algorithm. This is actually a mix of an API use problem as well as domain knowledge. Developers must know the state of the art in proper security (domain knowledge) and must know how to pass these parameters to the appropriate method calls (API use).

The remaining participants (S4- P28, P35, P36) mentioned class-loading issues, missing functionality in the API which means resorting to additional libraries, and the fact that it is very time consuming to understand the API.

S4-Obs.4: Participants' main obstacles are lack of high-level APIs, poor documentation, and bad API design (e.g., misleading defaults & difficult debugging).

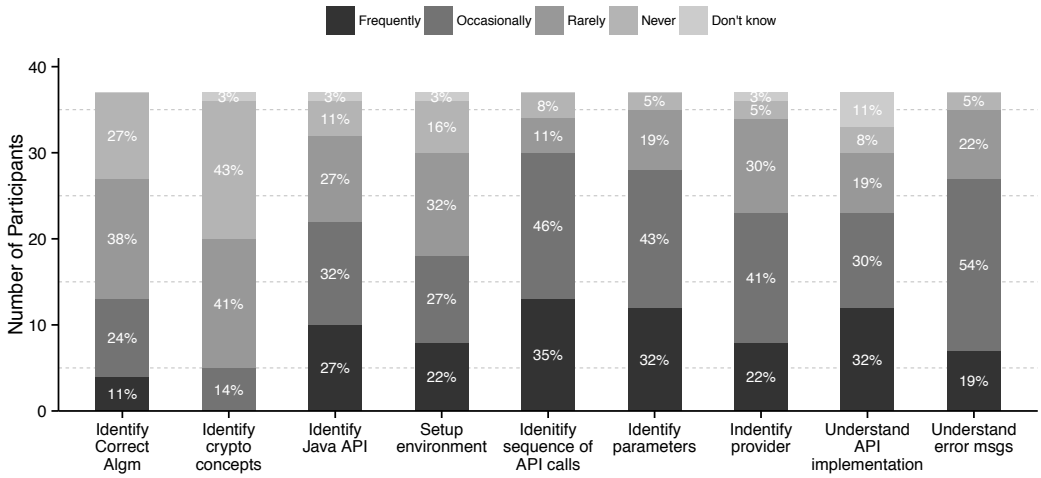


Figure 3: Obstacles rated by Study 4 participants

Features of easy-to-use APIs. Similar to Robillard and DeLine [26], we did not find the feedback on positive API features helpful. Since many participants only provided generic answers such as "Good API" or put periods as answers, we could only meaningfully evaluate seven answers. Surprisingly, six of these participants mentioned completeness, level of abstraction, documentation, and clean API endpoints as positive features (S4-P9, P13, P17-18, P21, P37). The seventh participant, S4-P29, mentioned that it is a matter of a steep learning curve rather than being generally hard.

Rated obstacles. We first checked if there is any correlation between the ratings of different obstacles using a Spearman correlation test. We found statistically significant strong correlations ($\rho < -0.5$ *or* $\rho > 0.5$ and p-value < 0.01) between the ratings of four pairs of obstacles: (1) identifying the API to use and setting up the environment ($\rho = 0.66$), (2) identifying the API to use and identifying the provider to use ($\rho = 0.59$), (3) setting up the environment and understanding the underlying implementation ($\rho = 0.53$), and (4) identifying sequence of method calls and identifying parameters ($\rho = 0.54$). We interpret these strong correlations as follows. For the first and second pair, identifying the API to use is usually followed by setting up the environment and determining which provider to use. Thus, the three obstacles can be seen as part of the startup process. The fourth correlation suggests that it may be the case that developers do not distinguish between troubles with method call sequences and those with the parameters to provide. We cannot find a reasonable explanation for the third correlation, but we note that the correlation coefficient is not that much higher than the threshold. All other obstacles were weakly correlated, suggesting that participants view them as distinct obstacles.

Despite the correlations mentioned above, we still looked at all the rated obstacles since they present potential features that future solutions should include. Fig. 3 shows how often participants face the obstacles we asked them about. Darker colors show more frequently faced obstacles. The figure shows that identifying the correct sequence of method calls is the most frequent obstacle (35%). This is followed by understanding the underlying API implementation (32%) and identifying the parameters to use (32%).

S4-Obs.5: The most frequently faced obstacle is identifying the correct sequence of method calls (frequently faced by 35%) followed by understanding the underlying API implementation and identifying the parameters to use (both frequently faced by 32%) .

None of the participants, including those with limited cryptography knowledge, marked identifying which concepts to use as a frequent obstacle (second column of Fig. 3). In fact, 43% of the participants said they never face such an obstacle. However, the first column also shows that 35% of participants cannot, at least occasionally, identify the correct algorithm to use. This suggests that most participants know which area of cryptography to use but may not always be sure about the trade-offs between various algorithms.

S4-Obs.6: 43% of participants never have problems identifying relevant cryptography concepts, but 35% cannot, at least occasionally, identify the correct algorithm to use.

Note that a Chi-squared test of independence showed no statistically significant relationship between participants' backgrounds, or their most used API, and their obstacle ratings.

Desired support. Twenty-seven participants provide suggestions for how to improve cryptography use in Java. We identified three general categories for the suggestions: *different API design* (14 participants), *better documentation* (10 participants), and *tool support* (6 participants). We also categorized parts of the feedback we got from three participants as *other* since the suggestions were specific feature requests such as improving certificate creation.

In terms of API design, nine participants mentioned words such as *use cases*, *task-based*, or *high-level* design. S4-P24 comments that *"cryptographic libraries should be reserved to people who implement protocols. The average software developer writing an application needs something much higher level."*. Along the same lines, S4-P6 asked for libraries that *"[provide] simple API calls (one or two methods and simple parameters) for different use cases"*. S4-P27 echoed that by suggesting having *"higher level task-oriented APIs for things like public key crypto, key exchanges, secure local storage, [..., etc.]"*. S4-P10 commented that *"the ability to [perform]*

simple cryptographic tasks in Java without jumping through hoops would be brilliant."

Participants also said that the API documentation can generally improve with more examples. Finally, some participants provided tool-based suggestions such as having a CryptoDebugger (S4-P11), cryptography-aware testing tools (S4-P13, S4-P17), analysis tools that identify where cryptography protection is needed and find configuration mistakes and weak algorithms (S4-P17), and code templates or factories for common tasks (S4-P7, S4-P9) through code generation IDE plugins (S4-P6).

S4-Obs.7: Participants suggest task-based solutions whether in the form of better API design, examples in documentation, or analysis and code generation tools.

6. DISCUSSION

6.1 Putting it All Together

RQ1: What obstacles, if any, do developers face during the use of Java cryptography APIs? Our results confirm that developers face obstacles while using the APIs. First, 65% of S4 participants found the APIs hard to use (*S4-Obs.3*) and S2 participants also spend a considerable amount of time reading resources (*S2-Obs.2*). Specifically, developers have problems **determining the correct sequence of method calls**, because **APIs are too complex to use** (*S1-Obs.2*, *S2-Obs.3*, *S2-Obs.4* and *S4-Obs.4*). S1 showed that another obstacle faced by developers is **lack of domain knowledge** (*S1-Obs.3*). However, based on *S4-Obs.6*, we speculate that the problem here often lies in determining the right algorithm to use rather than which general area of cryptography is needed. Another obstacle that appeared in both S1 and S4 is **understanding the underlying API implementation** (*S1-Obs.2*, *S4-Obs.5*). Finally, developers seem to have problems in **correctly setting up their environments** to use the APIs, supported by both *S1-Obs.4* and the fact that 22% of S4 participants frequently face this obstacle (Fig. 3).

RQ2: What are the common cryptography tasks developers need? The three most common cryptographic tasks (those highly-rated or mentioned frequently in more than one study) are **storing and authenticating user login** (*S2-Obs.1*, *S4-Obs.1*), **securing connections and communications** (*S4-Obs.1*), and **different forms of symmetric encryption** (*S1-Obs.1*, *S3-Obs.1*, *S4-Obs.1*).

RQ3: What tools or ideas would help developers use cryptography more effectively? Participants of both surveys advocated for more task-based solutions whether in the form of **better example-based/task-based API documentation** (*S2-Obs.5*, *S4-Obs.7*), **higher-level abstractions of APIs** (*S2-Obs.5*, *S4-Obs.7*), or **tools that catch common mistakes or generate code templates** (*S4-Obs.7*).

6.2 Moving Forward

Based on our findings, we see three directions for moving forward. The first is, naturally, to improve API documentation. Since this relies on the API creators themselves, we do not currently see a way to enforce this. Other documentation-related solutions include those that synthesize code examples for specific APIs [26] or those that try to present the existing documentation and online resources in ways more useful to developers (e.g., [24, 30, 33]).

The second is to encourage API designers to hide some of the unnecessary details and provide their clients with more high-level or task-based method calls. This was something suggested by more than one participant. There are several libraries—some non-Java—trying to achieve this (e.g., NaCl [7], Keyczar [6]). We also do not see this as something that can be enforced, but at least guidelines can be developed to help API designers achieve this.

The third solution removes the dependency on the API designers by building automated support tools on top of the APIs. Based on the suggestions we got, (task-based) code-generation tools are one example of such tools. Other examples are analysis or debugger tools that warn users or help them debug cryptography vulnerabilities. Based on our findings, we suggest the following list of (task-based) tool features that we encourage future tool or solution designers to consider. However, we do not currently know the relative importance of these features.

- Support *at least* the following tasks: storing and authenticating user login, securing connections and communications, and symmetric encryption.
- Given a particular cryptography task:
  - Identify the relevant library to use.
  - Setup the identified library correctly (including providers).
  - Identify the correct algorithm to use.
  - Identify the correct algorithm settings to use.
  - Identify the correct sequence of API method calls and their appropriate parameters.
- Given a piece of code that uses cryptographic APIs, identify any potential vulnerabilities.
- Given a piece of code, identify where cryptographic protection might be needed.

6.3 Threats to Validity

S1 and S3 relied on manual analysis of posts and code. The observations we make are, of course, subjective. In S1, we mitigated this risk by having two of the authors analyze the posts and discuss any discrepancies. In S3, due to the higher analysis time-cost, the projects were equally divided among two of the authors, providing us a mix of different interpretations. We also published our coded dataset on our artifacts page [1] to facilitate replication or further analysis.

The number of participants of S2 is fairly limited. Even though the results of the study provide interesting insights, we do not base any conclusions on S2 alone unless confirmed by at least one other study. The participants of S4 may not be completely representative of the whole population due to the non-probabilistic sampling we follow, which subjects the study to non-response bias. However, since our population is very specific (developers who use Java cryptography APIs) and our participants have a diverse background in both Java and cryptography experience, we believe our observations are still relevant to the larger population. Since our findings only accurately reflect the opinions of our participants, we encourage other researchers to conduct similar surveys on an even larger scale. Additionally, our survey did not include anyone without any cryptography knowledge. We did not intentionally filter such participants during recruitment or during analysis. However, since we were recruiting developers who already used the Java cryptography APIs, it is very unlikely that any of them would still have no cryptography knowledge. The list of libraries presented to participants in

S4-(Q_7) is not comprehensive. However, our experience analyzing posts in S1 suggests that the two libraries we listed there are the most popular ones. We also allowed participants to add their own libraries. This additional list of libraries identified can be used to guide future studies.

Our findings may suffer from confirmation bias. In our previous work [11], we presented a possible solution to help developers use Java APIs more securely. It may be the case that we, subconsciously, tried to confirm our prior beliefs. However, we argue that this is not the case since many of our prior beliefs are not confirmed by our studies (e.g., difficulties in differentiating between cryptography concepts). Additionally, we found several new tasks (e.g., securing connections) and tool requirements (e.g., setting up the environment and identifying missing cryptography protection in code) that we did not previously consider.

7. RELATED WORK

Misuse of cryptographic APIs. Researchers have already established many security vulnerabilities due to incorrect usage of cryptography APIs. Lazar et al. [20] manually investigated 269 published security vulnerabilities and found that 83% of them are caused by misuse of cryptographic libraries. Egele et al. [14] statically analyzed 11,748 Android apps for API misuse and found that 88% of these apps do in fact violate at least one of six basic cryptography rules. Similarly, Fahl et al. [15] also found that SSL API misuse causes many Android apps to be vulnerable to Man-in-the-Middle attacks. Georgiev et al. [16] show that even major web applications misuse SSL certificate validation libraries, allowing the authors to extract sensitive information such as credit card numbers. Most of the solutions suggested by these papers focused on preventing consequences rather than addressing the underlying problem of API misuse. An exception is our previous work [11] that proposed a tool to address API cryptography misuse. However, the tool does not include all features identified in Section 6.2, and we did not empirically validate the claims on which we design it—a gap that is filled by the studies in this paper where we examine the actual causes of misuse from a developer perspective.

Assessing General API Usability. Robillard and DeLine [28] surveyed and interviewed Microsoft developers to understand their API learning obstacles. They found that poor documentation is a major learning obstacle. Our work here is different in that we only focus on one API category, namely Java cryptography APIs, with the goal of finding ideas and requirements for solutions, not necessarily in the form of better documentation, that may help application developers use cryptography more easily and securely. Moreover, we are not limited to views from one company.

The work by Zibran et al. [37] also tried to identify API usability factors. They manually analyzed bug reports of four projects to identify feedback given by its API users. Hou and Li [18] also used some manual analysis similar to the one we use in S1. However, they focused on the newsgroup forum of a particular API, namely `Swing`. While we use SO in S1 and use a different classification scheme that emerged during open-coding, some of the obstacles we find are similar to theirs (e.g., wrong environment configurations). Roover et al. [13] also explored how certain APIs are used in a large project corpus with the goal of providing insights to both API designers and API users.

A main advantage of our work is that we do not rely on one source of information, but rather combine three different sources (SO, GitHub projects, and developer surveys).

API Documentation. Researchers also looked at how to assess and improve API documentation since it was often identified as a usability obstacle. This included identifying documentation issues through surveying industrial developers [34], identifying relevant documentation parts given a particular API element [27], and developing a taxonomy of API documentation knowledge patterns that can be used by practitioners to evaluate their own API documentation [21]. It has also been established that crowd-sourcing sites such as SO provide a rich documentation for many APIs [23].

API Usage Protocols. Sunshine et al. [31] focused on API usage protocols (restrictions on the order of API calls). Similar to our S1, they also relied on SO, but looked for questions that are specifically related to usage protocol restrictions. Similarly, Saied et al. [29] observed how four types of API usage constraints are exhibited in several APIs. While such usage restrictions are not our main focus, such studies complement ours and understanding API usage restrictions can alleviate developers' difficulties in identifying the correct sequence of method calls to use.

Proposed Solutions There have also been other solutions to help both API users and designers, such as informing API designers of common problems with their API [36] or recommending the relevant libraries to use for application developers [32]. Treude et al. [33] also recently propose the idea of task-based documentation navigation. They extract tasks from documentation as well as their related code elements. Our findings support that tasks are effective in helping developers use an API. We also empirically identified the cryptographic tasks relevant to developers. Finally, many solutions are based on synthesizing relevant API usage examples. We point the reader to the survey by Robillard et al. [26] that summarizes the techniques used in this area.

8. CONCLUSION

Many security vulnerabilities are caused by developers' incorrect use of cryptography APIs. However, it is not clear what obstacles such developers face when using cryptography in their applications. We reported on an empirical study to investigate such obstacles, in the context of Java, through examining questions on StackOverflow, GitHub repositories, and two surveys of a total of 48 developers. Our findings showed that developers do indeed face difficulties in using the Java cryptography APIs. We found that developers commonly need to authenticate users and store login data, to establish secure connections, and to encrypt different forms of data. In our surveys, developers indicated that the existing APIs are too low-level and asked for task-based solutions, whether in the API design, documentation, or through assistance tools. Based on these observations, we recommended a list of features that such solutions should include. This list can guide future tool and solution designs.

9. ACKNOWLEDGMENTS

This work is funded by the DFG, project E1 in CRC 1119 CROSSING. At the time this research was conducted, Eric Bodden was at Fraunhofer SIT and TU Darmstadt.

10. REFERENCES

[1] Accompanying online artifact page. http://www.st.informatik.tu-darmstadt.de/artifacts/crypto-api-misuse.

[2] Bouncy Castle. www.bouncycastle.org.

[3] GitHub search APIs. https://developer.github.com/v3/search/.

[4] Java Cryptography Architecture (JCA). http://docs.oracle.com/javase/6/docs/technotes/guides/security/crypto/CryptoSpec.html.

[5] JCE unlimited strength setup. http://www.oracle.com/technetwork/java/javase/downloads/jce8-download-2133166.html.

[6] Keyczar. https://github.com/google/keyczar.

[7] NaCI: Networking and crytptography library. http://nacl.cr.yp.to/.

[8] Python wrapper for GitHub APIs. https://github.com/sigmavirus24/github3.py.

[9] Stackexchange data explorer. http://data.stackexchange.com/stackoverflow/queries.

[10] R. J. Anderson. *Security engineering*. Wiley, 2008.

[11] S. Arzt, S. Nadi, K. Ali, E. Bodden, S. Erdweg, and M. Mezini. Towards secure integration of cryptographic software. In *Proc. of the SIGPLAN Symposium on New Ideas in Programming and Reflections on Software at SPLASH (Onward!)*, 2015. Accepted to appear. https://www.informatik.tu-darmstadt.de/fileadmin/user_upload/Group_EC-Spride/Publikationen/Onward2015.pdf.

[12] J. W. Creswell. *Qualitative inquiry and research design: Choosing among five approaches*. Sage, 2012.

[13] C. De Roover, R. Lammel, and E. Pek. Multi-dimensional exploration of API usage. In *Proc. of the International Conference on Program Comprehension (ICPC)*, pages 152–161, 2013.

[14] M. Egele, D. Brumley, Y. Fratantonio, and C. Kruegel. An empirical study of cryptographic misuse in Android applications. In *Proc. of the Conference on Computer and Communications Security (CCS)*, pages 73–84, 2013.

[15] S. Fahl, M. Harbach, T. Muders, M. Smith, L. Baumgärtner, and B. Freisleben. Why Eve and Mallory love Android: An analysis of android SSL (in)security. In *Proc. of the Conference on Computer and Communications Security (CCS)*, pages 50–61, 2012.

[16] M. Georgiev, S. Iyengar, S. Jana, R. Anubhai, D. Boneh, and V. Shmatikov. The most dangerous code in the world: Validating SSL certificates in non-browser software. In *Proc. of the Conference on Computer and Communications Security (CCS)*, pages 38–49, 2012.

[17] L. A. Goodman. Snowball sampling. *The annals of mathematical statistics*, pages 148–170, 1961.

[18] D. Hou and L. Li. Obstacles in using frameworks and APIs: An exploratory study of programmers' newsgroup discussions. In *Proc. of the International Conference on Program Comprehension (ICPC)*, pages 91–100, 2011.

[19] M. Kasunic. Designing an effective survey. Technical Report CMUSEI-2005-HB-004, Software Engineering Institute, Carnegie Mellon University, 2005.

[20] D. Lazar, H. Chen, X. Wang, and N. Zeldovich. Why does cryptographic software fail? A case study and open problems. In *Proc. of the ACM Asia-Pacific Workshop on Systems (APSys)*, pages 7:1–7:7, 2014.

[21] W. Maalej and M. Robillard. Patterns of knowledge in API reference documentation. *IEEE Transactions on Software Engineering (TSE)*, 39(9):1264–1282, 2013.

[22] L. Moreno, G. Bavota, M. D. Penta, R. Oliveto, and A. Marcus. How can I use this method? In *Proc. of the International Conference Software Engineering (ICSE)*, 2015.

[23] C. Parnin, C. Treude, L. Grammel, and M.-A. Storey. Crowd documentation: Exploring the coverage and the dynamics of API discussions on Stack Overflow. *Georgia Institute of Technology, Tech. Rep*, 2012.

[24] L. Ponzanelli, G. Bavota, M. Di Penta, R. Oliveto, and M. Lanza. Mining stackoverflow to turn the IDE into a self-confident programming prompter. In *Proc. of the Working conference on Mining Software Repositories (MSR)*, pages 102–111, 2014.

[25] M. Robillard. What makes APIs hard to learn? Answers from developers. *IEEE Software*, 26(6):27–34, 2009.

[26] M. Robillard, E. Bodden, D. Kawrykow, M. Mezini, and T. Ratchford. Automated API property inference techniques. *IEEE Transactions on Software Engineering (TSE)*, 39(5):613–637, 2013.

[27] M. P. Robillard and Y. B. Chhetri. Recommending reference API documentation. *Empirical Software Engineering*, pages 1–29, 2014.

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

[29] M. Saied, H. Sahraoui, and B. Dufour. An observational study on API usage constraints and their documentation. In *Proc. of the International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 33–42, 2015.

[30] S. Subramanian, L. Inozemtseva, and R. Holmes. Live API documentation. In *Proc. of the International Conference Software Engineering (ICSE)*, pages 643–652, 2014.

[31] J. Sunshine, J. Herbsleb, and J. Aldrich. Searching the state space: A qualitative study of API protocol usability. In *Proc. of the International Conference on Program Comprehension (ICPC)*, 2015.

[32] F. Thung, D. Lo, and J. Lawall. Automated library recommendation. In *Proc. of the Working conference on Reverse Engineering (WCRE)*, pages 182–191, 2013.

[33] C. Treude, M. Robillard, and B. Dagenais. Extracting development tasks to navigate software documentation. *IEEE Transactions on Software Engineering (TSE)*, 41(6):565–581, 2015.

[34] G. Uddin and M. Robillard. How API documentation fails. *IEEE Software*, 32(4):68–75, 2015.

[35] A. J. Viera, J. M. Garrett, et al. Understanding interobserver agreement: The kappa statistic. *Fam Med*, 37(5):360–363, 2005.

[36] W. Wang, H. Malik, and M. Godfrey. Recommending

posts concerning API issues in developer Q&A sites. In *Proc. of the Working conference on Mining Software Repositories (MSR)*, 2015.

[37] M. Zibran, F. Eishita, and C. Roy. Useful, but usable? factors affecting the usability of APIs. In *Proc. of the Working conference on Reverse Engineering (WCRE)*, pages 151–155, 2011.