UNIVERSITÄT
BERN

Cryptographic APIs

Add subtitile

Bachelor Thesis

Sophie Gabriela Pfister
from
Bern, Switzerland

Philosophisch-naturwissenschaftlichen Fakultät
der Universität Bern

31. Dezember 2021

Prof. Dr. Oscar Nierstrasz
Dr. Mohammad Ghafari, Mohammadreza Hazhirpasand
Software Composition Group
Institut für Informatik

University of Bern, Switzerland

Abstract

Contents

1 Introduction

Cryptography is a fundamental part of our digital world. It provides techniques to ensure confidentiality, authenticity, and integrity of information. Yet, Buchanan [3] describes the internet as an unsafe place. He complains that too little security is implemented in the services and protocols used. He insists that "the next generation of the Internet [...] must be built in a trustworthy way" (Buchanan, 2017, [3], p. 1).
At first sight, this statement seems surprising, since there are cryptography libraries for all common programming languages. They support developers building secure applications by providing services such as hashing, message authentication, symmetric, and asymmetric encryption. However, past research revealed that software developers are struggling to make use of them. They often lack *usability*. As you will see in chapter 2.2, this can be observed for a wide range of cryptography libraries and programming languages. The results of Acar *et al.* [1] imply that unusable cryptography libraries do not only prevent developers from writing functioning code but also lead to security vulnerabilities as programmers are more likely to misuse them.

The usability of an API depends on many aspects that influence each other (see chapter 2.1). The easiest way to increase an existing API's usability is probably improving documentation quality. It can be modified without the risk that working applications break. Additionally, a good documentation is a strong predictor for both, functioning and secure code. Acar *et al.* emphasize "the importance of creating official documentation that is useful enough to keep developers from searching out unvetted, potentially insecure alternatives" (2017, [1], p. 167).

Within the context of API usability, documentation usability, API misuse and unsafe code, we investigated to following research questions:

1. What issues do programmers face when implementing symmetric encryption using Java Cryptography Architecture?
2. What security risks can be found in code and advises shared on Stack Overflow referring to the implementation of symmetric encryption scenarios Java Cryptography Architecture?
3. To what extent are these issues linked to missing or inadequate documentation?

We focused on one library and one use case to gain a deeper understanding of the issues than previous research that analyzed the libraries on a more general level. The decision for the Java Cryptography Architecture (JCA) library and the use case of symmetric encryption was based on the findings by Nadi *et al.* [8] from 2016. They investigated the issues Java developers face when implementing cryptography tasks. According to their result, JCA is the most popular cryptography library for Java and symmetric encryption is the most common cryptography task. Also, JCA acquired FIPS-140 standards issued by NIST [1] to specify the requirements for cryptography libraries and modules [2].

To answer our research questions, we analyzed 150 threads from Stack Overflow containing issues related to our scope. We identified the issues as well as potential security risks and derived a set of questions that these issues relate to. Afterwards, we sought the answer in the documentation.

This thesis first presents the state of research in the related work chapter. The third chapter then describes the methodical approach of our study more precisely. You can find the results as well as their interpretation in the fourth chapter. The fifth chapter includes the conclusion as well as conductive thoughts. In the ”Anleitung zum Wissenschaftlichen Arbeiten” (chapter six), you can learn more about Mayring’s guidelines for qualitative content analysis that were followed during our analyses.

[1] National Institute of Standards and Technology
[2] FIPS-140-3

2
Theoretical Background and Related Work

2.1 API Usability

ISO 92411-11:1998 defines usability as ”the extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use” (ISO 9241-11, [**?**], p.2). Although this definition is very precise, it is not specific. It therefore does not tell us how to measure usability nor does it help us to determine whether a product is usable or not.

Past research on API usability approached the topic in different ways. The literature about API usability is therefore rather heterogeneous. While some researchers defined metrics, guidelines and heuristics, others focused on the concept of usability and tried to expand it such that it meets the requirements of the domain. Mosquiera-Rey *et al.* [7] combined these approaches by extending the usability model elaborated by Alonso-Ríos *et al.* [2] with a contextual dimension and mapping existing guidelines to the model's categories. They ended up with a total of 45 heuristics belonging to 9 different main aspects of usability.

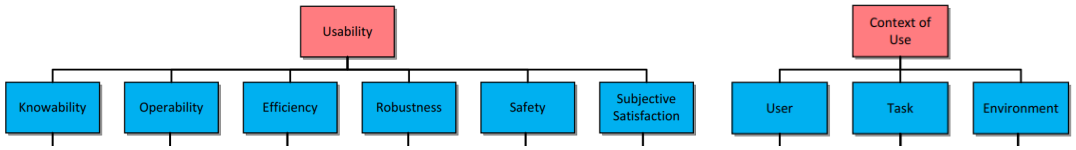

Figure 2.1: First Level of Usability and Context of Use Taxonomies (Mosquiera-Rey *et al.*, 2018, [7], p. 49 ff.)

This example is only supposed to illustrate the complexity of the domain. In the following section, we are focusing on the usability of documentation as this is what we are investigating in this thesis.

2.1.1 Usability Criteria for Documentation

According to Mosquiera-Rey *et al.* [7], documentation has an influence on the *Knowability* of an API. This property indicates to what extent a programmer can ”understand, learn, and remember how to use the system” (Mosquiera-Rey *et al.*, 2018, [7], p. 48). One heuristic says that the documentation should not contain irrelevant information such as meta data or obsolete and redundant comments. However, it ”should include code samples for the most common scenarios” (Mosquiera-Rey *et al.*, 2018, [7], p. 55) as well as tutorials on how to use the API. It also should identify deprecated methods, explain why these are deprecated, and propose alternatives. Another heuristic that is indirectly related to documentation says that ”names should be self-explanatory” (Mosquiera-Rey *et al.*, 2018, [7], p. 55). The programmer should not have to look at the documentation to understand what a function does.

ToDo

2.2 Usability of Cryptography Libraries

In 2016, Green & Smith [4]

2.3 Misuse of Cryptography Libraries

3 Methodology

To answer our research questions, we first had to identify the issues programmers face when implementing symmetric encryption using JCA. We derived them by analyzing 150 threads on Stack Overflow which is one of the most popular Q&A forums for programmers. We also scanned those threads regarding security risks. We then reprocessed the results to define a set of questions that should be answered in the documentation. Finally, we checked the documentation to see whether those questions are answered or not.

As we required several methodical approaches, this chapter is divided into five sections. The first section describes the sampling process. The second section refers to identifying issues from Stack Overflow posts. The third section explains how we checked the threads for security issues. The fourth and last section is about the analysis of the library's documentation.

3.1 Sampling

In JCA, symmetric encryption is implemented using the `Cipher` class. It supports a wide range of symmetric and asymmetric encryption algorithms. To search for suitable threads on Stack Overflow, we first defined a set of queries. We use the `[java]` tag combined with a minimal `Cipher.getInstance()` statement for each symmetric algorithm. This statement must be executed in all encryption scenarios using `Cipher` class.
As some of the symmetric algorithms supported by JCA are not very popular, the corresponding queries returned only a small amount of posts. We decided to exclude those algorithms and focused on the three most popular symmetric encryption algorithms: AES, DESede and DES.

Next, we calculated the sampling size using the sample size calculator by SurveyMonkey. To ensure a confidence level of 95% and a margin of error below 8%, we required 150 posts. Then, we computed sample size per query proportionally to the number of posts a it returns. The result can be found in table 3.1.

Query	**#Posts**	**Sample Size**
`[java] Cipher.getInstance(AES)`	3233	126
`[java] Cipher.getInstance(DESede)`	295	12
`[java]Cipher.getInstance(DES) --DESede`	300	12

Table 3.1: Computation of Sample Sizes

Finally, we selected the threads. We picked half of the sample from the newest[1] threads and the other half from the most popular[2] ones. This approach attempts to balance the ambition to detect present issues with the fact that most programmers first search for answers on Stack Overflow (generating views) before posting a question.
As the aim of the analysis was to reveal issues referring to the implementation of *symmetric encryption* using the *JCA* library, we excluded all posts that did not refer to this scope. Of course it is possible that several issues are discussed in one thread. A thread was included if at least one issue, question, or advise referred to our scope.

[1] based on the creation date of the question
[2] based on the view count

3.2 Analysis of Issues

The goal of the first analysis was to answer the first research question: *What issues do programmers face when implementing symmetric encryption using Java Cryptography Architecture?* We followed the guidelines for qualitative content analysis by Mayring [6]. They are a standard in humanities to evaluate text-based data and are especially well suited for method-integrative approaches that combine qualitative and quantitative elements. They allow to evaluate large amounts of material and quantify individual analysis steps - which is exactly what we need to answer our first research question. They also provide a set of criteria to evaluate the validity and reliability of a methodical approach.
In this section, we will only provide the information to understand the process. You can find the complete processing scheme in the "Anleitung zu Wissenschaftlichen Arbeiten", chapter 6.

We conducted the analysis in three rounds. We first applied *summarizing* to extract the relevant information (issues and questions) from the threads. Then we classified the issues in two rounds.

3.2.1 Summarizing

The goal of this first analysis step was to extract all relevant information and record it such that we did not have to re-read the entire threads during further evaluation. We considered the question post, posts marked as "accepted answers" as well as comments to those posts. If there was not any accepted answer, we considered all posts and comments and tried to display the discussion in our records.
We conducted summarizing first on our own and discussed the results afterwards to create a consistent and more objective list of records. In particular, we eliminated records that did not refer to our scope. As an example, we excluded all issues that referred to the conversion of plain text or the cipher text as data conversion (or character encoding) is not only required in the context of cryptography.

For each thread, we recorded the set of issues and questions that the person writing the question post (*original poster*) was facing. Then we tried to identify the reasons and solutions or answers. We aimed for a short issue description (e.g. an error message or a shortened form of a question) and more precise explanations for the reasons and solutions. To stay as near to the material as possible, we copy-pasted words, sentences or even paragraphs.

3.2.2 Classification

As already mentioned, we classified the threads in two rounds regarding two different kind of aspects. We first classified all records regarding technical aspects and then classified the records regarding requirements the person asking the question was not able to meet. In each round, we assigned at most one category to a record. We also wanted to assign at least one category to each record. So in the end, a record should have one or two assigned categories.

Technical Aspects

In the first round of classification, we focused on the technical aspects of implementing symmetric encryption. As categories, we defined a set of tasks that programmers must take care of when implementing symmetric encryption using JCA. If they implement all tasks correctly, the code compiles and runs without rising an error and leads to the expected result. So if programmers ask questions on Stack Overflow about a technical aspect, they either implemented a task incorrectly or they have a question regarding one of these tasks. In this classification we asked "What implementation step was performed incorrectly causing the error?" or "What implementation step is targeted by the question?".

Identifying a set of main tasks was simple: *Cipher Object Instantiation, Generating Algorithm Parameters, Cipher Object Initialization, Transformation, and Transmitting Algorithm Parameters*. But as we wanted to have more insight, we also defined subcategories. To do so, we consulted the "Java Cryptography Architecture (JCA) Reference Guide" [10] which provides code examples as well as explications regarding the use of the different APIs. To allow an unambiguous category assignment, we also defined the dependencies as categories. As a result, the following technical categories and subcategories were derived:

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

  Additionally, we defined the following subcategories:
  - **Dependency Encryption Mode - Padding**: The encryption modes determines whether padding is required or not. We assigned this category to all issues caused by an inappropriate specification of those two properties.

  - **Cipher Object Instantiation - Other** for issues and questions related to `Cipher` object instantiation but not any of the aforementioned aspects.

- *Generating Algorithm Parameters*: Depending on the specification of the cipher object, it requires different kind of parameters. For encryption, the programmer 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**, e.g. a `GCMParameterSpec` object which contains additional parameters for GCM encryption mode.

- *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. We defined the following subcategories:

  - **Dependency Algorithm - Key**: The algorithm determines what data type the key must be stored in. It also defines the allowed key sizes. We assigned this category to issues caused by passing an inappropriate 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 algorithms block size. As an example, an issue related to passing an IV of the wrong size to the `init(...)` method was assigned to this category.
  - **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 defined the following subcategories:

  - **Key Transmission**: The key must be kept secret.
  - **Transmission of Other Parameters** such as the initialization vector. They can be transmitted along the cipher text as they do not have to remain secret.

- **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 differing configurations.

You can find a list with concrete examples for each category in the appendix.

Requirements

As the first set of categories only targeted the implementation of symmetric encryption, we defined a second set of categories regarding the design of an application. We defined different kind of functional and non-functional requirements as categories. We consulted Sommerville [14] as a theoretical basis. During the analysis we asked ourselves "Which requirements are not met?". Not all requirements defined by Sommerville occurred in our analysis. We only assigned the following categories:

- **Functional Requirements** to issues where a programmer was not able to fulfill a certain requirement or tried to use encryption for an unsuitable use case.
- **Performance** to issues where an implementation was not as time-efficient as required.
- **Space** to code leading to an `OutOfMemoryException`.
- **Reliability** to situations where the implementation crashed frequently.
- **Portability** to implementations that behaved differently on different (Java) plattforms.
- **Interoperability** to issues where a developer was not able to decrypt a cipher text that was produced using another library or vice versa.
- **Security** to implementations containing security risks.

As we analyzed the discussion, we only assigned a category if either the person asking the question complained about not being able to meet a certain requirement or someone gave a hint. We conducted a broader examination of security risks in our second analysis (see chapter 3.3).

3.2.3 Evaluation

We conducted several forms of frequency analysis' on our data. We computed the absolute and relative frequencies for each category and subcategory. Finally, we also took a look at the combination of technical aspects and requirements categories. In our interpretation, we focused on the most occurring categories as these refer to more prevalent issues.
ToDo

3.3 Analysis of Security Risks

The goal of this second analysis was to answer the second research question: *What are common security risks in code and advises shared on Stack Overflow referring to the implementation of symmetric encryption scenarios Java Cryptography Architecture?*
We first defined a set of security rules regarding the implementation of symmetric encryption. Then we reprocessed the original sample checking them for rule violations.

3.3.1 Security Rules

We derived our rules from the rule sets used for CRYLOGGER tool (see Piccolboni *et al.* [13]) and the CrySL based compiler for Java applications by Krüger *et al.* [5]. We only considered the rules that were applicable to symmetric encryption and structured them using the categories from technical aspect's classification (see section 3.2). We also abstracted the context of the rules to facilitate their evaluation. As an example, R-04 of CRYLOGGER says " Don't use the operation mode CBC (client/server scenarios)" (Piccolboni *et al.*, 2020, [13], p.5). We abstracted the context and defined that a programmer should not use CBC at all. We often do not know in what context the original poster wanted to use the code. Additionally, if we think about someone naively copy-pasting code from Stack Overflow, the use of CBC is a potential security risk.
The resulting rules can be found in table 3.2,

3.3.2 Tracking Security Rule Violations

We reprocessed the original sample marking for each rule whether it was violated or not. We only considered the question post, the accepted answer post, and comments to one of these. We also distinguished between "question" and "answer" as well as "code" and "text". We analyzed the four aspects independently and made a list for each one of them:

- **Question Code** to track security risks in code snippets of the question post
- **Question Text** to track security risks in the text of the question post as well as comments to it by the original poster

- **Answer Code** to track security risks in code snippets of the answer post
- **Answer Text** to track security risks in the text of the answer post as well as any comment by another person.

While analyzing the code snippets, we focused on the parts where encryption, decryption, key derivation, IV generation, and key storage were implemented. As an example, if someone is defining a static key in the main method and passes 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 key is passed.
The answers were only analyzed if there was an accepted one.

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 (= constant) 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 (score < 3)
R-03-f	Do not reuse passwords multiple times.
R-04	***Rules for IV / Nonce Generation***
R-04-a	Do not use a static (= constant) 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 Initailization***
R-05	Do not reuse the same key-IV pair.
Rule ID	***Rule - Parameter Transmission***
R-06	Do not use a static (= constant) password for store.

Table 3.2: Security Rules

3.3.3 Evaluation

We conducted a frequency analysis for each list. Our aim was to find out the most prevalent security risks to answer the research question. ToDo

3.4 Analysis of Documentation

The results from the precedent analyses formed the basis to analyze the documentation for JCA. As it spreads over several documents and sources, we only examined the most basic ones: the Java Cryptography Architecture (JCA) Reference Guide [10], the Java Security Standard Algorithm Name Specification [11], and the entire API documentation for `javax.crypto` package starting from the package overview [9]. These three documents are valid for all providers and therefore apply to a wide range of platforms.

To analyze the documentation, we first defined a set of question that should be answered. Afterwards, we sought the answers in the documentation. Our aim was to answer the third research question: *To what extent are those issues*[3] *linked to missing or inadequate documentation?* The questions additionally gave us more insight in what programmers are struggling with. They helped us to answer then first research question more precisely.

3.4.1 Deriving Questions

As an API documentation should support the usage of the API and not educate the programmer, basic questions regarding cryptography should not be answered in it. However, a cryptography API should link reliable sources. We therefore created two lists of prioritized questions: one with ”questions to documentation” and one containing ”general questions”.

We derived the questions from the results of the former analyses. We reprocessed the records from the first analysis and tried to formulate a question for each one. If the question was new, we wrote it down and set its priority to one. If there was already a similar question, we increased its priority by one and sometimes reformulated the question.
The set of questions derived from the first analysis' results covered already a wide range of security related topics. We therefore did not have to add more questions based on the second analysis' findings. However, we adapted the priorities of those questions, setting them to the actual number of threads that included a related security risk.

[3] obstacles in implementation, security risks in code

3.4.2 Consulting Documentation

For each question on the list, we then tried to find an answer in the documentation. Depending on the question, we checked the different sources in another order. We aimed to find answers as efficiently as possible.
For ”questions to documentation”, we typically started with the reference guide to find general explanations and then consulted the related parts of the API documentation. For ”general questions” we started in the standard algorithm name specification. Of course, we knew the documentation better after answering a set of questions and therefore optimized our search strategies.
Once we found an answer to the question, we recorded its source as well as some remarks regarding documentation quality.

3.4.3 Evaluation

We first wanted to find out, what percentage of questions remained unanswered. This gave us a starting point to answer the research question. If most questions are answered in a clear way, we cannot blame the documentation for the issues programmers face during implementation. We also cannot blame it for unsafe code if it contains enough hints and advises regarding security.
In the further evaluation we focused on the unanswered questions and our comments regarding documentation quality to make suggestions regarding the improvement of the documentation.
We also had a closer look at the questions as they gave us more insight regarding the first research question.

4 Results and Interpretation

In this chapter, we discuss the results of our three analyses and their interpretation. There is a section for each of the three conducted analyses: The first section shows the issues and the second section the security risks we found in our sample data. The third section explains how the previous findings are linked to the documentation.

4.1 Issues

We recorded a total of 219 issues. We classified 197 records (90%) regarding technical aspects and 76 (35%) regarding requirements. 62 records were classified twice. There was one thread that did not provide enough information to be classified.

As shown in figure 4.1, the most common main categories in the first categorization round were *Generation of Algorithm Parameters* and *Cipher Object Instantiation*. The most common subcategory was *Key Derivation* with a total of 36 issues (16.4%). This is not surprising as previous work has already shown that programmers struggle with key handling. The set of issues referring to the key made up more than $\frac{1}{5}$ of all issues. This was only topped by the set of issues related to the various dependencies (27 %).
Within the *Cipher Object Instantiation* category, most issues referred to the encryption mode (18), padding (11) or the dependency between these properties (11).
The third most common main category was *Cipher Object Initialization*. Most issues were assigned to its *"Other"* subcategory. They often refer to the default behavior of the

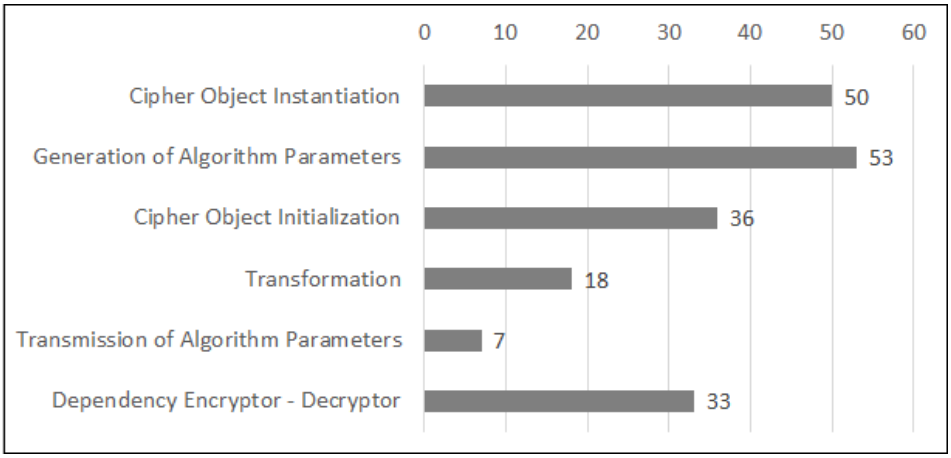

Figure 4.1: Number of Issues Assigned to a Technical Aspects Main Category ($N = 219$)

`init(...)` method if not all required properties were specified.
The fourth most common main category was *Dependency Encryptor - Decryptor*. The high prevalence of this category and the other dependency related subcategories implies that people 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.
The other main categories and subcategories were not assigned that often. You can find the complete list of categories and their frequencies of assignment in table 4.1.

	Implementation		
Main Category	**Subcategory**	**%**	**#**
(Generation of Algorithm Parameters)	Key Derivation	16.44%	36
(Dependency Encryptor - Decryptor)	Dependency Encryptor - Decryptor	15.07%	33
(Cipher Object Initialization)	Cipher Object Initialization - Other	9.13%	20
(Cipher Object Instantiation)	Encryption Mode	8.22%	18
(Transformation)	Transformation	8.22%	18
(Generation of Algorithm Parameters)	IV / Nonce Generation	6.39%	14
(Cipher Object Instantiation)	Padding	5.02%	11
(Cipher Object Instantiation)	Dependency Encryption Mode - Padding	5.02%	11
(Cipher Object Initialization)	Dependency Algorithm - Key	3.65%	8
(Cipher Object Initialization)	Dependency Algorithm/Encryption Mode - IV	3.65%	8
(Cipher Object Instantiation)	Algorithm	3.20%	7
(Transmission of Algorithm Parameters)	Key Transmission	2.28%	5
(Cipher Object Instantiation)	Cipher Object Instantiation - Other	1.37%	3
(Generation of Algorithm Parameters)	Generation of Other Algorithm Parameters	1.37%	3
(Transmission of Algorithm Parameters)	Transmission of Other Algorithm Parameters	0.91%	2

Table 4.1: Number of Issues Assigned to a Technical Aspect Subcategory ($N = 219$)

Within the categories referring to requirements, *Security* was the dominating category (46 records - 21%). This seems natural as this is what cryptography is all about. Most issues were not mentioned in the question post. They refer to security hints by people commenting or answering. Yet, there is massive under-reporting as our results from the security related analysis show (see section 4.2).
The other categories occurred much less often as you can see in figure 4.2

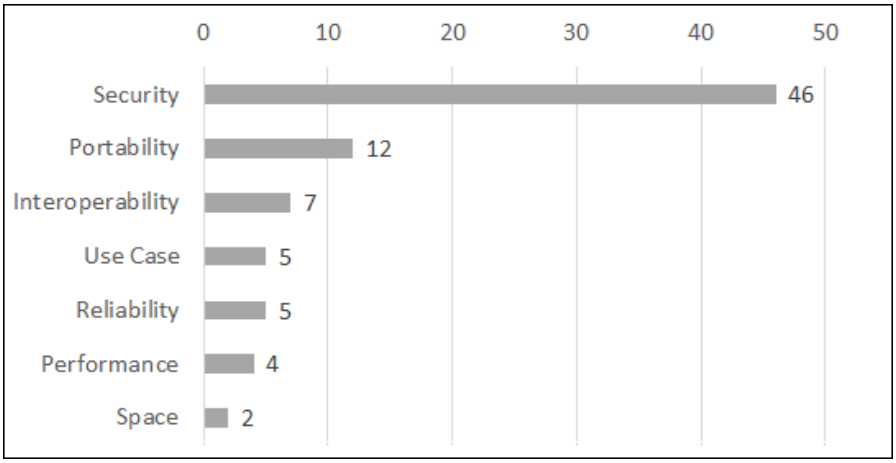

Figure 4.2: Number of Issues Assigned to a Requirement Category ($N = 219$)

We observed the highest overlapping of categories for *Generation of Algorithm Parameters* and *Security* as well as *Cipher Object Instantiation* and *Security*. The heat map in figure 4.3 shows the relative overlapping of categories: 21% of all posts classified as *Generation of Algorithm Parameters* or *Security* issue were assigned to both categories. As we had less data for the other requirements categories, it is not surprising that we found much less overlapping for those.

	Performance	Reliability	Portability	Interoperability	Security
Cipher Object Instantiation	0.06	0.00	0.08	0.04	0.16
Generation of Algorithm Parameters	0.00	0.00	0.05	0.05	0.21
Cipher Object Initialization	0.00	0.02	0.04	0.02	0.05
Transformation	0.00	0.09	0.00	0.00	0.00

Figure 4.3: Relative Overlapping of Technical Aspects and Requirements Categories

4.2 Security Risks

We manually checked 150 question, 84 answer posts, and related comments using our rule set. We found a total of 331 security risks. Most of them (249 - 75%) stem from code snippets in question posts. The text of question posts included only 38 security risks. But we observed that the questions often did not contain much text.
We found 35 rule violations in answer related code snippets and 9 in related text. We observed that some answers just fixed the functionality without improving its security. The resulting code therefore ”inherited” the security risks from the question. Another common observation was that people correctly gave the advise that ECB was not considered save. However, they suggested to use CBC which must not be used in client-server scenarios. Such advise is therefore not safe, especially if we do not know in what kind of application the code section is used.

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 (= constant) 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 (score < 3)	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 (= constant) 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 (= constant) password for store.	3	0	0	0	**3**

Table 4.2: Rule Violations per Check List

As we discovered most security risks in the question code, we focused on the evaluation of that check list. The average question post contained 1.66 security risks in its code snippets. We observed that the average for the most popular (and therefore older) posts (1.91) is slightly higher than the one for the newest posts (1.43). 24 question posts did not contain any security risk in their code snippets.
The most often violated rule was *R-02: Do not use ECB or CBC encryption mode*. In more than 75% of question posts, the original poster used one of these unsafe block cipher modes. It is followed by the presence of constant keys (*R-03-a*), unsafe password based key derivation procedures (*R-03-b to R-03-f*, 29 recorded rule violations), and static initialization vectors (*R-04-a*).
Of course, the number of posts showing violations of *R-01* is strictly related to the

sample: 24 of our threads referred to DES or TripleDES encryption. There were some code snippets in which the same key-IV pair was reused for several encryption operations. This was related to using a constant key and a constant IV in most cases.

4.3 Documentation

We derived a total of 64 questions from the first analysis: 44 referring to documentation and 22 more general ones. After checking the documentation, 10 questions remained unanswered and for 13 questions, we considered the answer as incomplete, unclear, or even misleading.

4.3.1 Questions

5 Conclusion and Future Work

RQ 1: *What issues do programmers face when implementing symmetric encryption using Java Cryptography Architecture?* Our study has showed that... ToDo

We would like to complete our answer to the first research question with an unintended observation. As the scope of our study was quiet limited, we excluded all threads that did not target it. To collect our sample, we had to exclude 296 threads. This means that for each thread that (partly) belongs to our scope, there are two threads that do not. Of those excluded threads, 138 (ca. 47%) do either not refer to cryptography at all or they did not refer to the Java Cryptography Architecture library. Some of the latter ones refer to the BouncyCastle library but the majority does not. Previous research implied that a lot of programmers implementing cryptography scenarios lacked domain knowledge (about cryptography). We would like to extent this assumption: Programmers do also lack knowledge about computer science in general. As an example, character encoding is a big obstacle as the plain text must be fed to encryption as a byte array. We must be aware that cryptography scenarios are implemented by programmers of all kind of backgrounds.

RQ 2: *What are common security risks in code and advises shared on Stack Overflow referring to the implementation of symmetric encryption scenarios Java Cryptography Architecture?* ToDo

RQ 3: *To what extent are those issues linked to missing or inadequate documentation?* Our study has showed that... ToDo

5.1 Limitations and Future Work

As this project followed a qualitative approach performing an in-depth analysis, it is reasonable to have a limited scope. Future work must extend it by examining more use cases and libraries.

When it comes to the methodical approach, one threat to validity is that the population of Stack Overflow threads matching the criteria cannot be described exactly. There is no guarantee that the used queries returned all posts referring to symmetric encryption using the targeted libraries. However, this is not possible with a finite set of queries as you cannot know which posts you did not find. This circumstance implies that the results must be verified with further investigations.
Also, the sample cannot be considered as representative for the posts returned by the defined queries as a lot of posts (ca. $\frac{2}{3}$) were excluded because they did not match the predefined criteria. The high ratio of posts not referring to cryptography and not referring to the library implies that people do not only struggle with cryptography when implementing a cryptography scenario. This observation reveals new research areas and questions. But it also complicates the design of cryptography APIs as it should satisfy a security expert equally as a ”newbie”.

5.2 Conclusion

6 Anleitung zu wissenschaftlichen Arbeiten

This chapter describes the methodical approach of qualitative content analysis according Mayring [6]. Each section refers to a work step of the processing scheme which is first explained generally and then applied to the derivation of issues from Stack Overflow threads, see chapter 3.2.
Please note that the aim of this chapter is to specify the processing scheme more precisely. All kind of justification can be found in the methodology section of this thesis.

6.1 Presenting the Sample

Defining and presenting the sample is the first step in qualitative content analysis: We describe the population, the context of its creation and its formal characteristics as well as the sampling process.

6.1.1 Population

Stack Overflow is one of the most popular online forums for programmers. According to the developer survey from 2020, about 50 million people visit the platform per month. Approximately 50% of them are either professional developers or university-level students [12]. As we only want to examine threads that refer to the implementation of symmetric encryption using the Java Cryptography Architecture library, we define our population as the set of posts returned by at least one of the following queries:

- `[java] Cipher.getInstance("AES")`
- `[java] Cipher.getInstance("DESede")`
- `[java] Cipher.getInstance("DES") --DESede`

We do not know a lot about the context of our population. Especially, we do not know anything about the people writing the posts and comments. Based on the findings by Nadi *et al.* [8], we assume that the people asking questions are professional developers (industrial or freelance), researchers or students who must implement a symmetric encryption scenario either for work or education. However, their backgrounds regarding experience with Java as a programming language or cryptography are very heterogeneous. In Nadi *et al.*'s survey, the test persons declared that they "spend at least several hours reading through online resources" (2016, [8], p. 938) before asking on Stack Overflow.

All material is available online on Stack Overflow. It reflects a conversation between one or several users. For each post, we can see the username of the author as well as the date and time of its creation. Some posts have been edited after their creation. In that case, we can see the username and the date of the last edit. However, we cannot reproduce what has been changed. Sometimes, persons also delete their posts or comments. Such posts are not visible anymore.

6.1.2 Sampling Process

Based on the total number of posts returned by the queries (3828) we calculated that 150 threads should be enough to satisfy our requirements of a confidence level of at least 95 % and a margin of error below 10%. Next, we computed the number of threads per query proportionally to the number of posts it returned. Finally, we selected the threads for our sample. We picked half of the sample from the "newest" threads which was determined by the creation date of the question post. We took the other half from the most popular ones based on the view count of the question. Also, we only selected threads that addressed at least one issue within our scope. Other posts were excluded for at least one of the following reasons: ToDo: C# Beispiele austauschen

- **too general**: posts referring to cryptographic concepts or cyber security in general rather than the targeted API
  Example: When will C# AES algorithm be FIPS compliant?
- **does not refer to cryptography**: issues occurring in a non-cryptographic context, i.e. establishing a network connection or file access
  Example: Syntax error

- **does not refer to symmetric encryption**: posts referring to other cryptographic concepts such as asymmetric encryption or hashing
  Example: RSA decryption (fails)

- **does not refer to JCA**: issues occurring during a symmetric encryption scenario but which are not due to Java Cryptography Architecture. Such issues can refer to another library (e.g. BouncyCastle) or to some other aspect such as character encoding.
  Example: 256bit AES/CBC/PKCS5Padding with Bouncy Castle

- **does not refer to targeted algorithm**: posts referring to other algorithms than the targeted ones
  Example: how to use password with `PBEKeySpec` *hash*

- **looking for an equivalent** or **interoperability issue**: looking for equivalents / counterparts in different programming languages
  Example: Change Rijndael decryption logic from C# to Python

- **negative votes** or **closed**: posts of poor quality

- **academic**: posts with a different focus than obstacles when using the API

- **duplicate**: Duplicates are often left unanswered (or only answered with the reference to a similar post). For posts belonging to the ”most popular” category, we included a duplicate if it had more views than the original.

6.2 Specific Research Questions

The second step of qualitative content analysis is to describe precisely what the aim of the interpretation is. We must provide a set of well defined research questions that are theoretically differentiated.

The aim of our first analysis was to find an answer to the first research question of the thesis: *What issues do programmers face when implementing symmetric encryption using Java Cryptography Architecture?* A (piece of) software must meet various requirements. ToDo: Consult Sommerville

6.3 Processing Scheme for Interpretation

Summarizing: The coding unit of this first round of analysis was a single word (e.g. an error message). The context unit was an entire post or a comment.

6.4 Evaluation of Specific Quality Criteria

Bibliography

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

[2] David Alonso-Ríos, Ana Vázquez-García, Eduardo Mosqueira-Rey, and Vicente Moret-Bonillo. Usability: a critical analysis and a taxonomy. *International journal of human-computer interaction*, 26(1):53–74, 2010.

[3] William J. Buchanan. *Cryptography*. River Publishers, 2017.

[4] Matthew Green and Matthew Smith. Developers are not the enemy! the need for usable security apis. *IEEE security & privacy*, 14(5):40–46, 2016.

[5] 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*, pages 1–1, 2019.

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

[7] Eduardo Mosqueira-Rey, David Alonso-Ríos, Vicente Moret-Bonillo, Isaac Fernández-Varela, and Diego Álvarez Estévez. A systematic approach to api usability: Taxonomy-derived criteria and a case study. *Information and Software Technology*, 97:46–63, 2018.

[8] 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.

[9] Oracle. Class cipher, 2021.

[10] Oracle. Java cryptography architecture (jca) reference guide, 2021.

[11] Oracle. Java security standard algorithm names specification, 2021.

[12] Stack Oveflow. 2020 developer survey, 2020.

[13] Luca Piccolboni, Giuseppe Di Guglielmo, Luca P Carloni, and Simha Sethumadhavan. Crylogger: Detecting crypto misuses dynamically. 2020.

[14] Ian Sommervile. *Software Engineering*. Pearson, 9th edition, 2011.

7 Acknowledgments

8 Appendix

A Sample

B Classification

C Security Analysis

D Questions

E Answers