UNIVERSITÄT PADERBORN
Die Universität der Informationsgesellschaft
Faculty for Computer Science, Electrical Engineering and Mathematics
Department of Computer Science
Research Group Software Engineering

Bachelor's Thesis Proposal

Submitted to the Software Engineering Research Group

Automatically Applying Recommendations From Regulators to Cryptography Specification Codes

by
Daniel Schwietert
Matr.-Nr. 7072413

Thesis Supervisor:
Prof. Dr. Eric Bodden

Paderborn, November 24, 2020

1 Introduction

Cryptographic standards and requirements are constantly developing. Algorithms become deprecated, their parameters get tuned and changed or new algorithms are developed. Therefore the standards and recommendations for algorithms, which are considered secure, change. Recommendations are issued regularly by regulators such as Federal Office for Information Security (BSI) [1] or National Institute of Standards and Technology (NIST) [2] on an annual basis.

Often software developers are not security experts and do not fully know these recommendations and how to use cryptography correctly [3]. It is hard for developers to stay up to date and to understand the recommendations, as they have to read the technical guidelines and have to have a good understanding of the topic.

Developers do not have to write their own cryptography algorithms, because cryptography frameworks provide access to implementations of cryptographic algorithms. On Java the standard cryptography framework is the Java Cryptography Architecture (JCA) [4]. Using this framework applications can request security services, which are implemented by independent Cryptographic Service Providers (CSP). These providers, such as BouncyCastle [5], are written by security experts and implement several cryptographic algorithms securely. Although some of these frameworks choose insecure deprecated default configurations, e. g. ECB mode for block ciphers, when no operation mode is explicitly configured. This is one main reason why developer write insecure code [6] and can be seen in Listing 1.1. That Java code example encrypts a text with DES using the JCA. Therefore, first a key is generated in lines 3-5. Here a KeyGenerator is used, which is specified for DES and initialized with a key size of 56 . Then a Cipher object is created, specifying the algorithm, and initialized with the secret and encryption mode used. As no mode of operation and padding is specified, the provider's default is chosen, which is the ECB mode and PKCS5Padding for the standard CSP. The ECB mode is not considered secure [1].

```
1   String plainText = "Hello World!";
2
3   KeyGenerator keyGenerator = KeyGenerator.getInstance("DES");
4   keyGenerator.init(56);
5   SecretKey secret = keyGenerator.generateKey();
6
7   Cipher cipher = Cipher.getInstance("DES");
8   cipher.init(Cipher.ENCRYPT_MODE, secret);
9
10  byte[] cipherText = cipher.doFinal(plainText.getBytes());
```

Listing 1.1: Example of DES encryption with the JCA

To avoid this and to help developers use these cryptography frameworks securely and write secure code, static analysis tools, such as CogniCrypt [7], have been developed. Those tools check among other things the use of proper algorithms and secure parameters. CogniCrypt uses the specification language CrySL [8]. This language enables cryptography experts to specify the secure use of cryptographic frameworks. Tools like CogniCrypt can then compile the specifications of CrySL rule sets into their static analysis. The CrySL language uses white listing to define secure uses, so each secure use is listed explicitly. A CrySL rule is specified for each cryptography Java class. Each CrySL rule has different sections. We only look at the CONSTRAINTS section of a CrySL rule, in which the allowed algorithms and parameters, which can be used securely, are defined. An example for this CONSTRAINTS section is given in 1.2. It restricts the Java Cipher object described in Listing 1.1. All possibilities of an algorithm, mode of operation and padding, which are considered secure, are defined here in the CONSTRAINTS section. In our example we have used the algorithm (alg) DES, mode ECB and padding (pad) PKCS5Padding. The ECB mode is not listed in the CrySL rule and therefore not considered secure by our CrySL rule. Tools, which implement this rule, will therefore warn the programmer.

```
1  CONSTRAINTS
2      alg(transformation) in {"AES", "DES"}
3          && mode(transformation) in {"CBC", "PCBC"}
4          => pad(transformation) in {"PKCS5Padding", "ISO10126Padding"};
```

Listing 1.2: Example of the CONSTRAINTS section of a CrySL rule for restricting javax.crypto.Cipher

As cryptographic frameworks [9] and recommendations change, CrySL rules have to be updated regularly. In our example, not only the mode of operation is insecure, but also the encryption with DES is outdated [1] and has to be removed from the CrySL rule, so Line 2 in Listing 1.2 has to change to `alg(transformation)in {"AES"}`. Thus tools can warn the users using this setup.

To keep these CrySL rule sets up to date, we will develop a service, which can automatically fetch the latest recommendations from a regulator. Using these recommendations we update on demand an existing set of CrySL rules to match the specifications of the regulator. This ensures that the latest recommendations from regulators are followed and that the rule set is maintained promptly. Thus this propagates up-to-date recommendations for the secure usage of cryptographic frameworks to the developer community.

2

Design and Implementation

In this thesis, we design a service application, which incorporates recommendations from regulators and uses these to update CrySL rule sets automatically. This service application runs as a server and regularly polls specifications from regulators, checks for new regulations and updates specified rule sets if required.

To do this and separate the process of obtaining the information from the program logic that processes the regulations, we use the regulator design pattern as suggested by Indela et al.[10]. They developed it to incorporate certificate validation into program code. The regulator pattern is a behavioral design pattern to incorporate information from external sources.

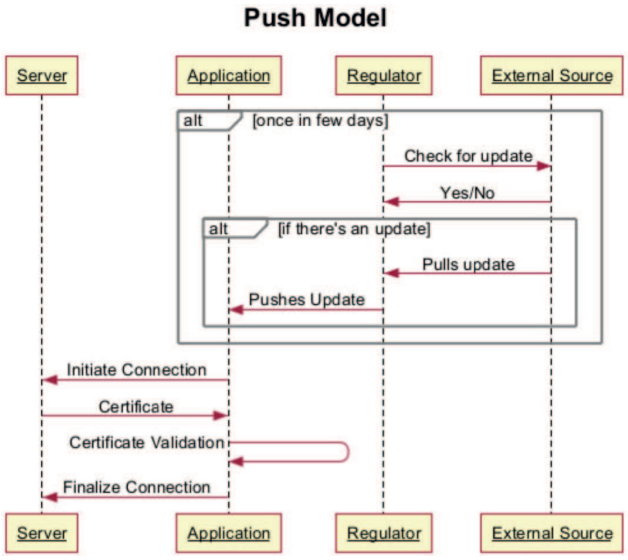

Figure 2.1: Sequence diagram showing the regulator pattern (Push Model) [10]

Indela et al.have developed multiple models for the regulator pattern. We describe the Push Model here. Figure 2.1 shows a sequence of this model with four participants: Server, Application, Regulator, External Source. The information retrieval is done by a Regulator object, which regularly pulls the data from External Sources and checks for changes to previously loaded data. If there have been any changes, the regulator pulls the update and pushes it to the application. Therefore the application has the latest data available and can use this to validate e. g. certificates obtained from a server.

We adopt this regulator design pattern, but instead of validating a certificate (2.1), we use it to make a CrySL rule set compliant with the regulators recommendations.

In our case we want to incorporate data from (multiple) regulators. Currently regulators only provide these recommendations in form of PDF files. These are tedious to read and translate to CrySL by hand. Automating this process presents challenges as well. The goal is to establish an interface, which provides machine-readable lists of cryptographic algorithms, modes of operation and key sizes that are recommended and considered secure. The regulators have to provide this information as a REST API.

Our service regularly polls recommendations from specified regulators and compares them to previously loaded regulations. If any changes are detected, the local database is updated with the new data and the main application is triggered. In the main application the CrySL rule sets are updated to match specifications from defined regulators. If multiple regulators should be considered, the recommendations from the regulators are merged by keeping only recommendations all regulators agree on. To load the CrySL rule set, we build an adapter, which can load CrySL specification language rule sets from a given git repository.

```
1    CONSTRAINTS
2        alg(transformation) in {"AES"}
3            && mode(transformation) in {"CBC", "PCBC"}
4            => pad(transformation) in {"PKCS5Padding", "ISO10126Padding"};
5        alg(transformation) in {"AES"}
6            && mode(transformation) in {"GCM", "CTR", "CTS", "CFB", "OFB"}
7            => pad(transformation) in {"NoPadding"};
```

Listing 2.1: Example of the CONSTRAINTS section of a CrySL rule for restricting AES algorithm in javax.crypto.Cipher

As said, we look at the CONSTRAINTS section of each CrySL rule, in which allowed algorithms and parameters are listed explicitly. Listing 2.1 shows the updated version of the example 1.2 and extends to all possible modes for AES. In this example we restricted the allowed modes and paddings combinations for AES Ciphers to only secure combinations. Here we use conditional constraints to define the allowed paddings for sets of AES modes. So for example, if AES with GCM mode is chosen, only NoPadding can be used to have an secure combination. Our application service searches for specified algorithms and parameters in this section and compares them against the new collected regulations from the regulators. If there are algorithms and key lengths, which are no longer recommend, or new ones missing in the CrySL rules, we change the rules accordingly.

In case the rules have been changed, we export the changes in a commit and recommend them as a pull request (PR) to the repository of the rule set. The maintainers of the rule set can then review and merge the changes. The service tracks the changes made, so we do not resubmit the same changes.

3 Evaluation

The goal of the developed application is to get the latest security recommendations from the regulators and integrate them in the CrySL rule set automatically. This helps maintaining the rule set and makes sure the recommendations of regulators are followed and quickly integrated. In addition to that the tools, which use these rule sets for their code analysis, can provide the latest standards to the programmer community.

To evaluate that our proposed service provides the required features we aim to answer the following research questions:

RQ1: Are the changes of the recommendations from the regulators correctly implement in the CrySL rules by the proposed service?

RQ2: How long does our service need to update a CrySL rule after it has detected a change in recommendations?

To test the service in depth, we build a test regulator API, which provides the same data as the regulators. With this we can define our own regulations and make selected changes to the recommendations. For the evaluation we will use this test regulator API, which first provides the recommendations of the BSI. We will run our service on the JCA rule set [11] and with the test regulator API as the source for recommendations. These recommendations will be changed during the course of the evaluation. Changes to the recommendations are, for example changing the key strength or removing an algorithm.

In order to answer RQ1 we will mark the changes our service proposes to the CrySL rule set as correct or incorrect. A missing change in the rule set will be treated as an incorrect change. We will calculate a success rate by simulating multiple of these changes.

To answer RQ2 we will measure the average time in total it takes until a change from the regulator is adopted to the CrySL rule set and committed as a PR. For each change, we will measure how long the process of detecting the change, implementing it into the rule set and submitting it as a PR takes. So we can calculate the expected value and standard deviation of this.

4 Thesis Structure

1. Introduction
2. Motivation
3. Goals
4. Related Work
5. Approach
   (a) Service Concept
   (b) Regulator Design Pattern
   (c) Interfaces
6. Implementation
   (a) Regulator API
   (b) Git Adapter
   (c) CrySL Parser
   (d) Regulator
   (e) PR Adapter
7. Evaluation
   (a) Correctness
   (b) Time
8. Conclusion
9. Future Work
10. Literature

5 Time Schedule

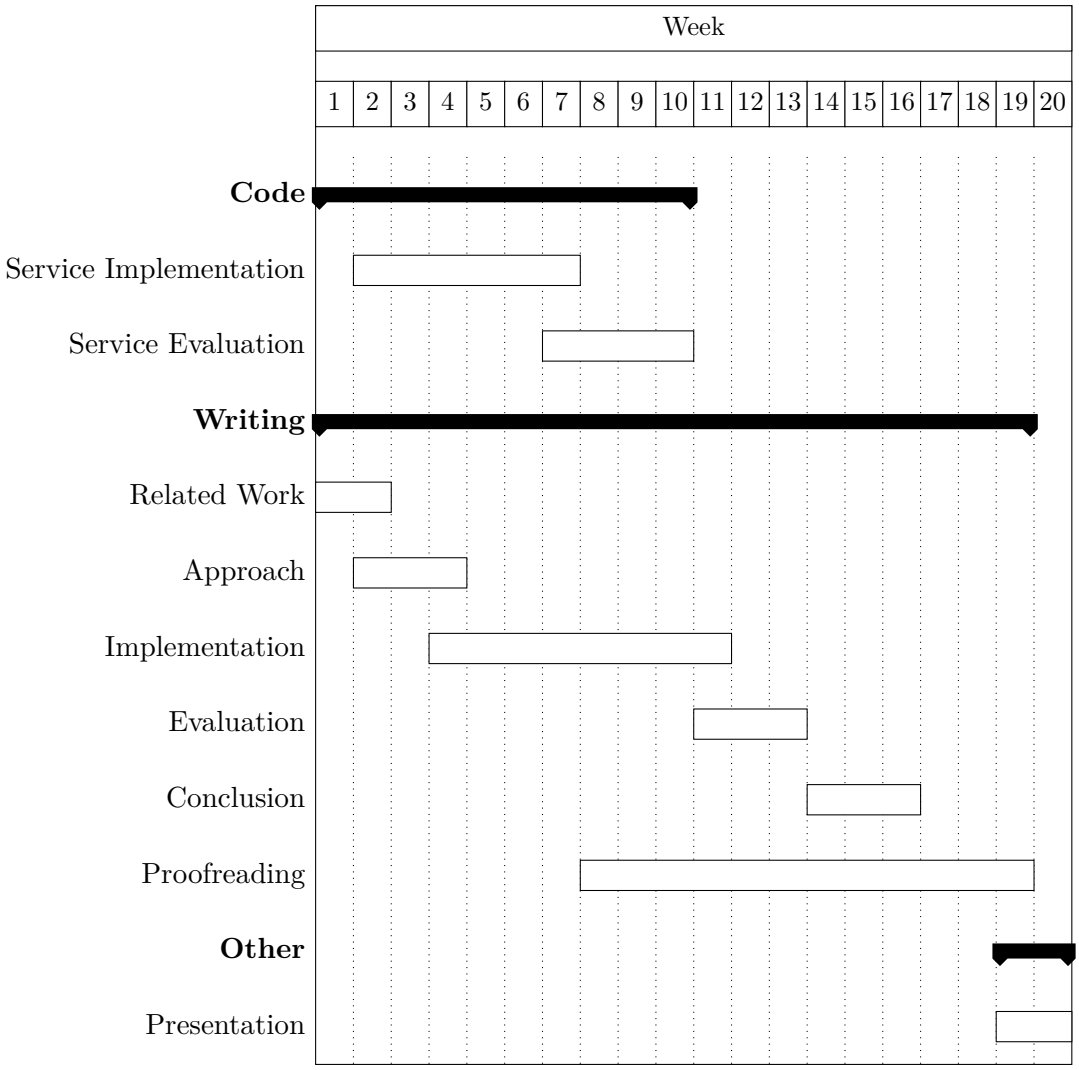

Figure 5.1: Sketch of the time schedule for the work on the thesis

Bibliography

[1] BSI. (2020) Tr-02102-1 cryptographic mechanisms: Recommendations and key lengths. [Online]. Available: https://www.bsi.bund.de/SharedDocs/Downloads/EN/BSI/Publications/TechGuidelines/TG02102/BSI-TR-02102-1.html

[2] E. Barker and A. Roginsky, "Transitions: Recommendation for transitioning the use of cryptographic algorithms and key lengths," *NIST Special Publication*, vol. 800, p. 131A Revision 2, 2019.

[3] D. Lazar, H. Chen, X. Wang, and N. Zeldovich, "Why does cryptographic software fail? a case study and open problems," in *Proceedings of 5th Asia-Pacific Workshop on Systems*, 2014, pp. 1–7.

[4] O. Inc. (2020) Java cryptography architecture (jca) reference guide. [Online]. Available: https://docs.oracle.com/javase/10/security/java-cryptography-architecture-jca-reference-guide.htm

[5] L. of the Bouncy Castle Inc. (2020) Bouncy castle crypto apis. [Online]. Available: https://www.bouncycastle.org/java.html

[6] M. Egele, D. Brumley, Y. Fratantonio, and C. Kruegel, "An empirical study of cryptographic misuse in android applications," in *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, 2013, pp. 73–84.

[7] S. Krüger, S. Nadi, M. Reif, K. Ali, M. Mezini, E. Bodden, F. Göpfert, F. Günther, C. Weinert, D. Demmler *et al.*, "Cognicrypt: supporting developers in using cryptography," in *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2017, pp. 931–936.

[8] S. Krüger, J. Späth, K. Ali, E. Bodden, and M. Mezini, "Crysl: An extensible approach to validating the correct usage of cryptographic apis," *IEEE Transactions on Software Engineering*, 2019.

[9] Oracle. (2020) Oracle jre and jdk cryptographic roadmap. [Online]. Available: https://www.java.com/en/jre-jdk-cryptoroadmap.html

[10] S. Indela, M. Kulkarni, K. Nayak, and T. Dumitraş, "Toward semantic cryptography apis," in *2016 IEEE Cybersecurity Development (SecDev)*. IEEE, 2016, pp. 9–14.

[11] C. TUD. (2020). [Online]. Available: https://github.com/CROSSINGTUD/Crypto-API-Rules