

Format Preserving Encryption

Bachelor Thesis

Degree programme:	Bachelor of Science in Computer Science
Author:	Matthias Liechti, Rizja Schmid
Thesis advisor:	Prof. Dr. Reto Koenig
Expert:	Stefan Berner
Date:	12.06.2015

Management Summary

This thesis aims to give a theoretical as well as practical overview of an emerging issue in the field of IT security named Format Preserving Encryption (FPE).

Although FPE is not new, it is relatively unknown. It is used in the full-disk encryption and some other areas. Nevertheless, it is to this day even unknown to many cryptographers. Another issue that is on everyone's lips is the Internet of Things (IoT). IoT offers a whole new scope for FPE and could give it possibly a further boost.

Format Preserving Encryption is - as the name says - an encryption in which the format of the encrypted data is maintained. When a plaintext is encrypted with FPE, the ciphertext then has the same format again. As illustrated for example on the cover page: If we encrypt the owner and the number of a credit card with AES we get an unrecognizable string. If we use FPE instead, we might get for example Paul Miller and the number 4000 0838 7507 2846. The advantage is that for man and/or machine nothing changes. The encryption is therefore not noticed without analysis of the data. The advantage can also become a disadvantage. An attacker has with the format of the ciphertext already information about the plaintext.

This thesis starts with an introduction to the Format Preserving Encryption. In doing so, different variants of FPE are shown. In a next step, a Java library is explained and documented, in which we have implemented some of these FPE variants. This library is designed to enable programmers to use FPE without the need for detailed knowledge about the functionality. Then we explain by means of a tutorial and step by step with a concrete and simple example, how a subsequent integration of FPE could look like. In a final part the integration into a more complex and already widely used application is shown, an Android app called OwnTracks.

With this combination of theoretical and practical information a broad basic knowledge should be provided on the topic, which then can serve as a basis on how FPE can be used and whether a use is reasonable.

Contents

Management Summary	2
Contents	3
1 Introduction	5
2 Project Management	6
2.1 Organization	6
2.2 Resources	6
2.3 Project Goals	7
2.4 Milestones	8
2.5 Use Case and Test Case	8
2.6 Timetable	9
2.7 Communication	11
2.8 Preliminary Work	11
2.9 Project Reflection	12
3 Introduction to Format Preserving Encryption	15
3.1 Cryptography	15
3.2 Format Preserving Encryption	16
3.3 Areas of Application	18
4 FPE Schemes	19
4.1 Tiny Space FPE Schemes	19
4.2 Small Space FPE Schemes	21
4.3 Large-Space FPE Schemes	24
4.4 Cycle Walking	27
4.5 Rank-Then-Encipher	28
5 FPE-Library	31
5.1 Class Overview	31
5.2 Classes	34
5.3 External Libraries	39
5.4 Juridical Aspects	40
5.1 Development Potentialities	40
5.2 Performance	41
5.3 Security	43
6 Tutorial – Apply FPE in an Existing Context	44
6.1 Use Case	44
6.2 Current Application	44

6.3 Analysis	45
6.4 Application extended with FPE	48
7 Demo App	52
7.1 OwnTracks	52
7.2 MQTT	53
7.3 Lack of Security	54
7.4 Implementation	54
7.5 User Manual	59
8 Summary	62
8.1 Project Management:	62
8.2 Introduction to Format Preserving Encryption	62
8.3 FPE Schemes	62
8.4 FPE Library	63
8.5 Tutorial	63
8.6 Demo app	63
8.7 Conclusion	63
9 List of illustrations	64
10 Contents of the table	65
11 Glossary	66
12 Bibliography	67
12.1 Papers	67
12.2 Websites / Blogs	69
12.3 Videos	71
12.4 Implementations of FPE	71
12.5 Applications which use FPE	72
13 Appendix	73
13.1 Definition of Bachelor Thesis Project	73
13.2 Tiny-space FPE schemes	74
13.3 Small-space FPE schemes	75
13.4 Large-space FPE schemes	76
13.5 Prefix cipher	77
13.6 Permutation numbering	77
13.7 OwnTracks Helper Class FormatPreservingEncryption	78

1 Introduction

Time and again one can hear of cyber-attacks against large companies. Often times these companies have a large budget for the IT security and invest considerably in security precautions. But it is well known that a chain is only as strong as its weakest link. It is much easier for the attacker to find the weakest link than it is for the company to secure the whole chain.

The American bank JPMorgan Chase is a recent case which caused a worldwide sensation - According to BBC News criminals were able to capture the address data of 76 million private customers and seven million business customers.[49] In retrospect, one wonders whether this could not have been avoided.

In the case of JPMorgan, the attacker managed to get administrator rights due to an inoperative two-factor authentication and was able to tap various user data as a consequence. Presumably the data was copied directly from the database. Most likely the webserver and web application were well protected, but the data was available unencrypted in the database and could be read directly out of it with the appropriate privileges. This problem is quite common, as it is often considered legitimate to store data unencrypted when the database is not visible from the outside and the attack vector thus believed to be small.

Whether the encryption of the database would have prevented the raid of the hacker can hardly be answered by an outsider. But what can be discussed are the consequences which subsequent equipment with encryption functionalities might entail. The persistence layer would have to be extended to encrypt the data for the application and further the data would have to be encrypted. The encryption might also entail a redesign of the database scheme because the encrypted output is binary data. The complex changes to the database scheme and the downtime during the encryption probably deter companies from choosing this solution. Instead they might think twice about this step and look for a better way.

Could the impact of such an intervention not be minimized? There is a little known cryptographic application which could – amongst other possibilities – help in this scenario. It is named Format Preserving Encryption (FPE). Format Preserving Encryption – as the name suggests – makes it possible to preserve the format of an encryption. Meaning the ciphertext looks like the original plaintext. In the above scenario, this would have had the advantage that one has to make little to no changes on the database scheme and that the encryption could be integrated almost on the fly.

Because Format Preserving Encryption has so much unheeded potential, it is a very interesting topic to write our bachelor thesis about. Our goal is to make this cryptographic area more tempting for computer scientists or potential users. But before we hurl ourselves into Format Preserving Encryption, let us first have a look at our project management for this bachelor thesis.

2 Project Management

2.1 Organization

Name / Company	Role	Responsibilities
Prof. Dr. Reto Koenig <i>Bern University of Applied Sciences</i>	Advisor	Principal point of contact and defines course of action. Assessment based on scope of work and supplied deliverables.
Stefan Berner <i>Diso AG</i>	Expert	Assessment based on scope of work and supplied results plus two meetings.
Matthias Liechti	Student	Independent project management. Develop tasks in accordance with requirements. Communication with advisor and expert.
Rizja Schmid	Student	

Table 1: Involved persons and their roles

2.2 Resources

Resource Requirements	Costs
Personnel costs	We used 780 hours in total. Because this is a school project and the hours were not paid, no costs incurred.
Non-personnel costs	No non-personnel costs incurred. No devices or software licenses had to be bought for this project.
Costs of education	No education cost incurred.
Service costs	No service cost incurred.

Table 2: Resource requirements

2.3 Project Goals

From the definition of the bachelor thesis, the following points should have been achieved at the end of the project:

- Providing a library which easily allows applying FPE in an existing context.
- Providing a detailed tutorial on how to apply FPE in an existing context.
- Providing one implementation of an application that has been enriched by FPE.

The original assignment is listed in appendix 13.1.

Based on these abstract requirements, we derived the following concrete goals for our work.

Goals marked as optional will only be implemented if time allows.

2.3.1 Goals for the Java FPE-Library

Goal	Mandatory	Optional	Achieved
One implemented FPE scheme for each of these categories: <ul style="list-style-type: none"> • Tiny Space FPE • Small Space FPE • Large Space FPE • Rank-Then-Encipher FPE 	x	x	   
Message spaces supplying each of these formats: <ul style="list-style-type: none"> • Numbers • Strings defined by Regular Expressions / Automaton • Lists / Enumerations 	x		  
Code-Tests written and performed	x		
Documentation: <ul style="list-style-type: none"> • Theory FPE: Introduction / basics / explanation of terms • Technical documentation of the library 	x		 

Table 3: Goals for the Java FPE-Library

2.3.2 Goals for the Tutorial

Goal	Mandatory	Optional	Achieved
Demonstration of the steps implementing FPE in an existing context, based on an example	x		
Realization of code according to the example	x		
Encryption in the example demonstrated with real values	x		

Table 4: Goals for the Tutorial

2.3.3 Goals for the Android Demo App

Goal	Mandatory	Optional	Achieved
Realization of a demo app on Android: <ul style="list-style-type: none"> • Encryption and decryption of data with FPE • Proper GUI / Usability • Secure key management 	✘	✘ ✘	✓ ✓ ✘
Documentation: <ul style="list-style-type: none"> • Implementation of FPE in the app • User guide 	✘ ✘		✓ ✓

Table 5: Goals for the Android Demo App

At the end all mandatory goals were achieved. With the exception of the secure key management in the demo app OwnTracks, also all optional goals could be achieved. Details about the problems we had with the secure key management will be given in chapter 7.4.4.

2.4 Milestones

As milestones we target the following intermediary result at the following dates:

Date	Target	Deadline
Wed. 25. March	FPE library ready	
Mon. 20. April	Documentation of theory and library	
Wed. 13. Mai	Demo app ready	Meeting with expert at 20. Mai
Wed. 27. Mai	Tutorial	
Thu. 11. June	Release bachelor thesis	Yes
Fri. 12. June	Presentation	Yes
Thu. 18. June	Vindication	Yes

Table 6: Milestones

2.5 Use Case and Test Case

It was not the aim of this thesis to develop something completely new or to become specialists in this field of cryptography. The use case that results from the assignment is to demonstrate the integration capabilities of FPE.

The test cases are covered as follows: JUnit test cases are available for the library, which cover the logical correctness. The holistic accuracy can be verified by playing through the tutorial or the demo app.

2.6 Timetable

The thesis starts 16.2.15 and ends 12.6.15 with the presentation respectively 18.6.15 with the vindication. It comprises 12 ECTS points. One point corresponds to about 30 hours and therefore the volume of work is approximately 360 hours per student. Based on the project goals (including all optional goals) we estimated overall volume of work of 786 hours. Effectively we used 780 hours.

Name	Duration planed	Duration effective	Difference
Project management			
Kick-Off meeting	2 h	2 h	
Requirements	8 h	8 h	
Time planning	8 h	8 h	
First meeting with expert	4 h	4 h	
Second meeting with expert	4 h	4 h	
Documentation Project Management	0 h	16 h	+16 h ⁽¹⁾
FPE library			
Revision prototype library	68 h	68 h	
Implementation large space FPE	56 h	56 h	
Implementation tiny space FPE	16 h	24 h	+8 h ⁽²⁾
Documentation			
Introduction	32 h	32 h	
FPE theory	100 h	120 h	+20 h ⁽³⁾
FPE library	100 h	90 h	-10 h ⁽⁴⁾
Tutorial	64 h	64 h	
Completion documentation	36 h	36 h	
Demo App			
Install development environment	4 h	4 h	
Familiarize with Android programming	12 h	12 h	
Requirements / Planning	24 h	24 h	
Develop prototype	48 h	48 h	
Develop app	48 h	40 h	-8 h ⁽⁵⁾
Documentation	32 h	32 h	
Presentation / Reserve			
Reserve	32 h	26 h	-6 h ⁽⁶⁾
Poster / Summary Book	16 h	16 h	
Prepare presentation	32 h	32 h	
Finaltag	16 h	16 h	
Prepare vindication	16 h	16 h	
Vindication	8 h	8 h	
Total	786 h	780 h	

Table 7: Timetable

2.6.1 Remarks to the Time Differences

- (1) At the very beginning we did not plan a dedicated chapter for the project management. Later we subsumed the contents under this chapter and added some deliberations.
- (2) Development took more time than expected because explanations are very open (see 2.9.3).
- (3) Understanding of the papers took more time than expected (see 2.9.2).
- (4) Because of our time delay from the previous point we tried to save time in this chapter.
- (5) Development took less time than expected because we did not realize a whole app on our own as planned in the beginning.
- (6) We used only 26 hours of the designated 32 hours and needed therefore 6 hours less in total.

2.6.2 Planned Estimation of Work Volume

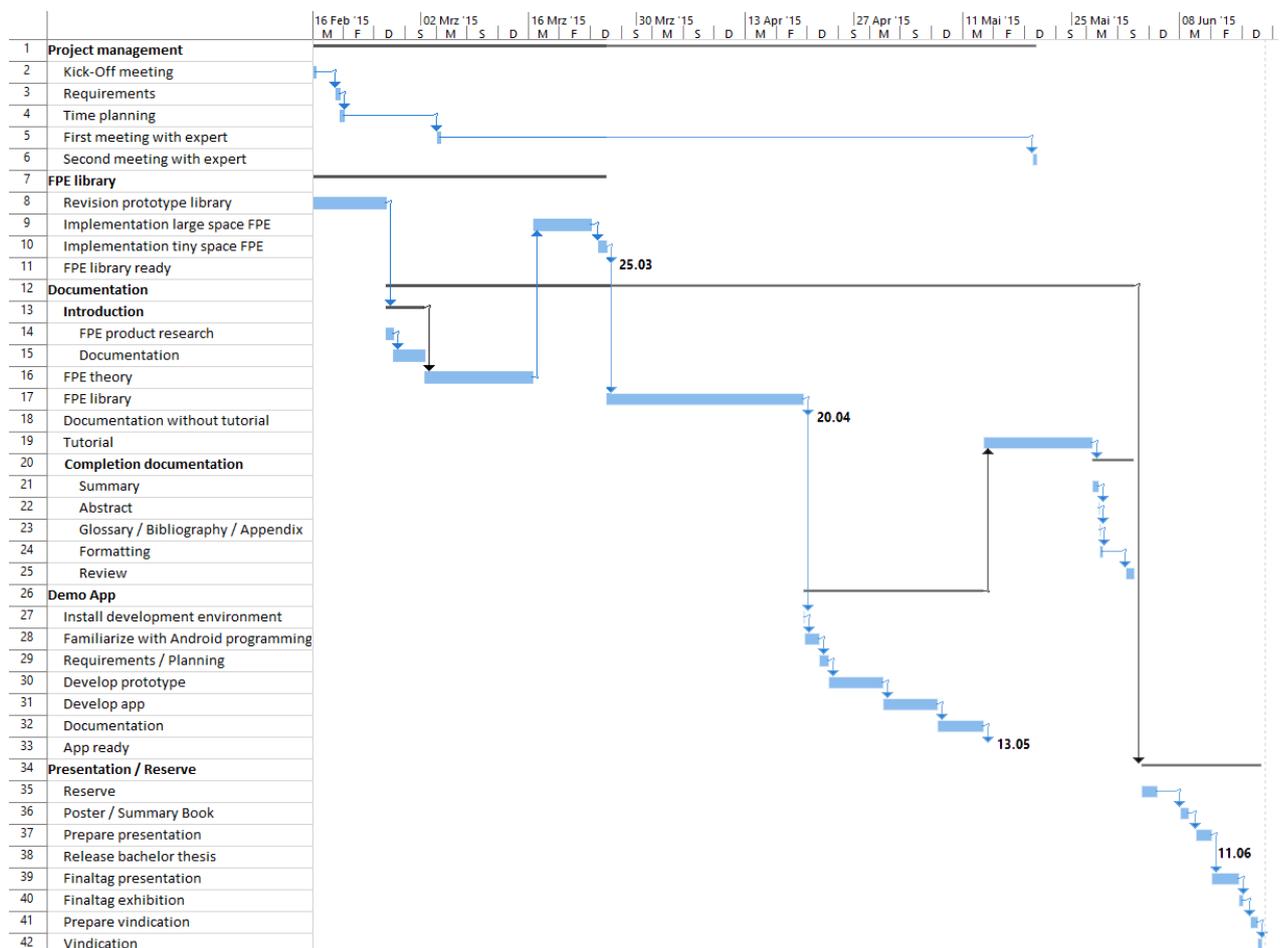


Figure 2: Planned Estimation of Work Volume

2.7 Communication

To avoid misunderstandings, to ensure that we answer the expectations and for exchange of ideas, opinions and corrections, we tried to communicate regularly.

2.7.1 Advisor

The communication with our advisor Reto Koenig took place over email and on periodical meetings. Principally we met every two weeks. And based on our documentation we could regularly ask if we are on the right track and meet the expectations.

Meetings	Main Topics
19. February	Kick off meeting / Goals / Expectations
10. March	Time planning / Content documentation / Goals / Tech. questions library
24. March	Large space FPE / Pseudorandom number generator for Knuth shuffle
14. April	Tech. questions library / Content documentation library / Demo app
28. April	OwnTracks demo app requirements and bugs
19. Mai	OwnTracks
4. June	Documentation / Presentation

Table 8: Meetings with the advisor

2.7.2 Expert

With our expert Stefan Berner we had two meetings.

Meetings	Main Topics
4. March	Introduction / Target / Dates
20. Mai	State of the project / Demo OwnTracks

Table 9: Meetings with the expert

2.7.3 Team

The communication among us took place over WhatsApp, email and in periodical meetings. We met at least once a week – even there were no open questions – to exchange on what we were currently working. Our work was mostly individual work. But we often asked each other for an opinion or support via WhatsApp or email.

2.8 Preliminary Work

During the autumn semester of 2014, within the scope of the module “Project 2” of our study, we already had to deal with the topic Format Preserving Encryption. In that project we made a basic structure for an expandable Java-library including a first implementation of the FPE-algorithm FFX-A2. This library will act as a base for the implementations of this bachelor thesis, but of course we are going to critically reconsider the decisions we made in the design of this library and we try to find improvements.

2.9 Project Reflection

In this chapter we want to reflect our thoughts and problems we had to achieve the milestones in this bachelor thesis from the project management point of view. We discuss the achievements as well as the failures during these steps and what we could have done to change certain negative things respectively what we could improve for future projects.

2.9.1 Project Management

One thing that went very well from the beginning of the thesis was the communication in our team, with our advisor, as well as with the expert. Every time we had a question or a problem during the work, our thesis advisor immediately answered our emails or made free time to organize a meeting to discuss our issues. So we would like to take this opportunity to thank our advisor Reto Koenig for his great support!

A further point we want to mention is our teamwork, which was also never a cause for alarm. We both have already done a few projects together during our studies, so we knew each other's work mentality well and also who likes to do what kind of work. We were able to split the tasks fair, so that everybody of us had more or less the same amount of work to do.

Overall considered the timetable could be kept more or less as planned. Details about some few time differences are described in section 2.6.

2.9.2 Documentation

When we were writing the introduction to cryptography and FPE, we struggled with the question of where to start. As cryptography is a very wide field with a lot of topics to get lost in. So we had to define what we can expect from the reader to know about this topic and what do we have to explain. At the end we decided to write a short paragraph which topic respectively terms in the field of the cryptography should be known to understand our work.

As mentioned a few times in this documentation, Format Preserving Encryption leads a miserable existence in the field of IT security today. As a consequence there are not a lot of understandable or easy written documentation or tutorials about FPE-techniques. Most of the papers which can be found on the Internet about this topic are written in a very technical way with a lot of mathematical formulas and security proofs, expected to be read only by experts. Most of the papers we found about FPE-algorithms like FFX or EME2 were just drafts, made for standardization purpose. Official documentation of these algorithms, which are released after their standardization are only available for a lot of money. This sometimes made the research in the field of FPE schemes complex and annoying.

But overall the documentation part went good as well. Sometimes it was a little bit uncomfortable to write in English, because it felt like we could not unfold our whole literary potential as we could have in German.

2.9.3 FPE Library

From the technical aspect, the FPE library was the most challenging part of our work. In the following the main points are listed we had to cope with.

- **Maven:** We developed our library in eclipse. At the beginning we did not create it as Maven project because it was not necessary. When we planned to integrate it in OwnTracks – which uses Gradle – it suddenly becomes very useful. At this point we tried to create a Maven project out of our current source code, which worked out well after some research and configuration changes. A big advantage was that we could also link easily to dk.brics.automaton, the third-party-library we use, which is also available as Maven project.
- **FFX / EME2:** The implementation of the two main FPE ciphers - FFX and EME2 - was challenging because we are not used to read and implement pseudo code from technical papers. In general this is not a very hard thing, but some points were not completely clear. For instance in the definition of EME2 it says that a value has to be padded with “a 1 byte followed by 0’s”. For us it was not fully understandable if it has to be padded with a byte that has the value one which is “00000001” or with one full byte which is “11111111” and followed by bytes with the value zero. Such points were sometimes exhausting, but fortunately we had reference implementations, where we could have a look at it and get an idea how it has to be done. However, the main problem with the ciphers was that we did not have any test vectors to test our implementation. So at the end we can just assume that they are implemented the correct way by counter-checking the code. For this reason we do not recommend to use our library for high-security applications.
- **Encryption Key:** The Key class contains the symmetric key for the encryption and is able to derive it to the length. We did not plan this class from the beginning. In the process of developing the tiny- and large-space scheme we noticed that we need different key lengths. This would have been no problem because at this point we used a byte array as type for the key. The user then has to supply the needed length according to the scheme used. But problems could arise if one uses the Rank-Then-Encipher-scheme. Because this cipher automatically chooses the integer cipher used for the encryption by itself, one probably does not care which cipher is used. But one is forced to because of the different key lengths. To provide a better usability we then developed this class. It has the advantage that it is possible to derive keys from arbitrary data like a password and to use one key for all ciphers. The disadvantage is that for the user it is less transparent which key length is used.
- **String Message Space:** In the string message space we had to implement the rank and unrank function from P. Rogaway [2, page 10]. But because this code only allows ranking within the same length (called slice) we had to adopt it to rank within the whole message space. This required some brainpower but the real problem was with retrieving the order of the message space. We tried for hours to calculate this value and waited too long until we contacted our advisor. He then showed us that the needed information is already available.
- **Enumeration Message Space:** The EnumerationMessageSpace class contains a collection with elements that are part of the message space. We had many discussions about which data type we should use to store these elements. We determined that the user defines the order of the elements and not for example the alphabet due to performance and other reasons. In this case a Java List would be suitable. However another requirement is that the collection must not contain duplicates. For this case a Java Set would be suitable but it supports no order. And we did not want to use a LinkedHashSet because – when possible – we want to use a well-known format that also does not force the user to convert his data. Because we have to copy the elements at creation anyway so it is not possible to alter them afterwards, we decided to use List and do a manual check for duplicates. Because List does not allow an efficient lookup of the positions of an element, we decided to use an additional Map to allow faster ranks.
- **Knuth Shuffle Cipher:** The Knuth shuffle, respectively the given information cipher was a challenge. P. Rogaway only stated that it is possible to use the Knuth shuffle as FPE cipher but not how to. The algorithm itself is short and clear. But we had a lot of leeway for the integration of the key and tweak. Because we are no cryptographers we had to be careful to only use basic crypto primitives and combine them in a secure manner.

2.9.4 Android App

The main problem dealing with the Android App OwnTracks was the fact that we first had to understand its architecture and functionality. This took quite some time, but fortunately we were prepared for this case and set aside some time for it in our schedule at the beginning of the project. Besides the app we also had to come to grips with the development of mobile apps by itself. We never worked with Android Studio before and had no idea how the GUI with its activities and views is built, for instance. The fact that we had no experience in this development field is also the reason why we were not capable to provide a good solution for storing user password in the Android key store, which is a system we have never worked with and did not know its peculiarities.

Concerning OwnTracks it was sometimes annoying that a lot of functionalities did not properly work. The reason for that is that the version of OwnTracks we forked from GitHub is a developer version which still has a lot of bugs. Fortunately, there were no heavy bugs in the classes we used to implement the encryption. For a similar project it would probably be a good idea not to take the version which is still in development, but rather to take the last official release. But this is only a good solution if it is not planned to send a pull-request to the project team.

At the end we could say that it was a really good feeling to see our FPE-library in a practical example and to observe that everything is working as well as we expected. We are glad that we had the possibility to get an insight into mobile development, an experience that we certainly will be grateful for the future.

2.9.5 Tutorial

The goal of this chapter is to provide a detailed developers tutorial on how to apply FPE in an existing context. The challenge for us was to find this existing context. It seems to us that taking an existing application as an example is not reasonable. It would be too large and complex and an additional introduction to this application would be necessary, which is not the subject of this thesis. The development of an own example, however, is complex. We nevertheless opted for it, also for the reason that we can continue to think about the question of the introduction - which consequences a subsequent equipment of the web application with encryption functionalities might entail. The challenge was that we do not have a lot of experience in developing Java web applications. So we spent some time with research and with troubleshooting.

3 Introduction to Format Preserving Encryption

3.1 Cryptography

Format Preserving Encryption (FPE) is a topic that belongs to an important branch of the computer science, the IT security, or to be more precisely into the cryptography division of ITnsecurity. In the following paragraphs we give a short introduction into cryptography and explain a few basic terms in this field, which are important for understanding the techniques of FPE.

Generally, cryptography describes the science that tries to secure information systems against unauthorized reading and altering. Cryptography builds together with the cryptanalysis, the science of the cryptology. In contrast to the cryptography where engineers try to design secure algorithms e.g. for encryption, the cryptanalysis is the science where one tries to break such algorithms.

Cryptography was mostly used as a synonym for encryption, but today it covers a much wider range of security techniques. According to Wikipedia, the four main goals in cryptography to protect information values could be defined as follows: [19]

1. Message **confidentiality** (or privacy): Only an authorized recipient should be able to extract the contents of the message from its encrypted form. It results from steps to hide, stop or delay free access to the encrypted information.
2. Message **integrity**: The recipient should be able to determine if the message has been altered.
3. Sender **authentication**: The recipient should be able to verify the identity of the sender, the origin or the path it traveled (or combinations) from the message in order to validate claims from emitter or to validate the recipient expectations.
4. Sender **non-repudiation**: The emitter should not be able to deny that he sent the message.

Format Preserving Encryption is a technique which covers only the first goal of cryptography, the confidentiality. The latter is achieved through encryption, which means that it alters data in such a way that only people who possess a specific key are able to decrypt and read the information. Other protective mechanisms as ensuring the integrity of data for example, are not part of FPE and must be tackled with a separate procedure, which will not be covered in our bachelor thesis.

To be able to understand our work and the techniques of Format Preserving Encryption, we expect to have a basic knowledge of symmetric encryption [20] including block ciphers [21] and their mode of operation [22], as well as a familiarity with deterministic encryption [23].

3.1.1 Why Using Cryptography?

When people are asked about their behavior concerning cryptography in general, as for example if they sign their emails or encrypt their hard disk, most of them will probably answer that they do not even know the meaning of this words or they just do not care about their data because they have nothing to hide or to lose. But this attitude is not good at all, for today more than ever before, Internet companies, social networks, intelligence agencies or malicious hackers try to gain all the personal information they can get, even from innocent and unsuspecting citizens. Afterwards this data is used for advertisement purposes, for selling them to third parties or to make a detailed profile of the user. In business or financial environment the risk of compromised data is even higher because of the huge monetary benefit attackers can achieve.

To protect privacy and personal data it should be common sense to use cryptography in everyday applications such as the Internet (SSL/TLS), eBanking, email encryption/signing, VPN technologies, encryption of files or complete hard disks.

3.2 Format Preserving Encryption

Format preserving encryption (FPE) is one specific technology to achieve encryption. In short, format preserving encryption is – as the name suggests – an encryption where the output has the same format as the input. With common encryption technologies, as AES for instance, the output is an unreadable bit string. When we encrypt the Swiss phone number 032 321 61 11 with AES, we get something like C2AC 2A8D 5910 53BE 4F80 FD24 1C53 47F6. If we use a FPE cipher the result could be something like 043 453 23 92, again a valid Swiss phone number.

First, we give an example of one possible application. A more general overview about the possibilities is given in section 3.3 *Areas of Application*. After that we will take a look at the formal definition and make the distinction to other similar technologies.

3.2.1 Basic Example

For the explanation of FPE the example of encrypting a credit card number is used often. But why is the format of the encryption an issue? Could not we use the encrypted credit card number as a bit string?

We consider the following example. A bank owns an ATM and the ATM communicates with a bank server. The communication is made over an external network of a financial service provider with which other banks are also connected. Therefore the communication has to be in a given format and the credit card number must not be encrypted in order to ensure that the routing is possible.

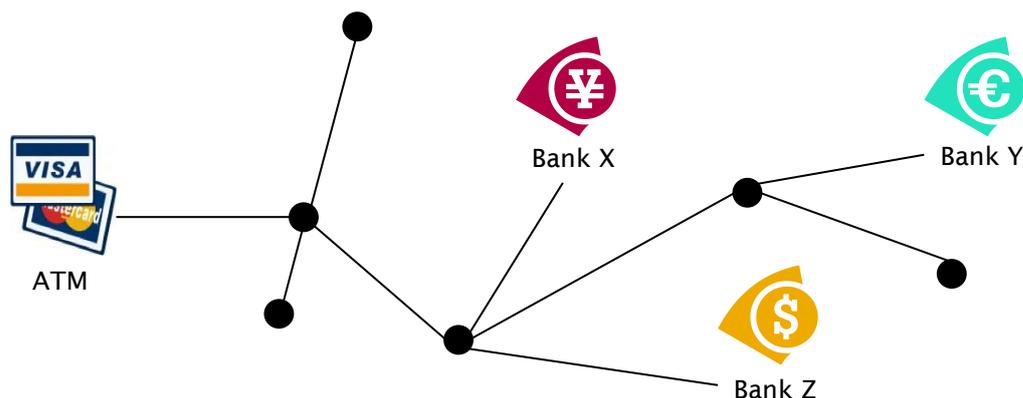


Figure 3: Communication between bank and ATM

What can a bank do to protect itself against the financial service provider? Here FPE can come into play. With FPE it is now possible to encrypt a credit card number and the encrypted output is again a credit card number. The financial service provider is therefore unable to distinguish an encrypted from an unencrypted one. But in this scenario the encryption of the whole number is not possible.

A credit card number consists of a six-digit Issuer Identification Number (IIN) which identifies the issuing organization, an individual account identifier with variable length up to 12 digits and a checksum calculated with the Luhn algorithm. When we encrypt the whole number the routing would not be possible anymore. But the bank does not have to encrypt her Issuer Identification Number which is public anyway. It can encrypt only her individual account identifier with FPE, prefix it with her Issuer Identification Number and finally calculate the checksum on it.

For example the number “371449 63539843 1” can result in the encrypted “371449 37194735 9”. We see that the credit card number is valid and can be routed by the financial service provider but the sensitive information is not accessible.

3.2.2 Definition and historical background

The idea of a format preserving encryption is not new. Already in 1981, a DES-based approach [6] was described to encrypt strings of a fixed alphabet A which could be for example $A = \{0, 1, \dots, 9, a, b, \dots, z\}$. A plaintext ($x \in A^*$) with arbitrary length is then encrypted into a ciphertext ($y \in A^{|x|}$) of the same length. This is done by creating a pad with the symmetric encryption algorithm DES, which has the length of the plaintext and is from the alphabet A . The pad is then added letter by letter to plaintext modulo the biggest character of the alphabet. [1] But it turns out that this technique is defective and very easily attackable. If an attacker finds a plaintext/ciphertext-pair, he also has the pad and is able to decrypt all other ciphertexts with the same or shorter length.

In 1997 a more general technique was proposed by Brightwell and Smith [5]. They called it data type-preserving encryption and the goal was to encrypt database entries without changing the data type. Philipp Rogaway says in [1] that the scheme they sketch is “cryptographically naïve, complex and not fully pinned down, but the recognition of the problem was highly insightful”.

There are many more FPE schemes (for more details see chapter 4 *FPE Schemes*) and over the years many individuals and research groups dealt with the FPE problem. Some research findings resulted in standards. This is the case in the field of the hardware full-disk encryption, because there was a great demand for a cryptographic solution. Other findings become less popular and some were replaced with a better successor or were discarded because they are insecure.

In general FPE is a symmetric encryption $E: X \times K \rightarrow X$.

E is a reversible function which performs a permutation. With the forward computation some plaintext is encrypted. With the backward computation the ciphertext is decrypted and we receive the original plaintext. K is the shared symmetric key. And X specifies the domain of the plaintext respectively the ciphertext. The domain is not restricted in the general definition. If the domain is a range of natural numbers, we speak of integer FPEs. If the domain is an arbitrary but nonempty and finite alphabet, we speak of string FPEs.

FPE is per se not randomized or stateful; it is deterministic. In other words every time a plaintext is encrypted with the same key we are going to get the same ciphertext. Some FPE schemes like FFX prevent this by allowing supplying a further value, called initialization vector (IV) or tweak in case of FFX. Because this tweak is encrypted as well, two identical plaintexts with different tweaks result in different ciphertexts. In many scenarios it is of advantage or even essential that an attacker cannot decide that ciphertexts originate from the same plaintext. The tweak has the aim to bring randomness and is not secret, has therefore not to be encrypted and can be sent with the ciphertext. In some cases the term “associated data” is used instead of tweak. This is because often data associated with the plaintext is used as tweak. For instance when encrypting hard disk sectors, the sector number is a very good solution for a tweak, because it is related to the plaintext (the sector) and the relation remains the same for the ciphertext (the encrypted sector).

3.2.3 Distinction

There are two technologies which are similar to FPE, which we do not speak about in this paper. These are the format-transforming encryption (FTE) and the tokenization.

Similar to FPE, format-transforming encryption allows defining the domain of plaintext and ciphertext. However FTE allows another format for the output as for the input. FTE is used for example by the Tor Project to circumvent deep packet inspection [33] or by the fteproxy [34].

Tokenization, on the other hand, is not an encryption but can be an interesting alternative to FPE in some cases. With Tokenization the sensitive data is not encrypted but replaced with a random but unique substitution (token). The token can be within the same domain but does not have to. For example a credit card number can be replaced with another random but valid credit card number. The mapping is saved in a secure database and only the token is used in the external network. Tokenization has the advantage that it removes the data entirely and therefore reduces the risk of exposure. In other applications FPE is the only solution because the data has to be available in different places and not only the token.

3.3 Areas of Application

There are mainly two reasons why one wants to preserve the format of a plaintext. On one side because there is a restriction on a system, similar to the example with the credit card number we have seen. In this case the goal is to trick a computer program into believing that there is no encryption applied on the data. On the other side there are usages where the goal is to trick a human being into believing that there is no encryption on data.

The following non-exhaustive list provides a compilation of some applications, where Format Preserving Encryption is already in use or where it could be a serious alternative to existing encryption techniques.

- Subsequent encryption of data in a database without the need of altering the data structures respectively the data types of the attributes. FPE allows encrypting a database, which has to be running 24/7, because for the database nothing actually changes on the format or structure. Only the data changes through the encryption. The only thing that possibly changes for the database is the addition of an attribute that defines if a record is already encrypted or not. But this step varies depending on how the FPE feature is implemented.
- Legacy systems which should be secured but on which nobody wants to alter anything. For example because there is no knowhow anymore in the company about this system or the impacts of direct changes on it are not clear.
- External or third party Applications that expect a particular input format, for instance for an address or a phone number. One does not want to provide real and possibly personal data to an application but wants to be able to reproduce the real information out of the given mock data.
- For the encryption of data on which third parties like cloud provider should not be able to see that they are actually encrypted. In this case a further use of personal data can be excluded.
- Application tests where real data in a database should be used to have an authentic test scenario with the correct format, a logical meaning and sensible relationships. Sometimes it is not possible to use productive data because they are confidential. With FPE this data could be encrypted to generate test data.
- Encryption of hard disks, where the goal is to encrypt a hard disk in such a way, that it is possible to connect it to an operating system without getting any format error. A hard disk is divided into so-called disk sectors and to successfully bind it to an operating system, these sectors have to be recognized and have to be correlated with the according sector number. This means an encryption has to be provided that does not change anything on the structure of the hard disk sectors and remains the header information, as well as their size, which is traditionally 512 bytes large for common hard disk drives.

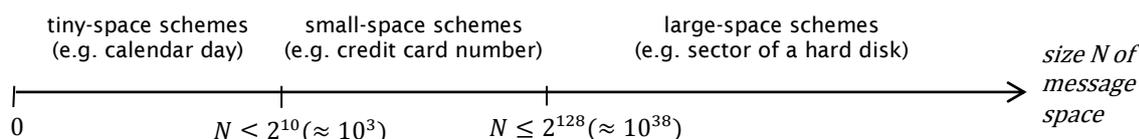
For most of these areas of applications for FPE commercial products with FPE techniques already exist. You can find some examples in the section *12.5 Applications which use FPE*.

4 FPE Schemes

One possibility to categorize FPE schemes is to differentiate them according to the size of their message space. The message space is the domain in which the plaintext and the ciphertext lie. The order of the message space defines the amount of possible plain- respectively ciphertexts. Based on the amount of bits used, these schemes could be categorized into tiny-space-, small-space- and large-space-schemes.

The limit between tiny-space- and small-space-schemes is not fixed. It can vary depending on application and requirements. It is also possible not to distinguish and to operate only with small-space schemes. The advantage of a differentiation lies in the security of a small-space scheme, which can be proven mathematically. There is no known attack against small-space schemes with tiny domains (e.g. the day of the week) but they typically have worse or no provable security guarantees (see [1], page 4). The advantage of doing no differentiation is to reduce complexity and error-proneness because less cryptographic schemes are in use. A further point is the efficiency of tiny-space schemes, which can be smaller. The time complexity is $O(N)$. For an encryption/decryption all elements of the message space have to be passed through. To compensate this, all values could be pre-computed and filed into a table so that they do not have to be calculated for every encryption call.

The limit between small-space- and large-space-schemes however is fixed. It is given by the block size of the block cipher used. With the most commonly used AES encryption standard it is 128 bits.



We choose one scheme of each category that we implement in our FPE library and describe in detail. For a list of different schemes see the appendix.

4.1 Tiny Space FPE Schemes

Tiny space FPEs are the schemes with the smallest message space. As stated above, the limit is around 10 bits but varies depending on the application. Message spaces of this size could be for example PINs, IDs, calendar days, week numbers, gender, street numbers, zip codes, age and so on.

4.1.1 Schemes

In [1] Philipp Rogaway refers to three schemes which can be used for tiny space FPE encryption: Prefix Cipher, Knuth Shuffle and Permutation numbering. We choose one of the three for the implementation. The choice fell on the Knuth shuffle because of its simplicity. It has similar properties like the prefix cipher but can be implemented more efficiently.

	Prefix cipher	Knuth shuffle	Permutation numbering
Initialization:	- slow setup of permutation table		+ no setup
Enc-/Decryption:	+ fast by lookup in permutation table		+ few AES calls needed - complex implementation (standard Java class library does not have support for the factorial number system)

Table 10: Advantages and disadvantages of three tiny space FPE schemes

The Knuth shuffle is described in the following. A short description of the other two can be found in the appendix.

4.1.2 Knuth Shuffle

Some tiny space schemes are based on the card shuffling metaphor. We consider the elements as a deck of playing cards. We first have to decide the natural order and note the position for each element. Then we use the key as receipt to shuffle the cards. There are a variety of techniques respectively algorithms for shuffling. The key is always used to bring the necessary randomness. After the shuffle procedure we take the new position for the permutation.

The Knuth shuffle (or Fisher–Yates shuffle) is one technique to generate such a permutation. It is named after his inventors Ronald Fisher and Frank Yates respectively Donald Knuth who popularized it. An improvement reduced the time complexity from $O(n^2)$ to $O(n)$. The improved algorithm looks as follows [30]:

```
To shuffle an array a of n elements (indices 0..n-1):
  for i from n - 1 downto 1 do
    j ← random integer with 0 ≤ j ≤ i
    exchange a[j] and a[i]
```

To explain the scheme, we consider as an example the encryption of a weekday. Therefore the domain could be $X = \{0 = Sun., 1 = Mon., 2 = Thu., 3 = Wed., 4 = Thu., 5 = Fri., 6 = Sat.\}$. We now have to decide which day decrypts to which other day with the use of a symmetric key.

When we do again the example with encrypting weekdays by pencil-and-paper we could come to the following permutation:

Index i	Random j	Original	Permutation
		0 1 2 3 4 5 6	
6	4	0 1 2 3 6 5	4
5	3	0 1 2 5 6	3 4
4	0	6 1 2 5	0 3 4
3	1	6 5 2	1 0 3 4
2	2	6 5	2 1 0 3 4
1	0	5	6 2 1 0 3 4
			5 6 2 1 0 3 4

Table 11: Example of a Knuth shuffle

In contrast to other use cases of the Knuth shuffle, the “random integer” must not be truly random. A shuffle, for example in a card game, should be as randomly as possible. But here it has to be deterministically so that a message can be decrypted again. Therefore the random integer has to be derived deterministically from the key. But it is of interest that the reverse is not possible. If an attacker guesses a permutation he should not be able to get the corresponding key.

Such a pseudorandom number generator (PRNG) could be implemented in the following way:

1. We use a block cipher like AES and encrypt a value like “hello world” with the given key. The value can be chosen arbitrarily but then has to be retained to get deterministic results. The key and possibly an IV/tweak are the secret elements used to generate the pseudorandom number.
2. To get the random integer j in the Knuth shuffle loop, we take the AES output and calculate it modulo the index $i + 1$. Due to the addition of one we cover the given range of $0 \leq j \leq i$.

The security is given because only standard crypto primitives are used and combined in a common manner.

4.2 Small Space FPE Schemes

Small space schemes support larger messages spaces than tiny spaces schemes but are limited to the block size of the block cipher used. If we use the AES encryption standard it is limited to 128 bits. Therefore with a binary alphabet we can use 128 digits, with a decimal alphabet 38 digits and with 24 case insensitive letters 27 digits and with the 95 printable ASCII characters 19 digits. Examples are credit card number, IBAN, names, short addresses et cetera.

4.2.1 Schemes

In the following chapter we explain in detail the small space FPE scheme FFX. FFX stands for “Format-preserving, Feistel based encryption”. An alternative scheme is BPS, named after the authors Brier, Peyrin and Stern. It was developed independently and published almost simultaneously.

The choice to consider FFX and use it as small space FPE scheme in our library was given by an earlier project work in which we developed a prototype of it. Due to the limited time of this work we had to confine ourselves to this scheme.

In the appendix 13.3 *Small-space FPE schemes* is a non-conclusive list of further schemes.

4.2.2 FFX

In FFX the underlying structure is a Feistel construction. For a better understanding of FFX we first look at the characteristic of Feistel.

4.2.2.1 (Balanced) Feistel

Feistel gives a general structure for the construction of a block cipher. It is generic because it only defines the principle and leaves the parameter open to be defined by the cryptographer. It was developed by Horst Feistel, a researcher of IBM, which did one of the first non-military researches in the field of cryptography.

The classical, balanced Feistel is constructed iterative, in several rounds. The process of every round is the same: the input is processed and the output serves as input for the next round. Every round provides additional security.

The construction starts by splitting the input in a left and a right half. These two parts serves as input for the first round. The input has to be of even length in order that the halves have the same length. In this sense the construction is balanced. That is why the classical variant is also called “balanced Feistel”.

One round then proceeds as follows: the right half is handed over unaltered to the left side. The left input however is changed: with a XOR the output of the round function F is applied. As input for the round function servers the right half of the input and a round specific key. The definitions of the round function F is one of the primary duties at the design of a Feistel cipher. The genius with the Feistel round function is that it do not have to be reversible.

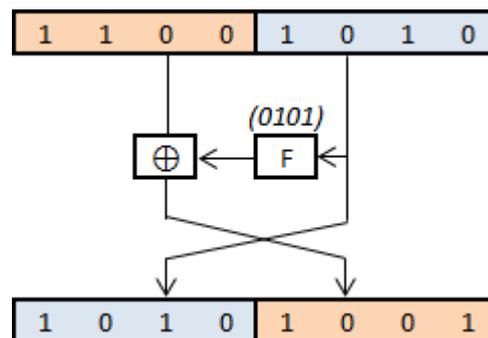


Figure 4: Balanced Feistel

At the decryption the whole construction is passed through backwards. At the end one obtains the original value. The round function must not be reversible. Because the input is the same as in the encryption it supplies the same deterministic result. The subsequent XOR reverses the encryption process. This is the case because the XOR gives back the original value at the second application.

4.2.2.2 Feistel on Non-Binary Alphabets

Because we often do not work with binary alphabets, the previous input has to be converted before it can be committed to the Feistel construction. Another possibility is to work directly on the non-binary alphabet. For example we could use directly a decimal number.

This has two practical consequences. For one thing one cannot longer use the XOR because it is a bitwise operator and works only on binary numbers. And for another thing also the round function has to be adapted that it provides outputs in the respective alphabet.

A replacement of the XOR operator could be realized by

adding the two blocks. The last carryover is, if present, ignored. In the illustration on the right we combine 1234 with 9217. This results in $1234 + 9217 \equiv 10451 \equiv 451 \pmod{10^4}$. In

the decryption the process is reversed with a subtraction: $451 - 9217 \equiv -8766 \equiv 1234 \pmod{10^4}$. This method is called block-wise addition. Another possibility is the character-wise addition, which is similar but the individual characters are added instead of the whole block. P. Rogaway states in [1] that there is no security difference between them but there might be an efficiency difference in some settings.

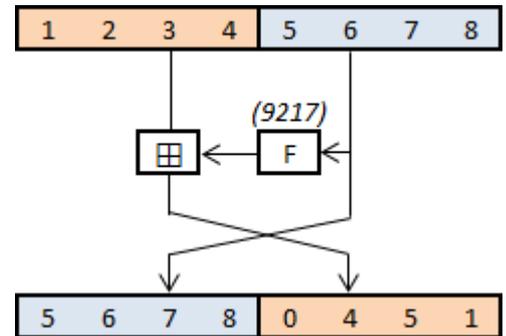


Figure 5: Feistel on Non-Binary Alphabets

4.2.2.3 Unbalanced Feistel and Alternating Feistel

By now we only looked at inputs of even length. But the Feistel construction can also be generalized and applied to inputs of even or odd length. There are two popular techniques for that: the unbalanced Feistel and the alternating Feistel.

Unbalanced Feistel

The unbalanced Feistel works principally like the balanced Feistel. The difference can be found between the rounds. After every round the two parts are repartitioned so that they always have the same length and can be used equally.

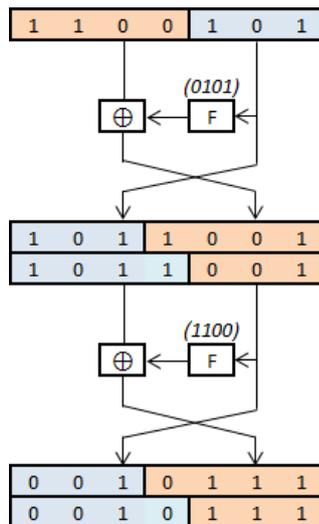


Figure 6: Unbalanced Feistel

Alternating Feistel

With the alternating Feistel the two parts are not exchanged crosswise but the round function is applied alternately to the left and the right side. The small space FPE scheme FFX is based on alternating Feistel.

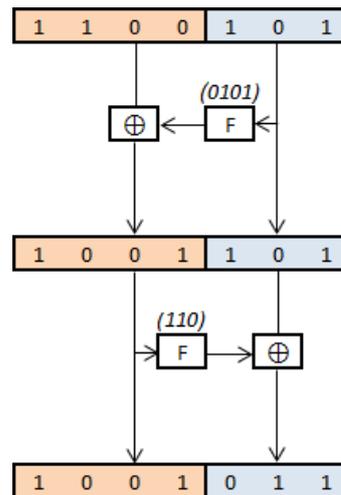


Figure 7: Alternating Feistel

The disadvantage of the unbalanced Feistel is the repartition. What only looks like a shift of the viewpoint and not actual work is likely to result in work in the actual implementation. The disadvantage of the alternating Feistel is that it alternately has to produce outputs of different length.

4.2.2.4 FFX

As mentioned, FFX is based on alternating Feistel. Previously we looked at the construction of Feistel. To understand the whole functionality of FFX we now have to look at the variable parts of Feistel. These are the round function, the round keys and the number of rounds. In FFX they are defined in the following way:

- Round function: The round function is based on the AES CBC MAC. Reason for using an existing cryptographic function is that the AES CBC MAC is a broad-based and well known technique. It creates from an arbitrary long input and a symmetric key a 128-bit value, called message authentication code (MAC). A MAC is normally used to verify the integrity of a message. Thus to verify that a message was not altered on the way from the sender to the receiver. Here in FFX the CBC MAC is used to generate a pseudorandom number. As input serves beside the input from the Feistel round also other information like the round number, tweak and others. The output is a 128-bit string which then has to be shortened to the needed size.
- Round keys: The same 128-bit FFX-key is used for all rounds.
- Number of rounds: In a first version [3] the number of rounds were defined depending on input size. For a shorted input size more rounds are used than with longer inputs. In the subsequent addendum [4] it was reduced to 10 rounds for all input lengths and alphabets.

FFX was first defined for a binary (FFX-A2) and a decimal (FFX-A10) alphabet. It was submitted to the American National Institute of Standards and Technology (NIST) in 2010. Shortly afterwards another independently developed FPE method was submitted: BPS by Brier, Peyrin and Stern [11]. As a response the authors of FFX revised their specification and proposed a new version – FFX[radix] – with the following improvements: First the alphabet is expanded and not more limited to the binary or the decimal alphabet. Second the length of the allowed messages is enlarged. And finally it uses a tighter round count independent of the message size.

The following table gives an overview of the FFX parameters defined in the specification [3] and [4]. The algorithm of the round function F and further details can be found in the papers.

Parameter	Description	FFX-A2	FFX-A10	FFX[radix]
radix	Radix of the numbering system. Defines the alphabet of plain- and ciphertext.	2 (i.e. digit 0 and 1)	10 (i.e. digit 0 – 9)	$2 - 2^{16}$
length	Set of allowed lengths of plain- and ciphertext.	8 – 128	4 – 36	$2 - (2^{32} - 1)$, if radix < 10 min. is 8
key	Symmetric key for encryption/decryption.	128-bit AES key		
tweak	Nonempty string used to prevent determinism.	$2^{64}-1$ Byte (i.e. limit of 64-bit OS)		$2^{32}-1$ Byte (i.e. limit of 32-bit OS)
addition	Defines addition operator in the Feistel construction. Either 0 for character-wise or 1 for block-wise addition.	0 (=XOR)	1 (=block-wise addition)	
method	Defines which Feistel method is used: 1 for unbalanced or 2 for alternating Feistel.	2 (=alternating Feistel)		
split(n)	Position after which Feistel input is split.	$\lfloor n/2 \rfloor$ (maximally balanced Feistel)		

rounds(n)	Number of Feistel rounds.	$\begin{cases} 12 & \text{if } 32 \leq n \leq 36 \\ 18 & \text{if } 20 \leq n \leq 24 \\ 24 & \text{if } 14 \leq n \leq 18 \\ 30 & \text{if } 10 \leq n \leq 14 \\ 36 & \text{if } 8 \leq n \leq 10 \end{cases}$	$\begin{cases} 12 & \text{if } 10 \leq n \leq 36 \\ 18 & \text{if } 6 \leq n \leq 9 \\ 24 & \text{if } 4 \leq n \leq 5 \end{cases}$	10
F	Feistel round function	AES based round function.		

Table 12: Overview of the FFX parameters

It is beyond the scope of this work to explain the security of FFX. Provable security results of FFX can be found in [2] and of the Feistel construction in general in [18].

But in [1] P. Rogaway states that provable security results for Feistel are steadily improving but there is still a huge gap between today's attacks and what we can prove. But on the other hand Feistel is the best-known approach to build block ciphers for now 40 years. Attackers almost never tried to attack the construction itself, but the round function. So, brute-force attack on the AES key space is the best known attack on FFX.

4.3 Large-Space FPE Schemes

Large-space FPE schemes are necessary to encrypt binary numbers bigger than 128 bits or in other words, they are used to encrypt message spaces with more than 2^{128} possible values.

As we have seen in the previous chapter, FFX[radix] allows encrypting arbitrary long binary strings. Theoretically we could just implement this version of FFX and we would have a FPE-scheme that covers small-space and large-space scenarios. However, in the documentation of FFX[radix], the developers of this scheme, including P. Rogaway, give the following statement:

"First, we would emphasize that, while we have permitted enciphering very long strings, mode FFX[radix]—and indeed FFX in general—is intended for enciphering relatively short strings. For binary strings whose length exceeds 128 bits, EME2 may be a better choice than FFX[radix]." [4]

Because of this advice we decided to have a look at EME2 and not to implement FFX[radix]. Another reason for this decision is that we have already implemented the FFX-A2 scheme as a prototype in an earlier project work, which covers the small-space part of our library and we now only need a large-space FPE. Besides that, our general opinion is that we would like to become acquainted with various FPE schemes and we do not want to get stuck in only one scheme like FFX.

The large-space FPE scheme EME2 was originally developed for the encryption of hard disks, a scenario we explained in section 3.3. EME2 was developed by the in 2002 formed IEEE P1619 Security in Storage Working Group (SISWG), which is responsible for the development of standards of disk encryption schemes. Before we are going to see the algorithm of EME2 in detail, let us first have a closer look at the standards released by this working group and the development of EME2.

4.3.1 IEEE P1619

The IEEE P1619 Security in Storage Working Group (SISWG) is a working group that develops standards related to encrypted storage media including encryption and key management.

The standards resulted of the work of this group are so far:

- P1619.0 (2007): Standard Architecture for Encrypted Shared Storage Media (AES-XTS)
- P1619.1 (2007): Authenticated Encryption with Length Expansion for Storage Devices
- P1619.2 (2009): Standard for Wide-Block Encryption for Shared Storage Media (EME2, XCB)

In the base IEEE P1619 standard, the AES-XTS mode was introduced. XTS stands for XEX-based Tweaked CodeBook mode with ciphertext stealing and is an advancement of the XEX block cipher mode of operation proposed by Phillip Rogaway. Without going into great detail, the actual advancement of XTS is the ciphertext stealing part, which allows encrypting data that is not a multiple of the block size without any enlargement of the ciphertext compared to the size of the plaintext – which makes it to an FPE scheme according to the definition. AES-XTS is today supported by a lot of disk encryption software like BestCrypt, TrueCrypt, FreeBSD's geli or Mac OS X Lion's FileVault. [26]

So if even today a lot of applications use AES-XTS mode to encrypt hard disks, why was there a need for another standard with newer large-space FPE algorithms as they resulted in XCB and EME2 introduced in the standard P1619.2?

The reason for this is actually quite understandable, as almost every other cryptographically algorithm AES-XTS has its weaknesses too. AES-XTS is a so-called narrow block algorithm, which means that it encrypts only a small size of plaintext, in the case of AES 16 bytes, into the same size of ciphertext. This makes the mode susceptible to traffic analysis, replay and randomization attacks on sectors and 16-byte blocks. As a given sector is rewritten, attackers can collect fine-grained (16 byte) ciphertexts, which can be used for analysis or replay attacks. [31]

The flaw of AES-XTS is commented as follows by the IEEE P1619 working group:
“IEEE Std 1619-2007 acknowledges in D.3 that the attacker has greater malleability with 16-byte narrow block encryption modes like XTS, versus wide-block encryption modes like EME-2 or XCB (see IEEE P1619.2). However, by making this trade-off, the encryption algorithm need only make one pass over the data unit instead of the two passes required by wide-block encryption modes. It is important for the designers to understand this trade-off and pick the appropriate mode. XTS is appropriate if the attacker has limited access in making malicious attacks and if the programs have a high probability of detecting a randomized 16-byte block.” [12]

As explained in this comment, these weaknesses can be avoided by using wide-block algorithms like the mentioned XCB or EME2 which are able to encrypt a whole sector of 512 bytes or more at once. But these wide-block schemes have a disadvantage and that is their bad performance. This is also the main reason why algorithms like XTS are still in use. As we are going to see, EME2 is designed with an encrypt-mix-encrypt principle which means that the whole data has to be encrypted twice and not just once like in a narrow-blockcipher. The costs for encrypting n AES blocks with EME2 are thus $2n+1$.

Even though P. Rogaway recommends to use EME2 as alternative to FFX[radix] we did not want to automatically discard all other schemes, because of the fact that Rogaway is also the main developer of EME2 and it is understandable that he first recommends his own scheme. However, in [1] Rogaway gives an alternative to EME2 if a large-space FPE is needed and that is the other scheme standardized in P1619.2, the already mentioned XCB.

For the sake of completeness let us say that there are of course other blockcipher-based large-space FPE schemes beyond the standard P1619.2. But because of the fact that these two algorithms are standardized, therefore tested and approved by the IEEE and recommended by famous cryptographers as it is P. Rogaway, we decided to focus on these two algorithms for our decision. For an assortment of alternative large-space FPE schemes, have a look at the list in appendix 13.4.

4.3.2 Large-Space Scheme of our Library

So in the end we had to decide if we want to implement EME2 or XCB as large-space FPE scheme for our implementation and our choice fell on EME2. Before we introduce the EME2 algorithm, let us make a few words about the loser XCB and our main thoughts for this decision.

The extended codebook (XCB) mode of operation for block ciphers was developed by D. McGrew and S. Fluhrer from Cisco Systems Inc. XCB has almost the same properties as EME2. The minimum bit length for the input plaintext is also the block size of the underlying block cipher and the choice of the block cipher is also free, with a recommendation for AES-128. As well as EME2, XCB supports arbitrary long tweaks. There are actually two main differences between XCB and EME2. First of all XCB does not use ECB as mode of operation for the underlying block cipher, but the Counter Mode (CTR) and the construction of XCB relies on a much more complicated design based on a hash-encrypt-hash principle.

The first reason for our choice on EME2 is the fact that its design with the encrypt-mix-encrypt principle is way easier to understand and therefore to implement than the hash-encrypt-hash design of XCB. Even after an intense study of the XCB-algorithm we were still not completely sure how it is exactly designed, where on the other side the design of EME2 was relatively simple to understand. The complexity of a cryptographic algorithm is very important, because if an application using cryptography is broken, normally not the used algorithm was the weak point but the mistakes made

during the implementation of the algorithm. So in such a situation as we have, it is usually the better choice to implement an easier algorithm where the likelihood of creating bugs is smaller, if the given security is the same. Actually this complexity of hash-encrypt-hash schemes like it is implemented in XCB was the reason why P. Rogaway developed the encrypt-mix-encrypt-approach EME2. [1]

However, the main reason for our decision was not even the complexity of XCB but rather the amount of security concerns on XCB which are much higher than they are on EME2. Since these two algorithms were standardized in 2009, cryptanalysts all over the world tried to find vulnerabilities in them. In 2013 three researchers from Mexico and India published a paper where they describe vulnerabilities of XCB. They found out that it is possible to make an easy distinguishing attack on XCB, which means an attacker is able to distinguish encrypted data from random data. Modern encryption schemes are normally designed to be immune to such an attack, which means to have ciphertext indistinguishability. Other vulnerabilities were found, based on the fact that security proofs in the standard of XCB are not correct and the security bound of XCB is significantly weaker than what has been claimed. [17]

On the other site, concerning the security aspects of EME2 we found a paper that was published by some other Mexican cryptanalysts. In this document they describe weaknesses in EME2 and its predecessor EME towards side-channel attacks. In cryptography, side-channel attacks are procedures that do not attack the cryptographic algorithm itself to find weaknesses in it, but they attack or gain information from the device (e.g. smart card) or software the algorithm is implemented in it. These possible attacks they found do not directly weaken the proofed security of EME2, as they clearly explain in the conclusion of their paper: *“We presented some attacks on EME and EME2 assuming that our specified operation xtimes leaks some information. These attacks does not contradict the claimed security of the modes, as the security definition and the security proofs for these modes does not assume any side-channel information being available to the adversary. Also the consequences of these attacks shown are not immediate. But, it points out that using xtimes indiscriminately may give rise to security weakness in true implementations.”* [16]

The only advantage we would have had if we had chosen XCB as large-space FPE would have been the speed. By comparing these to modes by the cost of the encryption of n AES block, XCB needs n+1 where EME2 needs 2n+1. So we could say that XCB is twice as fast as EME2. The main reason for that is, as we are going to see in the explanation of the algorithm, that EME2 needs two AES-encryptions per plaintext block where XCB only needs one. Besides that, the mode of operation for the underlying block cipher on XCB is the Counter Mode (CTR), where EME2 uses ECB. Under certain circumstances used by AES, CTR can be up to 8% faster than ECB. [13]

So at the end we had the choice between a way understandable and more secure scheme against a weaker although fast scheme. Our general opinion is security before performance and so our choice fell on EME2.

4.3.3 EME2

EME2, respectively EME* as it was called at the beginning, is a further development of the EME mode of S. Halevi and P. Rogaway. EME is a mode of operation known for his efficiency and parallelism but does not support messages of arbitrary length. This is why the two decided to take the mentioned advantages of EME and evolve it to EME2. EME2 uses a regular block cipher and turns it into a length-preserving encryption scheme for message of arbitrary length. The underlying block cipher, for example AES with its block size n of 128 bits and a key length of 128 bits is taken, runs on the Electronic Codebook mode of operation (ECB). This is also where the name EME actually comes from, ECB-Mask-ECB.

The smallest possible value to encrypt is the block size of the underlying block cipher and that means 128 bit with AES. So this scheme can only be used as a large-space FPE and not for smaller applications. The arbitrary maximum is just theoretical, because there exists also an upper limit for the bit length of the plaintext which is $n(2^n - 3)$ that corresponds to 6.8×10^{38} bytes and should obviously not be a practical limitation anytime soon. The EME2 scheme is tweakable meaning one can add associated data to prevent deterministic encryption.

The key for EME2 consists of one key of the underlying cipher and two additional n -bit blocks, where n is the block size of the underlying block cipher. So if taken AES with key length 128-bit as block cipher, the provided key for an EME2 encryption has to be 384 bits long.

EME2 is mainly based on an encrypt-mix-encrypt approach. As can be seen in the following figure, there is a kind of a layer at the beginning where first all the input data is encrypted. The input plaintext is divided into blocks of 16 bytes length and encrypted separate. If the input is not a multiple of 16 bytes, the last block will padded to 16 bytes.

In the middle part of the algorithm, the encrypted data is used to create different masks (SP,SC,MP,M,MM). With the help of these masks and the given tweak, which is first formatted to a 16-byte block, the already encrypted data will now calculated xor. This procedure is called the mixing part. At the end there is a layer where the masked data is encrypted once again.

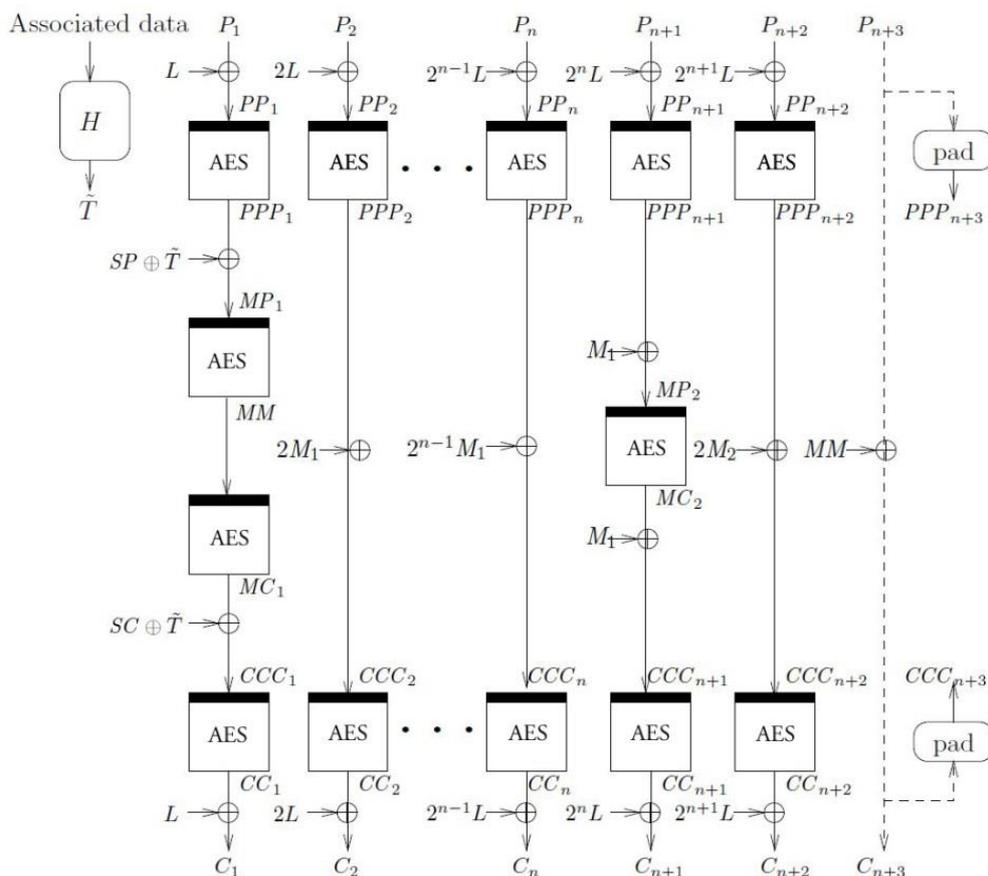


Figure 8: Procedure of EME2

4.4 Cycle Walking

In some cases we can come to the problem that the result of an encryption is no longer in the message space. Cycle walking is a technique to deal with this problem. The idea is encrypt the encrypted output until it is again inside the message space.

In section 4.2 *Small Space FPE Schemes* we get to know the Feistel construction. How the input is split in two halves and preceded through the round function in several rounds. The encryption of a decimal number on a binary alphabet with Feistel is an example where we can run into this situation that an encryption is no longer in the message space.

Say we want to encrypt an integer with three digits with a binary alphabet. Thereto we need ten bits, because $2^9 < 10^3 < 2^{10}$. But what happens when we exceed three digits (999)? With ten bits it is also

possible to represent some four-digit number (1000 – 1024). The chance that we get a four-digit number – on the supposition that the calculation is random – is $(2^{10} - 10^3)/2^{10} = 2.3\%$

Here the cycle walking technique comes into play. It is one possibility to deal with the situation that a message can end up outside the message space. Another possibility with Feistel is to not convert the input and work directly on the non-binary alphabet. With cycle walking, one takes the output which lies outside the message space and encrypt it again. The output of the second encryption is validated again. If it is now inside the message space the procedure is successful and the message is returned. Otherwise it is repeated as long as necessary. The process for encryption and decryption is exactly the same.

```
public Message encryptWithCycleWalking(Message m) {
    do {
        m = m.encrypt();
    } while (messageSpace.containsNot(m));
    return m;
}

public Message decryptWithCycleWalking(Message m) {
    do {
        m = m.decrypt();
    } while (messageSpace.containsNot(m));
    return m;
}
```

Figure 9: Encryption and decryption with cycle walking

In the abovementioned example with a four-digit number the chance for a repetition is relatively small. In other cases it could be considerably higher. For example with a two-digit integer (and 16 bits) the probability to exceed the desired range is at 37.5%. Because this is a probability the effective number of repetitions can be much higher. This unpredictable runtime behavior can be disturbing if a user has to wait a long time in some cases and in some not.

4.5 Rank-Then-Encipher

What we have seen so far are FPE schemes which enable to encrypt a number in a predefined number range with the guarantee, that the encrypted number will also be a value in this range. But these FPE schemes have actually a lot of restrictions. For instance, if we use FFX as small-space FPE and EME2 as large-space FPE together, we are able to encrypt numbers from zero to arbitrary length, but we cannot encrypt negative numbers or define a number range which is not starting by zero or 128 bits. In other words, it is not possible to cover very simple use cases with for instance, a number in the range of 500-1000 and the encryption with the goal to get another number in this range as ciphertext.

Apart from these limitations, until now we were only talking about encrypting numbers with FPE schemes. But would it not be interesting to be able to encrypt any kind of string in a certain format and to be ensured that the encrypted string is also in the same format? This kind of question bothered also a lot of computer security researchers and so a new FPE-approach called Rank-then-Encipher (RtE) was developed.

The main idea behind this approach is that not the actual plaintext itself is encrypted, like all the FPE-schemes we have already seen do it, but a so-called “rank” is assigned to this plaintext which makes him distinctly identifiable in a certain message space. Following this, the rank which is just a simple number can be encrypted with an underlying FPE-scheme like FFX or EME2. By using one of these FPE-schemes and defining the maximum value of the message space as biggest possible encryption output, we can ensure that the encrypted number is also a valid rank in this message space. After the encryption we get another rank and by unranking this rank we get the encrypted ciphertext, which is simply the value assigned to the encrypted rank.

The basic idea behind this RtE-approach is obviously relative simple. One important part behind this approach is the already mentioned message space. Like the key or the tweak, the message space has always to be the same for the encryption and decryption of a value. But let us have a closer look to this construct.

4.5.1 Message Space

A message space X_N is a construct that contains all possible values given to a particular format. In other words, a message space contains each combination that is possible with a given amount of characters, a so-called alphabet. $X_N = \{X_0, X_1, \dots, X_{n-1}\}$

Each of these values are distinctly identifiable with a number inside a message space, where this number can be compared to an index in a database table. In the Rank-Then-Encipher approach this number is called “rank”. The order n of a message space gives the amount of values in it $n = |X_N|$.

To provide a short example, for the input-string “ba” the required alphabet would be $A = \{a,b\}$ and the corresponding message space would contain $X_N = \{aa,ab,ba,bb\}$. The order of this message space is therefore four.

The format of the elements of such a message space does not have to be strings like in the example, but can be defined variably, depending on what should be encrypted. Theoretically a message space can be built for every special format or application. It only has to provide a ranking and unranking function, as well as a function which returns the order of the message space. This requires that the elements in the message space are finite. In a real world scenario a message space could also be a direct binding to a database, where the ranks are given through the index of the tables. For our FPE-library we decided to implement the following message spaces:

- Number range
- Strings
- Lists / Enumerations

4.5.1.1 Number range

By determining a message space with a number range, a minimum value and a maximum value has to be provided. The minimum value or both of them can certainly be a negative number.

On the right there is an example for a number message space from -150 to 200 and their corresponding ranks.

Rank	Value
0	-150
1	-149
...	...
349	199
350	200

Table 13: Number range message space

4.5.1.2 Strings

For building a message space with string values in it, one have to provide rules which define what kind of strings should be stored in it, respectively what format is allowed for the elements in this message space. These rules can be defined by regular expressions (regex) or automaton.

By providing the regex $[A-Z][a-z]\{1-10\}$ which is a simple format for names beginning with an upper case letter and followed by one to ten lower case letters. The message space built with this regex would have the values shown in the table on the right somewhere in it.

As represented, these values would probably trick a parser into believing these are real names but a human being would recognize that these values are just random letters.

Rank	Value
...	...
56983261	Csieujale
56983262	Csieujalf
56983263	Csieujalg
56983264	Csieujalh
...	...

Table 14: String message space

4.5.1.3 Lists / Enumerations

To convince a human that encrypted values are not encrypted but real values, there is the possibility to build a message space with the help of lists. For instance a list with all possible names could be provided and the message space would look somehow like this:

In lists respectively enumeration, it is possible to store any kind of values in it and thus to encrypt them. A list could as a special example, hold Java Class Objects and the ciphertext of an encrypted object would emerge to another Java Object.

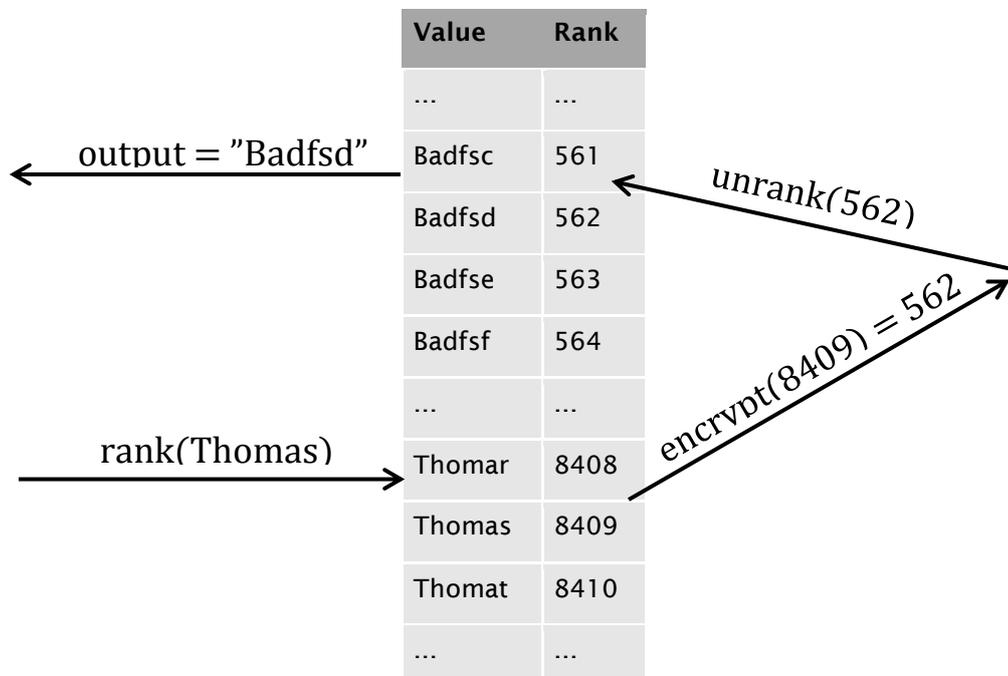
Rank	Value
...	...
4321	Manuel
4322	Marcel
4323	Martin
4324	Matthias
...	...

Table 15: Enumeration message space

4.5.2 Example of the Rank-Then-Encipher-Approach

To consolidate the idea of the Rank-Then-Encipher approach, let us have a look to a simple complete example. First we build a message space with a regular expression that determines a first name, starting with an upper case letter, followed by three to ten lower case letters: $[A-Z][a-z]\{3-10\}$.

By encrypting the value "Thomas", this value has first to be ranked. The ranking function delivers the corresponding index to this value in the message space and that is the number 8409. Now an integer FPE-scheme like FFX is used to encrypt the value 8409 to a new rank in this message space. Explained once again, we use a FPE-scheme to be sure that the encrypted rank is a number not bigger than the order of the message space. FFX returns 562 as new rank. By unranking 562 we get the corresponding value at this place in the message space and that is in this example "Badfsd". Hence we have encrypted the value "Thomas" to a value with the same format "Badfsd".



The decrypting process corresponds to the encryption. First the value „Badfsd“ would be ranked which gave consequently the rank 562. This rank decrypted will be the rank 8409 and the unranking of this rank will return the origin value „Thomas“.

5 FPE-Library

Implementing FPE features into a program is today a huge effort. One has to look up various FPE schemes, understand how they work and which one fits the best. Additionally the most important FPE scheme for most of the use cases scenarios, the Rank-Then-Encipher approach is not very known and so it is heavily possible that at the end FPE - or even encryption in general - will never be implemented because it seems too sophisticated.

Our goal with this FPE library is to enable programmers to integrate encryption with FPE techniques without any huge effort or the need of a big knowledge about this topic. The only thing a user will have to do is to import our library into his workspace, write a few simple lines of code and voilà, the program is successfully enriched with Format Preserving Encryption.

We decided to develop our library with the programming language Java. There are actually two reasons for this decision, first of all Java is one of the most popular programming language today and has the “write once, run anywhere” principle, meaning that compiled Java bytecode can run on any Java virtual machine regardless of the computer system on it. Therefore our library could be integrated in applications that are developed for a normal computer with Windows, Mac OS or Linux as operating system, as well as for instance on a cellphone running Android. Secondly Java is the main programming language of our study, so it is the language we know the best and feel the safest using it. We used the Java SE Development Kit 8 to develop our library, for this reason in minimum Java SE Runtime Environment 8 has to be installed on devices designated to run our library.

The complete source code of our library is hosted at: <https://github.com/EVGStudents/FPE>

5.1 Class Overview

Now we want to go a bit more into the technical internals. We first give an overview over the library classes and then explain them individually afterwards.

The library is grouped into three areas respectively Java packages.

- ch.bfh.fpe: Contains the base class from which all other FPE Ciphers are derived. It also contains the classes that could not be assigned to one of the two following packages.
- ch.bfh.fpe.intEnc: Contains all integer FPE schemes, thus FPEs which message space is a range of natural numbers.
- ch.bfh.fpe.messageSpace: Contains the different message spaces.

A general, not detailed overview looks as follows. On the left there are the two packages which contain the FPE ciphers, on the right the package which comprises the message spaces.

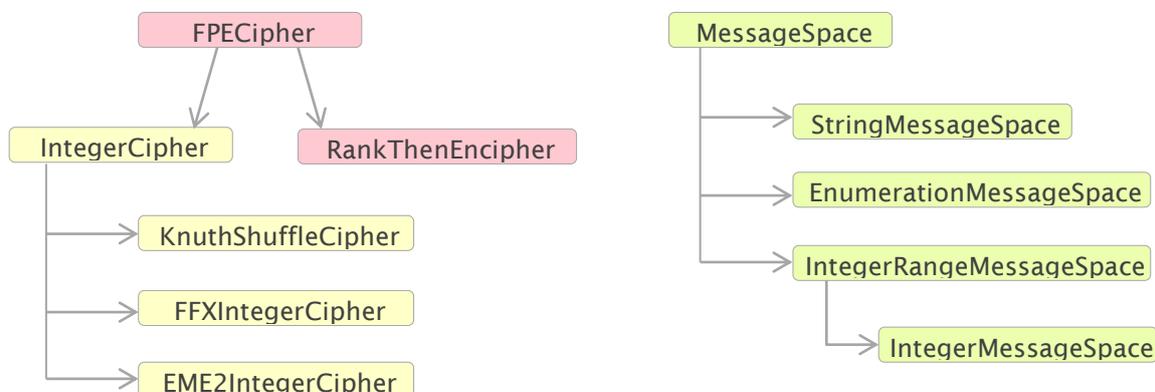


Figure 10: FPE-Library Class Overview

A more detailed view is given in the two following illustrations. The first shows the packages with the ciphers, the second the package with the message spaces.

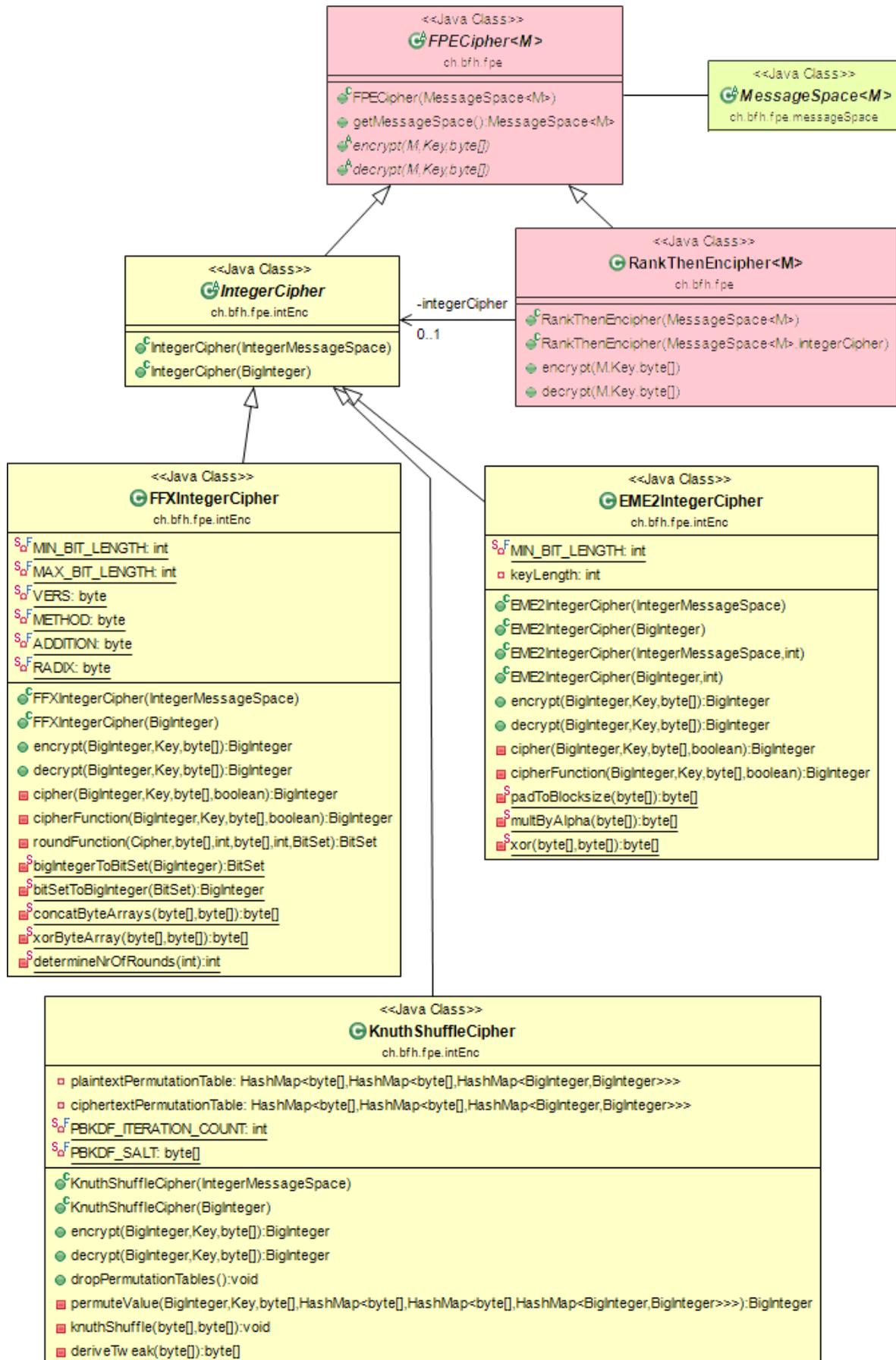


Figure 11: FPE Cipher Classes

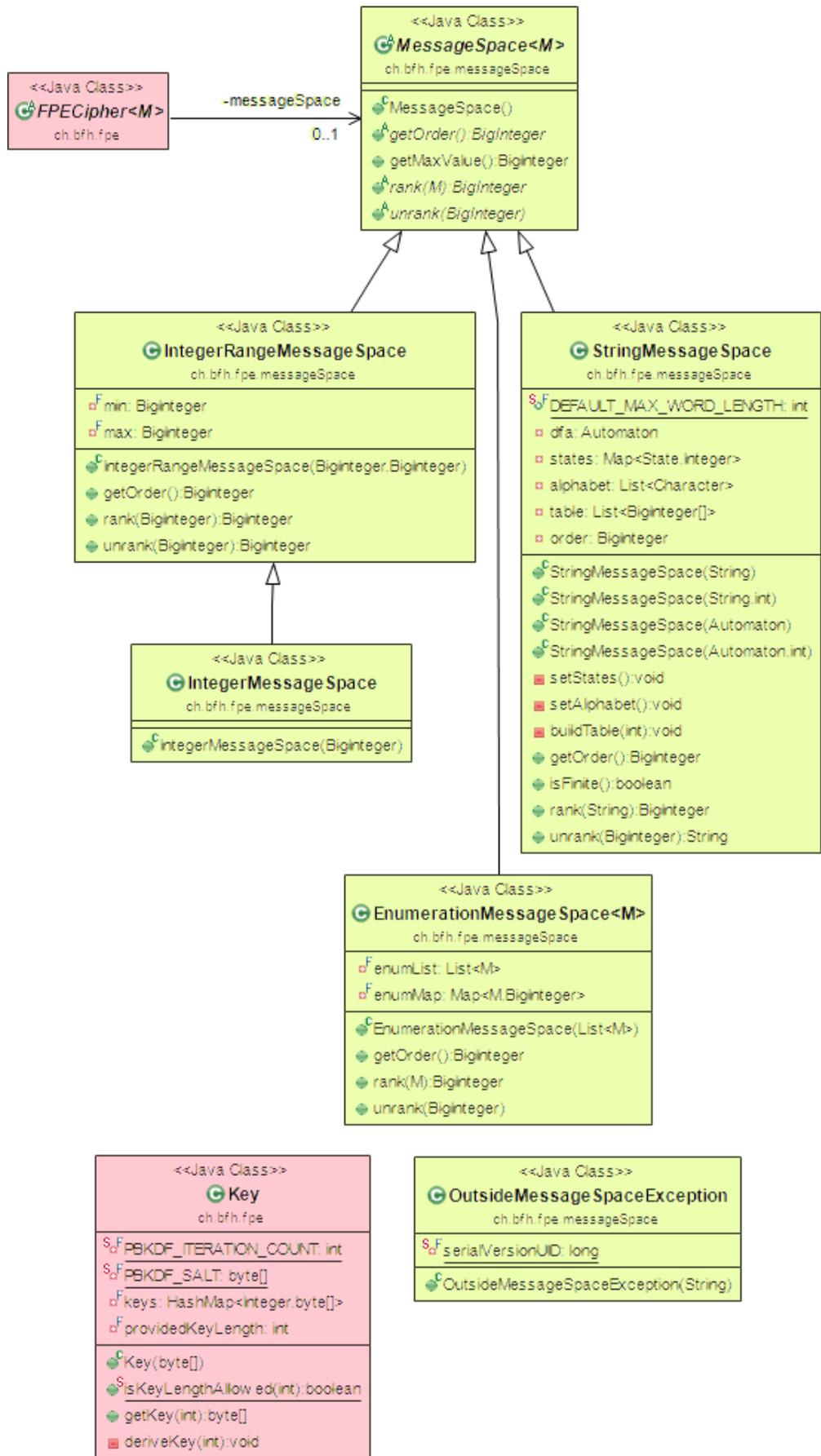


Figure 12: Message Space Classes and Key Class

5.2 Classes

5.2.1 FPECipher<M>

Package:	ch.bfh.fpe	Superclass:	Object
Type:	abstract class	Interfaces:	-
Description:	<p>This class is an abstract base class from which all FPE ciphers are derived. It contains a message space which is delivered at construction. Every plaintext and ciphertext must be an element of this message space otherwise an <code>OutsideMessageSpaceException</code> is thrown.</p> <p>Earlier we saw that an FPE cipher is a symmetric encryption $E: X \times K \rightarrow X$. The class defines this abstract method E for the encryption and also one for the decryption. The generic type <code>M</code> defines the type of an element $x \in X$ or in other words the type of an element of the message space.</p> <p>The method for the encryption and decryption contains three parameters: One with the value that should be encrypted/decrypted, one with the symmetric encryption key and one with the tweak.</p> <p>The class does not have program code but yet is a class and not an interface because it contains the message space.</p>		

5.2.2 Key

Package:	ch.bfh.fpe	Superclass:	Object
Type:	class	Interfaces:	-
Description:	<p>The <code>Key</code> class is used to provide the symmetric encryption key used by <code>FPECipher</code>. Therefore in every encryption/decryption call of a FPE cipher one has to supply an instance of this class.</p> <p>Because varying FPE ciphers use different key lengths, this class provides the functionality to derive a key of the required length. On creation, a key of arbitrary length is provided. When a key is requested one has to specify the desired size. If the requested length is the same as of the key supplied at creation time, the provided key is returned unaltered. If the requested length is smaller or longer, a new key is derived from the provided one.</p> <p>The method used to derive a key is PKCS#5 (PBKDF2 with SHA1-HMAC) with a fixed iteration count of 10'000 and a fixed salt. The salt is fixed because already the original key is a secret which has to be stored securely and the omission greatly simplifies the use for the user of the library. It is furthermore not obvious why in an encryption one has to provide one salt for the encryption and one for the key itself.</p> <p>The advantage of this class over a simple byte-array with a key is that one could use FPE ciphers with varying key lengths with storing and supplying only one key. For example with <code>RankThenEncipher</code> cipher one does not even have to define which cipher is used and let <code>RankThenEncipher</code> automatically determine an appropriate one. Therefore it is at the beginning not apparent which key length is needed. For example a value inside the regex-message space "[1-7]" is processed with a <code>KnuthShuffle</code> and therefore needs a 16 byte key. For value inside the message space defined by the regex "[A-Z][a-z]{1,20}" the <code>EME2</code> cipher is chosen which takes a 48 byte or 64 byte key.</p>		

5.2.3 IntegerCipher

Package:	ch.bfh.fpe.intEnc	Superclass:	FPECipher<BigInteger>
Type:	abstract class	Interfaces:	-
Description:	<p>The abstract class IntegerCipher represents a FPE integer cipher. Therefore all FPE ciphers which handle natural numbers are derived from this class. It encrypts a given number from a specified range in such way, that the output value is also a number from the same range. The range is defined by an IntegerMessageSpace delivered in the constructor or by an integer value which defines the upper bound. The upper bound is a convenience parameter; in the background a IntegerMessageSpace is constructed because every FPECipher is defined by a MessageSpace.</p> <p>IntegerCipher works with BigInteger as type of the elements of the message space. BigInteger allows theoretically arbitrary big numbers. In practice it is limited by memory and some methods that assume that the bits are addressable by int indices.</p>		

5.2.4 KnuthShuffleCipher

Package:	ch.bfh.fpe.intEnc	Superclass:	IntegerCipher
Type:	class	Interfaces:	-
Description:	<p>This class is an implementation of a tiny space FPE scheme based on the Knuth Shuffle. Phillip Rogaway states in [1] that the Knuth shuffle is a possibility to build a tiny space FPE but gives no detail on a possible implementation.</p> <p>More details about how we used the Knuth shuffle to create a permutation can be found in section 4.1.2. As we have seen in the introduction of chapter 4 <i>FPE Schemes</i>, it is recommended to use KnuthShuffleCipher for up to ten bits (one or two digit decimal numbers) and switch to FFXIntegerCipher for larger message spaces.</p> <p>At first use, for every key and tweak a permutation table is created, which defines which plain text is mapped to which cipher text and vice versa. The table is created when it is first used in an encryption or decryption. Every subsequent encryption/decryption is then a fast table lookup. The permutation table consists of two HashMap. One is use for encryption and the plaintext is used as key and the ciphertext as corresponding value. The other is used conversely for the decryption.</p> <p>The tweak can be of arbitrary length and is then adapted if necessary to the 16 byte needed in the AES function. The length is left open intentionally because it is neither restricted in the other integer ciphers. Because it is possible to use RankThenEncipher and let it automatically choose an appropriate integer cipher, this ciphers have to handle the input in the same manner. If the tweak is too short or too long it is adjusted with PKCS#5 (PBKDF2 with SHA1-HMAC). Because the tweak is a public value and only has to be adjusted to the needed size, the derivation does not have to be secure and an arbitrary function with this aim could be taken. Therefore PBKDF2 is used with the round count of 1 and a fixed salt.</p>		

5.2.5 FFXIntegerCipher

Package:	ch.bfh.fpe.intEnc	Superclass:	IntegerCipher
Type:	class	Interfaces:	-
Description:	<p>FFXIntegerCipher is an implementation of the "FFX Mode of Operation for Format-Preserving Encryption" [3] without the addendum [4] which was published a moment after. It enciphers numbers from zero to a maximum of 38 decimal digits (128 bits). The bit length of the message space has to be equal or greater than 8 bit, otherwise an exception is thrown. For smaller message spaces use the tiny space scheme KnuthShuffleCipher.</p> <p>FFX is based on a Feistel construct and in the round function it uses AES-128 as underlying block cipher. More details on the functionality are given in section 4.2.2.4.</p> <p>The free parameter of the FFX algorithm are set as follows:</p> <p>VERS =1, METHOD=2 (alternating Feistel), ADDITION=0 (characterwise addition, xor), RADIX=2 (only two symbols, zero and one)</p>		

5.2.6 EME2IntegerCipher

Package:	ch.bfh.fpe.intEnc	Superclass:	IntegerCipher
Type:	class	Interfaces:	-
Description:	<p>EME2IntegerCipher is an implementation of the EME2 (formerly known as EME*) [14]. It enciphers arbitrary big numbers. But the bit length of the message space has to be at least 128 bit. For smaller message spaces use the tiny space scheme KnuthShuffleCipher or FFXIntegerCipher.</p> <p>More details on the functionality are given in section 4.3.3.</p> <p>One constructor contains a parameter to define the key length used by AES which is either 128 or 256 bit in the case of EME2. The EME2 cipher function is based on the encrypt-mix-encrypt approach and uses AES for the encryption. The default key length is 128 bit to support interoperability because JCE without unlimited strength policy files is restricted to this size.</p>		

5.2.7 RankThenEncipher<M>

Package:	ch.bfh.fpe	Superclass:	FPECipher<M>
Type:	class	Interfaces:	-
Description:	<p>This class is an implementation of the "Rank-then-Encipher Approach" for Format Preserving Encryption Cipher [2]. This approach is explained in detail in section 4.5.</p> <p>There are two constructors available. In both one has to specify the message space on which one intends to work. But in one constructor the appropriate integer cipher is specified and in one not. If it is omitted, RankThenEncipher choose it based on the bit length of the message space. Up to 7 bit, the Knuth Shuffle is chosen, up to 128 bit FFX and for more than 128 bit EME2.</p> <p>The message space has to be of the same generic type M as RankThenEncipher. Therefore to encrypt strings one has to create a RankThenEncipher<String> and use</p>		

message space that extends `MessageSpace<String>` like the `StringMessageSpace`.

The method for encryption is short and plain. It is depicted for a better understanding:

```
public M encrypt(M plaintext, Key key, byte[] tweak) {
    BigInteger rank = getMessageSpace().rank(plaintext);
    BigInteger rankEnc = integerCipher.encrypt(rank, key,
    tweak);
    M ciphertext = getMessageSpace().unrank(rankEnc);
    return ciphertext;
}
```

The decryption function is identical except that it calls the `integerCipher.decrypt`.

5.2.8 MessageSpace<M>

Package:	ch.bfh.fpe.messageSpace	Superclass:	Object
Type:	abstract class	Interfaces:	-
Description:	<p><code>MessageSpace</code> is the base class for all message spaces. A subclass defines a message spaces which can be used by a FPE cipher.</p> <p>A message spaces enables to define a domain and it provides methods to get the rank of an element of this domain and the inverse, the element at a particular rank. It also provides a method to return the order of this message space, therefore the number of elements in the domain and the convenience method to return the maximal value which is the order minus one.</p> <p>The rank starts counting at zero and goes up to the order of the message space minus one. All implemented classes in this library are immutable. Thus the domain is defined over the constructor and cannot be changed after construction. This is sensible because a change after an encryption would lead to different result at decryption.</p> <p>In the context of this format-preserving encryption library it is used in two different ways:</p> <ul style="list-style-type: none">- A integer FPE cipher aims to encipher on a message space of the form $X' = [N] = \{0, 1, \dots, N - 1\}$ for some arbitrary number N. The implementing class <code>IntegerMessageSpace</code> is used to define N.- The rank-then-encipher approach is used when the message to encrypt is not part of the aforementioned domain X' but in a related domain X. The message space is then used to transform an element of X to an element of X' and backwards. For more details see the documentation of the <code>RankThenEncipher</code> class.- The message spaces could also be used independent from FPE to perform rank/unrank operations.		

5.2.9 IntegerRangeMessageSpace

Package:	ch.bfh.fpe.messageSpace	Superclass:	MessageSpace<BigInteger>
Type:	class	Interfaces:	-
Range:	$[n, m] := \{x \in BigInteger \mid n \leq x \leq m\}$		
Description:	Message space that is defined over an integer range. Negative values are allowed. The range is defined with the lower- and upper limit in the constructor. The lower limit must be smaller than the upper limit.		

5.2.10 IntegerMessageSpace

Package:	ch.bfh.fpe.messageSpace	Superclass:	IntegerRangeMessageSpace
Type:	class	Interfaces:	-
Range:	$[0, m] := \{x \in BigInteger \mid 0 \leq x \leq m\}$ respectively $[m, 0] := \{x \in BigInteger \mid m \leq x \leq 0\}$		
Description:	IntegerMessageSpace extends the IntegerRangeMessageSpace and therefore also specifies an integer range but the minimum is fixed to zero. Because we start counting at 0 for both rank and value, they are always the same. A IntegerMessageSpace has to be provided when creating a integer ciphers.		

5.2.11 StringMessageSpace

Package:	ch.bfh.fpe.messageSpace	Superclass:	MessageSpace<String>
Type:	class	Interfaces:	-
Range:	Defined by a DFA respectively a regular expression.		
Description:	<p>The StringMessageSpace defines a message space of strings. The domain of the string is defined by a deterministic finite-state automaton (DFA) respectively a regular expression. The constructor allows the specification of one of the two. A regular expression is then internally converted into a DFA.</p> <p>Both constructors allow the specification of the maximal word length. When not specified a default of 128 characters is used. This is necessary with some DFAs and regular expression because in DFAs there is no limitation of recursions and with regular expressions it is possible to define infinite languages. On the other hand a message space has to be finite in order to be able to rank/unrank. Thus if one use for example the regular expression operation * or the word length is longer than the default of 128 characters, this parameter should be used to define the maximum length of the string.</p> <p>For the rank and unrank method the approach described in [2] is implemented. With the exception that the rank is not only defined over strings of the same length but all strings within the message spaces.</p> <p>In this approach a table is pre-computed at creation time which then allows fast rank and unrank operations. The table contains for each string length (column) and for all states (row) the number of possible transitions from this state. Thus if one takes an initial state and some length one gets the number of possible transitions and therefore the order for this string length.</p>		

5.2.12 EnumerationMessageSpace<M>

Package:	ch.bfh.fpe.messageSpace	Superclass:	MessageSpace<M>
Type:	class	Interfaces:	-
Range:	Defined by a List<M>.		
Description:	<p>The EnumerationMessageSpace is defined by a List. The elements can be of arbitrary type M and are passed at construction time. Later they cannot be changed to retain the order. This is necessary because an altering would result in different ranks and consequently to wrong encryption/decryption.</p> <p>The user of this class specifies the order of the elements through the ordered list and is responsible to retain this order for every subsequent use.</p> <p>Internally the class consists of a List<M> with the elements. These are copied at construction time to prevent modifications and retain the order. Duplicate elements are only copied once because the list has to be distinct. The list allows a fast lookup of an element at specified position; thus a fast unrank. Additionally the class contains a Map<M,BigInteger> again with the element and furthermore the position index for a fast rank.</p>		

5.2.13 OutsideMessageSpaceException

Package:	ch.bfh.fpe.messageSpace	Superclass:	RuntimeException
Type:	class	Interfaces:	-
Description:	Exception that is thrown when a value is supposed to be an element of a message space but it is not.		

5.3 External Libraries

The following libraries from the Java core API and the Java core extension API (javax) we imported during the development of our FPE-library.

- java.math
- java.nio
- java.util
- java.security
- javax.crypto

The only third-party package, meaning it is not included in the standard Java Runtime Environment (JRE), we use in our library is Dk.brics.automaton. We use this package, which was mainly developed by Anders Møller at Aarhus University in Denmark, to handle automaton (DFA) and regular expressions (regex) in our StringMessageSpace class.

On the developers website the library Dk.brics.automaton is described as follows: "This Java package contains a DFA/NFA (finite-state automata) implementation with Unicode alphabet (UTF16) and support for the standard regular expression operations (concatenation, union, Kleene star) and a number of non-standard ones (intersection, complement, etc.).

In contrast to many other automaton/regex packages, this package is fast, compact, and implements real, unrestricted regular operations. It uses a symbolic representation based on intervals of Unicode characters." [50]

5.4 Juridical Aspects

Some of the libraries and algorithms we use in our library may have patent claims or copyrights on them. Within the scope of this bachelor thesis, the use of these algorithms is not a problem, but this is going to look different if one thinks about integrating this library into a commercial product, which will be sold.

For this reason we tried to evaluate and summarize the juridical aspects of these algorithms:

Algorithm / Library	Developer	Patent / License Evaluation
dk.brics.automaton	Anders Møller	The package dk.brics.automaton is under the BSD License. This means the code can be modified and sold with the only condition that there is a copyright reference to the original developer. [50]
FFX	M. Bellare, P. Rogaway, T. Spies	In February 2015 Hewlett-Packard acquired the company Voltage Security and owns now a FPE Patent which includes the FFX mode. [35]
EME2	S. Halevi, P. Rogaway	EME2 seems not to have any patents claims on it. In a mailing list of 2013 a message from P. Rogaway says that "There is no Rogaway or UC patent related to EME2." [36]
Knuth Shuffle	Donald Knuth	There is not patent on the Knuth Shuffle. In 1994 the famous computer scientist D. Knuth wrote a letter in which he holds the opinion that computer algorithms in general should not be patented. [37]

Table 16: Juridical aspects of the FPE-algorithms

5.1 Development Potentialities

In the following the areas are sketched out, in which we see possibilities to enhance the library.

5.1.1 FPE-Cipher

We implemented one FPE scheme of each message spaces size. Thus all necessary domains are covered and it is possible to work with strings of arbitrary length.

Still it could be interesting to provide an alternative scheme. For example all our three implemented schemes were proposed by P. Rogaway and he contributed to two of them. So it may be sensible to provide an alternatively developed method.

For a future work we propose to implement the following FPE-Schemes:

- **BPS** by Brier, Peyrin and Stern [11] because it is an alternative developed by different cryptographers and it is like FFX a NIST standard proposal.
- **FFX[radix]** because it is the extension of FFX-A2 that is actually implemented. It allows an arbitrary alphabet and message space sizes. Because FFX[radix] on a binary alphabet is not backward compatible it should be implemented additionally and not as a replacement.

5.1.2 Message Space

The implemented message spaces allow a broad range of application. An interesting extension could be a database message space which ranks and unrank enumerations directly in the database. This could be an advantage if the data is anyway in a database. It could be ranked efficiently without loading the whole data into the application. Another application is where the data does not fit into memory. The database will not have to load it completely into memory in order to rank it.

The question is how the position of an element (respectively row) is found. The simplest solution, which is also independent from the database management system, is to add an id number which is incremented for every row. For fast access this id should be the primary key respectively has an additional index on it. As in the EnumerationMessageSpace we have the problem that a list which is in use must not be altered. Additionally we have the issue that with the deletion of records it comes to gaps in the numbering which is not allowed. The numbering has to start with zero and it has to be gapless.

5.1.3 Generally

Our first aim was it to provide a library that allows a secure and flexible application of FPE. In a further step it would be sensible to direct one's attention to the runtime performance and the possibilities of optimization. In the following chapter we give some more details about the performance.

5.2 Performance

5.2.1 Benchmarks

The following benchmarks should give a feeling for the speed of our library. These measurements are just snapshots and will vary, depending on the allocation of the CPU, the amount of free memory on the system and other factors.

For all the tests the same random 16-byte key and the same tweak with 1296 bytes were taken. This tweak size was determined randomly. Be aware that the size of the tweak will also affect the speed of the algorithms.

Each measurement was made ten times and the averages of these passes were taken as measured end value.

To perform these measurements, the following devices were used:

Device	Type	Operating System	CPU	CPU Cores	RAM
PC	Steg PC Series	Windows 7 x64	Intel Core i7-2600 @ 3.4GHz	8	8 GB
Notebook	Samsung Ultrabook Series 5	Windows 8.1 x64	Intel Core i5-3317U @ 1.7GHz	2	4 GB
Smartphone	Samsung Galaxy S4 (GT-I9505)	Android 5.0.1 (Lollipop)	Qualcomm Snapdragon 600 @ 1,9 GHz	4	2 GB

Table 17: Devices used for the benchmark

The first benchmark shows a comparison of our three implemented FPE-Ciphers to one classical AES operation on all devices. For the sake of simplicity we only provide the time for the encryption. The decryption times were in average a bit faster than the encryption, but the proportion between the devices stays the same.

Cipher	Message Space Size	Time to encrypt		
		PC	Notebook	Smartphone
AES-128 (ECB Mode)	128 bit (1 Block)	0.05 ms	0.36 ms	0.07 ms
KnuthShuffle	7 bit	14.8 ms	86.4 ms	948.4 ms
FFX	128 bit	5.7 ms	34.8 ms	28.9 ms
EME2 (AES-128)	8'000'000 bit (1 MB)	121.8 ms	748.6 ms	5'368.7 ms

Table 18: FPE-cipher benchmark

The next benchmark shows how long it takes to build a string message space which contains all elements according to the given regular expression. In the background, the regex has to be converted to an automaton (DFA) which takes the main part of the time. To optimize the speed of an application using our library, all regex could be pre-converted to a DFA and saved as DFA-object, so this step would not be necessary every time building the string message space.

Regex	Message Space Size	Time to convert Regex in DFA and build String MS		
		PC	Notebook	Smartphone
[a-z]{1}	Tiny-Space (5 bits)	11.6 ms	184.0 ms	1.5 ms
[a-z]{1,22}	Small-Space (104 bits)	29.5 ms	245.4 ms	227.2 ms
[a-z]{1,500}	Large-Space (2351 bits)	559.9 ms	3'080.0 ms	100'582.9 ms

Table 19: Message space building benchmark

As can be seen, on the smartphone the large-space message space has over one and a half minute until it is built. The reason for this huge difference to the speed of the other devices is the lack of free memory. Because of that, the Java Virtual Machine on Android has to clean the heap continuously and reallocate the memory during the conversion and this takes a lot of time.

5.2.2 Possible Improvements

The encryption algorithms itself should not be changed and most likely there is no potential for optimization. However in the implementation there are several possibilities how a step is actually implemented and consequently it can be faster or slower.

A possible strategy is to analyze the code at execution time with a profiler, which measures runtime and usage of the varying methods. One could then focus on the methods which are called often or use a lot of CPU time and try to optimize it.

Because Java Code optimization is big and a complex theme and not the focus of our work, it is not treated further here.

5.2.3 Already Improved

One improvement we already made for FFX. It is an improvement of the round function F which is described in its documentation [3]. The algorithm in the paper contains in line 202 a value P, which is static in every round, when the length of the input and tweak does not change. In our implementation this is the case so we could pre-compute it outside the round function F and save one AES call for every Feistel round.

We compared the runtime with and without optimization by encrypting and decrypting a value several times. Due to the outsourcing an improvement of about two-thirds of the original time could be accomplished:

Operation	P inside round function	P outside round function	Time reduced by
encryption	7'144 μ s	2'620 μ s	63%
decryption	6'506 μ s	1'764 μ s	73%

Table 20: Comparison P inside and outside round function

5.3 Security

Our library will be used to achieve confidentiality by protecting information values against unauthorized reading. The only possibility to gain access to the plaintext of this encrypted data is to decrypt it with the same secret key used for the encryption. By implication this means, that if someone wants to extract the content of encrypted data without being in possession of the secret key, he has to attack our library or the algorithms used in our library.

For each cryptographically algorithm a mathematically prove is provided which claims that an algorithm cannot be broken under a certain amount of time. That applies also for the algorithms we used, where this security proof can be found in the corresponding specification of FFX [3] and EME2 [14]. As non-mathematicians and user of these algorithms we can only assume that these proofs are correct, because they are very hard to witness. P. Rogaway mentioned once that his FPE ciphers are secure as long as the underlying block cipher has no flaws. As recommended we only use AES as underlying block cipher in our library and according to Wikipedia there are no known practical attacks that would allow anyone to read correctly implemented AES encrypted data. [24]

We also did specific research on the Internet about the algorithms we used and it seems that no vulnerabilities were found or at least published until today. The only thing we saw is the possible side-channel attack under certain circumstances for EME2 as already mentioned in the theory of EME2. [16]

A very important fact is that all these mathematical proofs are only valid on the condition that these algorithms are correctly implemented. This is not as simple as it sounds, because if one implements an encryption algorithm according to a pseudo code of a definition paper and tries this code with some test-plaintexts, there is no possibility to see if the resulting ciphertext is good or not. Of course the ciphertext will look like random data, but is it that random data that the algorithm with the certain key and tweak should produce by this given plaintext? This question can only be answered by the help of test vectors. In this case we would have needed a list of plaintext and the resulting ciphertext, given by the developer of the algorithms. Unfortunately it was not possible to find such test vectors for FFX-A2 neither for EME2 on the Internet.

Consequently the only tests we could make were unit tests to test our code we have written on basic errors. We have made unit tests for every class which can be looked up in detail in the code of the library.

6 Tutorial – Apply FPE in an Existing Context

The following chapter should give an idea how to apply FPE in an existing context on the basis of a popular use case: encrypting a database without the need of a database redesign. First, we consider the steps to realize this encryption and in a second part we look at the actual program code of an exemplary implementation.

6.1 Use Case

As use case we continue to think about the case of the American bank JPMorgan Chase, where criminals had stolen millions of customer records, which made us wonder if the theft could have been avoided with the encryption of the database.

In the following section, we consider the case in which we want to encrypt the data ourselves and do not want to hand it over to the database. Many of the established database management systems would provide solutions to encrypt the stored data. Some solutions are transparent for the application – that is the application does not have to be adapted – and some are not. We do not discuss the advantages and disadvantages of encrypting the data directly in the database any further. Some critical points could be:

- MySQL does not support Transparent Data Encryption but only some encryption functions which have to be integrated directly into the query. But from version 4.0.2 (July 2002), where the encryption functions were introduced, to version 5.6.17 (March 2014), only the AES-ECB mode was supported [41].
- In Microsoft SQL-Server, Transparent Data Encryption is not available in the standard-, web- and express-editions and in Oracle one needs the costly advanced security option [42].
- Microsoft SQL-Server relies on the Windows Operation System Data Protection API (DPAPI) to store the master key. So we not only need to trust the database management system for the key management but also the operation system [43]. Passcape.com [44] describes an attack where it is under certain circumstances possible to decrypt DPAPI protected data with an administrator account respectively physical server access.

But we now move on and consider the consequences of what an implementation with FPE might entail.

6.2 Current Application

To comprehend the steps needed of an FPE implementation we developed a simple web application that stores its data unencrypted in a MySQL database. It is a simplified replica of JPMorgan Chase's online banking. It provides limited functions. On the other hand it is clear, easy to understand and a good starting position for a tutorial as a result.

The web application is written in Java (which is a prerequisite to use the library) and is based on the Spring MVC 4 Maven archetype [45]. It uses Java Persistence API 2 (Hibernate, Spring Data JPA) in the persistence layer and MySQL as the database management system behind it. The demo application can be found on GitHub: <https://github.com/Rizja/JPMorganDemo/>. There is one version without and one with implemented encryption.

This application appears as follows:
On the first page the client is able to sign in to his private banking account.
With the user 'guest' and password 'demo' there is some sample data available.

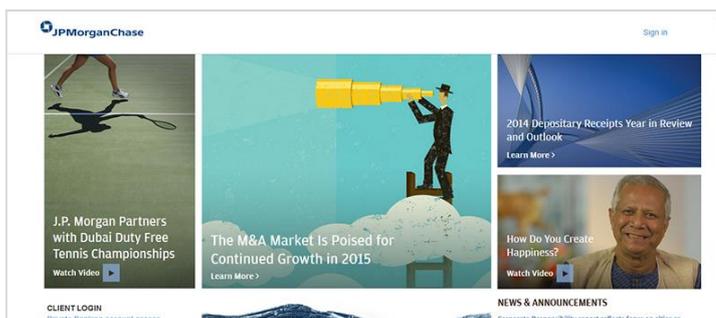


Figure 13: Start page of the application

A logged in client sees his assets. An account has a *name* chosen by the client, a *type*, a *bank number*, a *currency* and a *balance*.

WELCOME BACK, GUEST!		OVERVIEW OF MY ASSETS	
We care about your financial well-being! We look forward to receiving your enquiry and will be happy to offer you personal advice to suit your individual needs. Call your customer advisor or arrange an appointment online.			
■ PAYMENTS	Transaction Account CH7909000000302663071	CHF 5,843.20	Details
■ RESERVE TAX	Savings Account CH7909000000302663098	CHF 4,008.65	Details
■ SAVINGS	Call Deposit CH7909000000302663018	CHF 23,500.00	Details

Figure 14: Overview of the client's assets

With one click on the detail links of an account, one can see the transactions in which this account is involved. With the transactions-menu link it is possible to see the transactions of all accounts.

A transaction has a *timestamp*, a *source* and *destination-account number*, an *amount*, a *currency* and a *message*.

Date	Source	Destination	Amount	Message
2015 Mär. 26	02:00	CH7909000000302663018	CHF 500.00	carry-over wage
2015 Apr. 01	07:12	DE7409000000640840237	EUR 21.30	Amazon alarm clock
2015 Apr. 26	02:00	CH7909000000302663018	CHF 500.00	carry-over wage
2015 Mai. 26	02:00	CH7909000000302663018	CHF 500.00	carry-over wage
2015 Jun. 26	02:00	CH7909000000302663018	CHF 500.00	carry-over wage
2014 Jun. 26	03:00	CH7909000000232753945	CHF 4,528.00	Wage

TRANSACTIONS FOR ACCOUNT «RESERVE TAX»				
Date	Source	Destination	Amount	Message
2014 Jun. 16	08:18	CH7909000000302663071	CHF 2,000.00	reserve tax
2014 Dez. 13	12:34	CH7909000000302663071	CHF 2,000.00	reserve tax

Figure 15: Transactions for all accounts

6.3 Analysis

If we want to enrich our application with FPE, we have to ask ourselves some questions. First, we have to decide where in the existing code we want to integrate the library. Then we have to analyze the data types we use and decide for each the domain respectively message space we want to allow. Finally, we have to think about the way in which we encrypt the existing data.

6.3.1 Where to Put the Encryption Layer

In an application which uses a database there is often a persistence layer which abstract the database access. In our demo application it is Hibernate which provides us with a transparent database access. It reads data from the database without needing us to write database queries and supplies them to us already as Java objects.

There are now several places in the persistence- or a higher layer where we can integrate the encryption. Therefore encrypt the data which is supplied to the database, and decrypt the data which comes from the database. One approach might be to only modify the part of the code we know well. Another approach is to alter as little of the code as possible; Or to leave the encryption as transparent as possible to the application.

In our case we decided to integrate the encryption directly into the persistence layer. It has the advantage that it is almost transparent for the application and that only slight changes in the code are needed. We integrate the support by means of Hibernate user types. User types are custom data types. Hibernate allows us program our own data types by implementing the UserType interface. In the implemented methods which sets respectively gets a value we then inject our FPE functionalities.

As an example we wrote a custom type Balance for the encryption of a bank account balance by implementing the interface UserType. In our BankAccount class we can use it simply by annotating it:

```
@Type(type="ch.bfh.fpelib.jpmorgan.encUserTypes.Balance")
private BigDecimal balance;
```

Thanks to this user type we get the decrypted data in our application (BankAccount class) and Hibernate gets the encrypted data to store in the database.

6.3.2 Data Types

In a next step we have to analyze data types we use and specify a message spaces for each. We have to do this concretization because the domain of the Java data types does not correspond to the domain we need in FPE respectively that we want to allow. For example, strings in Java have a theoretical maximum size of $2^{31} - 1$ characters. Because we do not want strings of this size, we have to decide which domain we actually want to allow. As for example small- and upper-case letters up to ten digits. We then domain is set, we can choose an appropriate message space and FPE cipher from the library.

In our demo application we have decided to use the following domains, message spaces and ciphers:

Value	Domain	Message Space	FPE-Cipher
Account class (entity to store a client)			
Long id	Unique id generated by Database. Not encrypted because no secret.		
String email	Regex: “[0-9a-zA-z.-@]{1,20}”	String	RankThenEncipher
String password	Not encrypted because already hashed.		
String role	Regex: “ROLE_(USER ADMIN)”	String	RankThenEncipher
List<BankAccount> accounts	Not encrypted because BankAccount.id is not encrypted either.		
BankAccount class (entity to store a bank account)			
Long id	Unique id generated by Database. Not encrypted because no secret.		
String name	Regex for one or two words: “([A-Z]?[a-z]{1,10})([A-Z]?[a-z]{1,10})?”	String	RankThenEncipher
AccountType type	Enumeration of account types	Enumeration	RankThenEncipher
String number	IBAN (see following section 6.3.2.1)	String	RankThenEncipher
String currency	Regex for ISO 4217 country codes	String	RankThenEncipher
BigDecimal balance	Regex: “(+-)?[0-9]{1,12}(.[0-9]{1,4})?”	Integer	FFXCipher
Transaction class (entity to store a bank transaction)			
Long id	Unique id generated by Database. Not encrypted because no secret.		
String srcAccount	IBAN (see following section 6.3.2.1)	String	RankThenEncipher
String destAccount	IBAN (see following section 6.3.2.1)	String	RankThenEncipher
BigDecimal amount	Regex: “(+-)?[0-9]{1,12}(.[0-9]{1,4})?”	Integer	FFXCipher
String currency	Regex for ISO 4217 country codes	String	RankThenEncipher
Timestamp date	Regex for a timestamp [47]	String	RankThenEncipher
String message	Regex for one to four words: “([A-Z]?[a-z]{1,12})([A-Z]?[a-z]{1,12}){0,3}”	String	RankThenEncipher

Table 21: Domains, message spaces and ciphers of the demo application

6.3.2.1 Example of a complex data type – IBAN

We choose the IBAN (International Bank Account Number) as account number because it is well known here in Switzerland and a good example for a complex data type. But in America the IBAN is not used so it is not completely appropriate for our JPMorgan example.

The IBAN consists of up to 34 alphanumeric characters, as follows [H]:

- **country code** using ISO 3166-1 alpha-2 - two letters,
- **check digits** - two digits, and
- **Basic Bank Account Number (BBAN)** - up to 30 alphanumeric country-specific characters.

In this example we choose the requirement that within FPE an IBAN is mapped again to a valid IBAN of the same country. Therefore we do not want to encrypt the country, only the checksum should be valid again.

For an encryption we do the following steps:

1. Split the IBAN in the three parts: country code, check digits and BBAN.
2. Encrypt the BBAN with FPE based on the country specific format. Here we do another simplification: we only consider the length. But not every BBAN is valid because it contains a national bank code and other information depending on the country and every country has its additional regulations and exceptions.
3. Finally we generate the two check digits on the BBAN and compose the IBAN again with the unaltered country code, the computed check digits and the encrypted BBAN. The check digits are calculated as follows [H]:
 - a. Check that the total IBAN length is correct as per the country. If not, the IBAN is invalid
 - b. Replace the two check digits by 00 (e.g. CH00 for Switzerland)
 - c. Move the four initial characters to the end of the string
 - d. Replace the letters in the string with digits, expanding the string as necessary, such that A/a = 10, B/b = 11, and Z/z = 35. Each alphabetic character is therefore replaced by 2 digits
 - e. Convert the string to an integer (i.e. ignore leading zeroes)
 - f. Calculate mod-97 of the new number, which results in the remainder
 - g. Subtract the remainder from 98, and use the result for the two check digits. If the result is a single digit number, pad it with a leading 0 to make a two-digit number

For the decryption we proceed identically.

6.3.3 Encryption of Current Data

This final consideration could be omitted if we do not have existing data or do not want to use it. But normally it is the case that we continue to use our current data. So we have to encrypt it. At the moment the data is stored unencrypted in the database but we modified our application in the manner that it expects the data coming from the database to be encrypted.

We now have two possibilities to encrypt the existing data.

6.3.3.1 Scenario 1 – Without Modification / With Downtime

In the first scenario we proceed as follows: (1) we stop the application, (2) deploy our new version with FPE integration, (3) encrypt the whole database and then (4) start the application again.

This scenario has the advantage that there are no modifications in the database needed and fewer modifications in the application code than in the second scenario.

On the other hand we have the disadvantage that we need a separate application for migration respectively encryption of the data. Another point is that there is a longer downtime than in the second scenario because the application has to be stopped during the encryption as we do not know which data is encrypted and which not.

6.3.3.2 Scenario 2 – With Tiny Modifications / Without Downtime

In the second scenario we perform the following steps: (1) we alter the database table and add the possibility to say if it is encrypted or not for every entity, (2) deploy our new version with FPE integration. The application then determines if the entity read from the database is encrypted or not. If it is unencrypted it can be handed over without modification. If it is encrypted, it has to be decrypted first. When the data is saved to the database it is encrypted in every case and the encryption flag is set.

This scenario has the advantage that there is no separate application for the encryption needed and there is a short or no downtime while deploying the new application.

The disadvantages are that there are tiny modifications needed in the database. But the adding of a database column could be done without interruption and if the application accesses the database table with referring to the column names and not with "SELECT *", it should not notice the change. The other disadvantage is that the application has to be extended with the logic to determine between encrypted and unencrypted. Last but not least the data is unencrypted in the database until it is read and written again by the application.

This scenario could also be used to support other applications without FPE which generate data (write-only). They can insert data without setting the encrypted flag and our application with FPE is able to handle this new unencrypted data.

For our demo application we decided in favor of this second scenario because of its advantages.

6.4 Application extended with FPE

6.4.1 Changes in the Code

We now look at the actual changes needed in the code. As we already stated – thanks to the Hibernate user types – there were almost no alterations needed. We only need to annotate the current fields of the entities we want to encrypt with our own data types.

As an example in the BankAccount class the following changes were made:

```
@Id
@GeneratedValue
private Long id;

+ // encrypted = true, because always decrypted as soon as in application
+ // only false when unencrypted in database before read/written first time
+ private boolean encrypted = true;

+ @Type(type="ch.bfh.fpelib.jpmorgan.encUserTypes.BankAccountName")
private String name;

+ @Enumerated(value=EnumType.STRING)
+ @Type(type="ch.bfh.fpelib.jpmorgan.encUserTypes.BankAccountType")
private AccountType type;

+ @Type(type="ch.bfh.fpelib.jpmorgan.encUserTypes.IBAN")
private String number;

+ @Type(type="ch.bfh.fpelib.jpmorgan.encUserTypes.Currency")
private String currency;

+ @Type(type="ch.bfh.fpelib.jpmorgan.encUserTypes.Amount")
private BigInteger balance;
```

The new lines are marked with a plus sign. Beside the annotations we also introduced a new field to decide if the record is encrypted or not. This field is named after the new corresponding database column so that it is automatically mapped. The field is initialized with true (=record is encrypted) because the record is always encrypted when written into the database. When the record is read from the database, Hibernate overwrites the field with the value from the database. Thereby we know if we have to decrypt it or not.

For our custom data types we made a package with the name encUserTypes.

The two abstract classes EncryptedInteger and EncryptedString are implementations of the Hibernate UserType interface and provide the basic functionalities. The other classes in this package are subclasses of these two and are the one which are effectively used as data types.

The two interesting methods are nullSafeGet and nullSafeSet for getting or setting a value. In both we integrated the decryption respectively encryption functionalities.

The key used for the encryption has to be supplied to the Tomcat webserver at startup by setting a property. The secure storage of this key could be a bachelor thesis on its own and is not covered here.

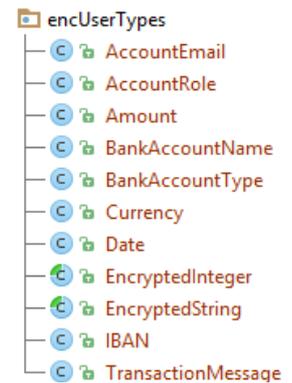


Figure 16: Package encUserTypes

```
public abstract class EncryptedString implements UserType {

    protected static final Key KEY = new
        Key(System.getProperty("master_key").getBytes(Charset.forName("UTF-8")));
    protected static final byte[] TWEAK = new byte[0];
    private FPECipher<String> encryption;

    public EncryptedString(FPECipher<String> encryption) {
        this.encryption = encryption;
    }

    public Object nullSafeGet(ResultSet rs, String[] names,
        SessionImplementor spi, Object owner) throws SQLException {
        String value = rs.getString(names[0]);
        if (value == null) { // if NULL in database return null
            return null;
        }
        int encColInd; // find column 'encrypted' from result set
        for (encColInd=1;encColInd<=rs.getMetaData().getColumnCount();encColInd++){
            if (rs.getMetaData().getColumnName(encColInd).equals("encrypted"))
                break;
            else if (encColInd==rs.getMetaData().getColumnCount())
                throw new RuntimeException("Encrypted field in entity missing.");
        }
        // only decrypt if 'encrypted' flag is set
        if (rs.getBoolean(encColInd)) value = decrypt(value);
        return value;
    }

    public void nullSafeSet(PreparedStatement st, Object value, int index,
        SessionImplementor spi) throws SQLException {
        if (value != null) {
            String encrypted = encrypt(value.toString());
            st.setString(index, encrypted);
        } else {
            st.setNull(index, Types.VARCHAR); //if NULL in Java set to NULL in db
        }
    }

    ...

    private String encrypt(String value) {
        return encryption.encrypt(value, KEY, TWEAK);
    }

    private String decrypt(String value) {
        return encryption.decrypt(value, KEY, TWEAK);
    }
}
```

One of the subclasses of EncryptedString is Currency. It encrypts a currency which is part of the 164 currencies in ISO 4217. We mapped the list to a regex. The Regexp is - with the help of a StringMessageSpace - supplied to the RankThenEncipher FPE cipher. The variable can be declared as static so that the performance intensive creation of the message spaces had to be done only once.

The RankThenEncipher instance is then passed via constructor to the super class where all other functionalities reside.

```
public class Currency extends EncryptedString {

    private static final String REGEXP =
        "A(ED|FN|LL|MD|NG|OA|RS|UD|WG|ZN)|B(AM|BD|DT|GN|HD|IF|MD|ND|OB|OV|RL|SD|TN|WP|YR|
        ZD)|C(AD|DF|HE|HF|HW|LF|LP|NY|OP|OU|RC|UC|UP|VE|ZK)|D(JF|KK|OP|ZD)|E(GP|RN|TB|UR)
        |F(JD|KP)|G(BP|EL|HS|IP|MD|NF|TQ|YD)|H(KD|NL|RK|TG|UF)|I(DR|LS|NR|QD|RR|SK)|J(MD|
        OD|PY)|K(ES|GS|HR|MF|PW|RW|WD|YD|ZT)|L(AK|BP|KR|RD|SL|YD)|M(AD|DL|GA|KD|MK|NT|OP|
        RO|UR|VR|WK|XN|XV|YR|ZN)|N(AD|GN|IO|OK|PR|ZD)|OMR|P(AB|EN|GK|HP|KR|LN|YG)|QAR|R(O
        N|SD|UB|WF)|S(AR|BD|CR|DG|SP|EK|GD|HP|LL|OS|RD|TD|VC|YP|ZL)|T(HB|JS|MT|ND|OP|RY|T
        D|WD|ZS)|U(AH|GX|SD|YI|YU|ZS)|V(EF|ND|UV)|WST|X(AF|CD|OF|PF)|YER|Z(AR|MW|WL)";

    private static final FPECipher<String> fpeCipher =
        new RankThenEncipher<>(new StringMessageSpace(REGEXP));

    public Currency() {
        super(fpeCipher);
    }
}
```

To the other data types we refer to as [F]. They are constructed in the same manner except the IBAN type which is a little bit more complicated due to the various message spaces, the partial encryption and the checksum.

6.4.2 Result

In the following section we see the outcome of our changes. OVERVIEW OF MY ASSETS

On the right we have a screenshot from the web application in which we see three bank accounts. At the bottom we have a database extract with the same three accounts.

At first glance we do not see that the data is encrypted. Looking at it in more detail we see that some columns have odd or unrealistic values (beside the column with the name encrypted). But in this case we did not aim to deceive an intruder but we intend to encrypt the database with almost no changes - and this aim we achieved.

	PAYMENTS <i>Transaction Account</i> CH7909000000302663071	CHF 5.843,00 Details
	RESERVE TAX <i>Savings Account</i> CH9300762011623852957	CHF 4.008,00 Details
	SAVINGS <i>Call Deposit</i> CH6309000000250097798	CHF 23.500,00 Details

Figure 17: Decrypted Data in Web Application

```
mysql> select * from bankaccount;
+-----+-----+-----+-----+-----+-----+-----+
| id | balance      | currency | encrypted | name          | number                | type      |
+-----+-----+-----+-----+-----+-----+-----+
| 1 | 384431263802 | SAR      | ☉          | Elwctthdhrq  | CH3665600779400830240 | TIME     |
| 2 | 113727532659 | SAR      | ☉          | Vchhzvjgnxg  | CH3396522560114631366 | SAVINGS  |
| 3 | 197965321587 | SAR      | ☉          | Iqvnfzbszcb  | CH2401362398124426232 | CALL     |
+-----+-----+-----+-----+-----+-----+-----+
3 rows in set (0.01 sec)
```

Table 22: Encrypted Data in Database

Finally we demonstrate the advantage of the additional database column "encrypted".

Thanks to FPE the data can be processed by any other application without FPE – when the application does not need to understand the data. But it is also possible that an application without FPE produces data. It can write to the database, ignoring the encrypted flag. In our example an application could generate a transaction with the following statement:

```
INSERT INTO transaction (date, sourceAccount, destAccount, amount, currency, message) VALUES
("2016-01-01 12:00:00.0", "CH3396522560114631366", "CH9300762011623852957", 12000, "CHF", "Interest");
```

In the database we see the new unencrypted entry...

id	encrypted	date	destAccount	amount	currency
10	⊗	2080-02-22 06:38:02.0	CH2401362398124426232	16655240448	SAR
11	⊗	1957-04-30 05:08:56.0	CH2401362398124426232	16655240448	SAR
12	⊗	2060-08-14 10:44:49.0	CH2401362398124426232	16655240448	SAR
13	⊗	1965-03-02 12:29:31.0	CH3665600779400830240	510900202752	SAR
14		2016-01-01 12:00:00.0	CH9300762011623852957	12000	CHF

...and in the web application the new record is handled correctly:

TRANSACTIONS FOR ACCOUNT «RESERVE TAX»

Date	Source	Destination	Amount	Message
2014 Jun. 16 08:18	CH7909000000302663071	<i>this account</i>	CHF 2.000,00	reserve tax
2014 Dez. 13 12:34	CH7909000000302663071	<i>this account</i>	CHF 2.000,00	reserve tax
2016 Jan. 01 12:00	CH3396522560114631366	<i>this account</i>	CHF 12,00	Interest

Figure 18: Web application successfully decides between encrypted and unencrypted data

The data is not encrypted until it is written to the database the next time.

7 Demo App

So far we have seen a lot about the theory of FPE, developed and documented a Java FPE-Library and delivered a detailed tutorial on how to implement our FPE-library into an existing environment. The last goal missing in this bachelor thesis is the implementation of an existing application which is enriched with Format Preserving Encryption.

The goal of this demo is to prove that it is possible to extend an existing, widely used application with FPE, respectively with our developed FPE-Library, with only few changes in the existing code and a small amount of work.

To achieve this goal we decided to work with Android smartphones. We came to this decision because we both have no experience in mobile developing or the development of Android apps and this should be a challenge that allows us to collect experience in this area. Furthermore, compared to personal computers or notebooks, smartphones just have a small amount of system resources and we are very interested to see how much our FPE-extension will influence the app's speed. Last but not least, smartphones provide a lot of very interesting use cases where FPE can be applied. Just think about some basic services running on them like calendar, address book, maps, even for pictures stored on a smartphone a FPE encryption could be used. In the address book all contacts including names, address information, phone numbers etc. could be encrypted with FPE and other apps maliciously accessing these information would not be aware that they are encrypted. In the calendar one could encrypt the dates of appointments with FPE resulting that all events would be on different days on different times and only the user with the key would be able to see the real appointment date.

We discussed a lot what use case we should cover with our app and during this decision-making process our thesis advisor Reto Koenig suggested taking an Android app called "OwnTracks" he is using in his courses.

7.1 OwnTracks

OwnTracks is an open-source tracking app available for iOS and Android. It allows recording GPS-positions and saving them as locations on the smartphone as well as sharing them with other people. It is also possible to define so-called geofences and to receive an alert if another user is entering this defined area.



Figure 19: OwnTracks Logo

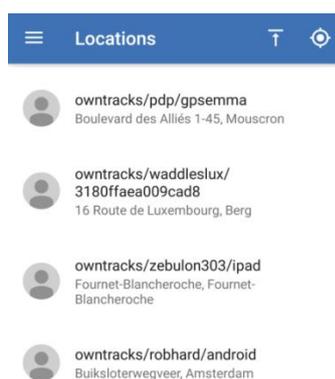


Figure 20: OwnTracks Locations

The left picture shows the main window of OwnTracks where all the received locations from other users can be seen.

By clicking on one of these locations, the map (as shown on the right picture) appears and shows where all the users are currently positioned.



Figure 21: OwnTracks Map

OwnTracks has a similar purpose as a former popular project of Google with the name “Latitude” which was closed down in August 2013. For Android smartphones this app can be found for free on the Google Play Store in the present version 0.4.21. The source code of OwnTracks is hosted as repository on GitHub where the current Android release of OwnTracks is available on version 0.5.11. Even though this release is not yet published and has some annoying bugs in it, this is the version we forked from GitHub and used as base to implement our FPE feature. But before we talk about how we integrated FPE into OwnTracks, we should first have a look at how OwnTracks sends and receives published locations.

7.2 MQTT

For sharing locations with others, OwnTracks uses an open communication protocol called MQTT. MQTT is a machine-to-machine (M2M)/“Internet of Things” connectivity protocol originally developed by IBM. It was designed as an extremely lightweight publish/subscribe messaging transport. It is useful for connections with remote locations where a small code footprint is required and/or network bandwidth is at a premium.[38]



Figure 22: MQTT Logo

To be able to use MQTT, a so-called MQTT-broker is needed. The client, in our case the OwnTracks app, connects to this broker and publishes data to specific topics which are hierarchically structured with branches. OwnTracks uses the topic branch “Owntracks/<username>/<device_id>” as default. A MQTT-broker can be private or public. If a broker is public, everybody can read the topics, in this case the current locations, and also publish their own data to every topic. There is no possibility to give any access restriction for example with an authentication on a public broker. Whereas a private broker is set up on an own infrastructure and one has full control over everything. But the operation of a private MQTT-broker is connected with cost and effort and in most scenarios it does not make any sense to use one. The average user of OwnTracks will use a public MQTT-broker as provided, for instance, by the Eclipse IoT Working Group (iot.eclipse.org) or by Mosquitto (test.mosquitto.org).

Normally, there is no restriction on MQTT-broker on what kind of data is allowed to be stored. OwnTracks, as well as most of the applications using MQTT, sends the actual data as JSON strings to the broker. In our case the payload includes the coordinates and timestamps.

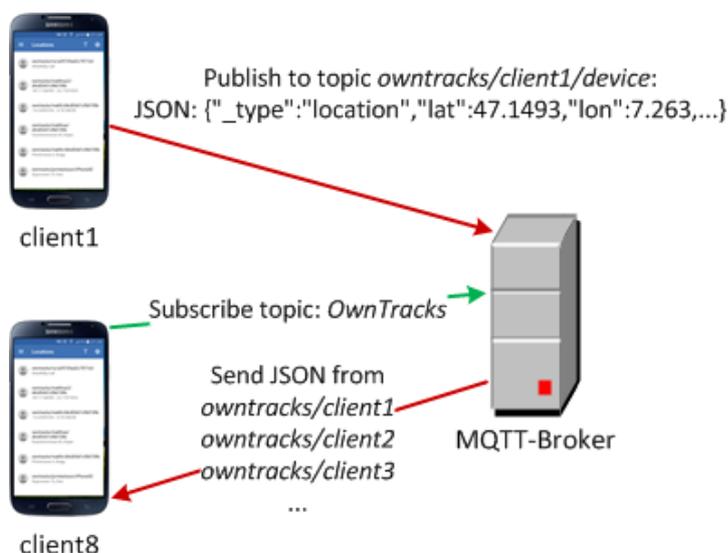


Figure 23: Communication between OwnTracks clients and MQTT-Broker

7.3 Lack of Security

As mentioned above, everybody is able to subscribe a topic and to track the location of every publisher of this topic by using a public MQTT-broker. Currently there is no mechanism in OwnTracks to prevent this. The only security feature OwnTracks has at this moment is the possibility to secure the connection to the MQTT-broker with a TLS- or a VPN-connection. This only helps in preventing attacks like sniffing or replay attacks and does not give complete confidentiality.

To provide confidentiality, meaning only people who are allowed by the publisher to know a location are able to effectively see the correct GPS-coordinates, encryption is needed. However, applying encryption to an existing application involves the risk that a lot of the existing code has to be changed so that everything still works. There might also be the problem that older version of an application are not able to handle the encrypted data. In the case of OwnTracks, the service which manages the outgoing and incoming location messages expects a certain format for the longitude, the latitude, the altitude, the timestamp and so on. A normal AES encryption would convert these values to a multiple 16-byte strings, probably encoded with Base64 and the location service of older versions of OwnTracks could not handle them anymore. We have not explicitly tested how OwnTracks would react to such a change of the input data. Possible scenarios are that OwnTracks will completely crash or just ignore the incoming location messages it cannot handle.

However, at this point Format Preserving Encryption comes into play. FPE allows encrypting the values in the location message to values with the same format, so that the service will treat the message as normal location, without recognizing that the values are encrypted. Hence, there is no chance that any error will occur. Only if the correct key is given to the decryption operation, the real coordinates will be shown.



Figure 24: Content of the JSON in clear text, encrypted with AES or FPE

7.4 Implementation

Our main goal during the implementation of the FPE feature into OwnTracks was to change as little as possible in the existing code and on the user interface of the app.

The version of OwnTracks with our implementation of Format Preserving Encryption can be found at: https://github.com/Rizja/owntracks_android

7.4.1 GUI

By implementing a feature like encryption in an application it is important to give the user the possibility to enable and disable such a function. We did this by adding an „Encryption“-switch in the options under Preferences – Connection – Security. In this submenu only a switch for enabling TLS was present and we decided there is enough space left on the screen to add our button as it is a good location for such a feature.

As a next step, a user has to be able to add passwords for the decryption of other user’s location information. For this purpose we added a “Decryption Passwords“-button in the same place, as shown in the following graph:

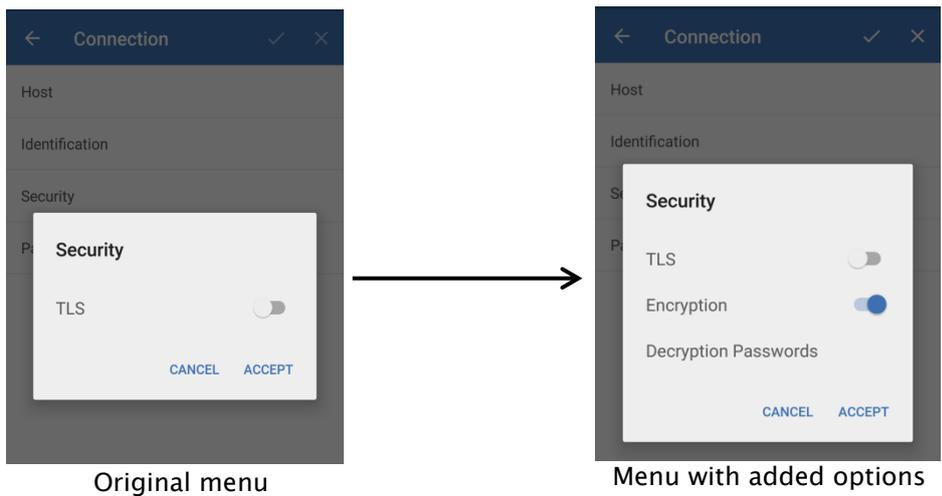


Figure 25: Original Security Menu compared to our additions

The entries of this submenu are the only things we changed on the existing GUI. All other dialog windows we added are under the submenu "Decryption Passwords" and are described in detail in the user manual below.

7.4.2 Encryption / Decryption

In OwnTracks every incoming and outgoing message is handled by a service class called ServiceBroker. We added a few lines of code to provide the encryption of outgoing messages:

```

if (Preferences.getEnc() && message.getTopic() != null){ //encrypt payload if fpe is enabled
    String user = message.getTopic().split("/")[1];
    String password = FormatPreservingEncryption.getPassword(user);
    Key key = new Key(password.getBytes(Charset.forName("UTF-8")));
    byte[] tweak = (message.getTopic()+json.optString("tst")).getBytes(Charset.forName("UTF-8"));
    json = FormatPreservingEncryption.encrypt(json, key, tweak);
}
message.setPayload(json.toString().getBytes(Charset.forName("UTF-8")));

```

This addition changes the logic for outgoing messages as follows:

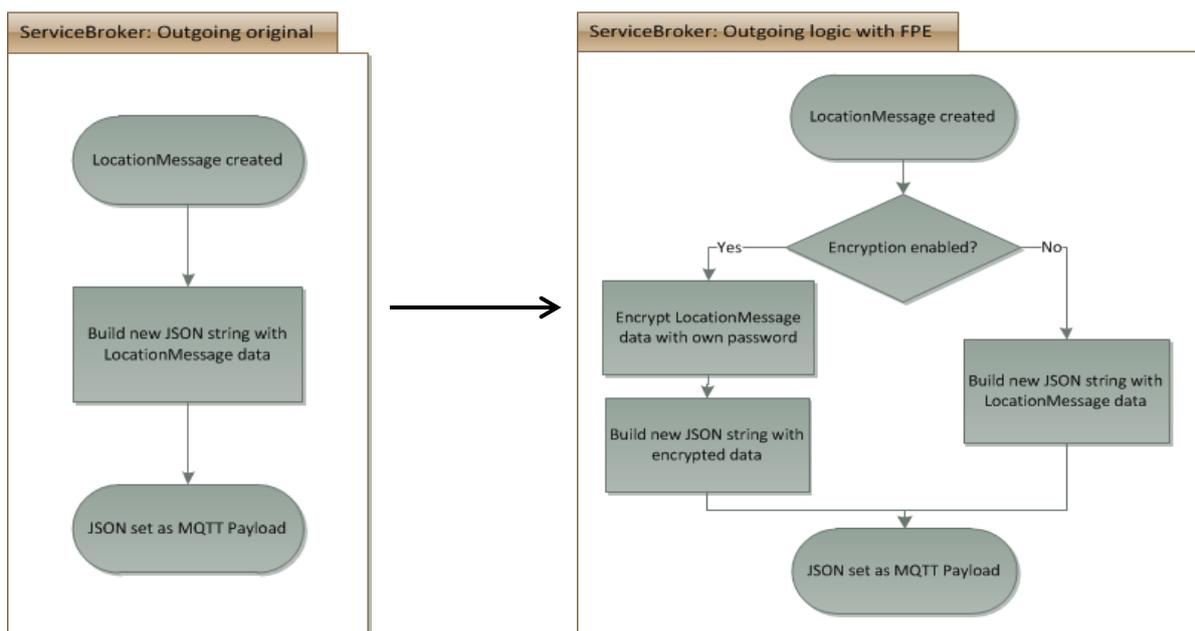


Figure 26: Changes in the logic of outgoing messages

In order that incoming messages are decrypted if a user password is available, we added the following code lines into the ServiceBroker:

```
String user = topic.split("/") [1];
String password = FormatPreservingEncryption.getPassword(user);

if (password != null) { //decrypt if key available for publisher of message
    Key key = new Key(password.getBytes(Charset.forName("UTF-8")));
    byte[] tweak = (topic+json.optString("tst")).getBytes(Charset.forName("UTF-8"));
    json = FormatPreservingEncryption.decrypt(json, key, tweak);
}
```

These lines change the logic for incoming messages as follows:

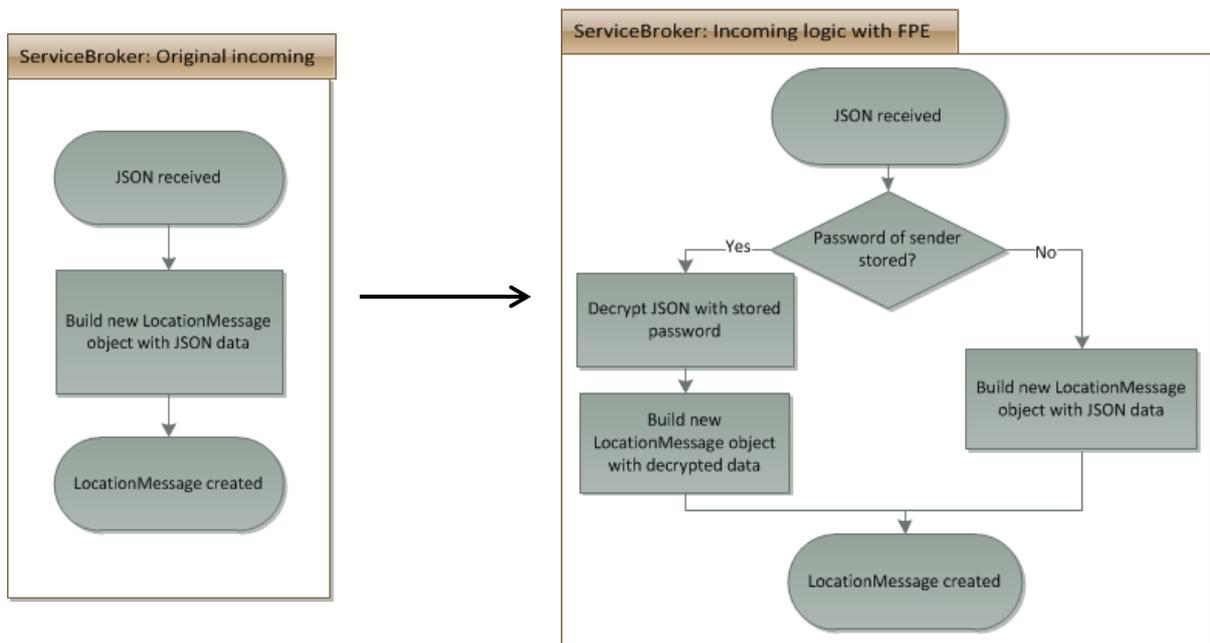


Figure 27: Changes in the logic of incoming messages

It is important to mention that all code we added in this class could easily be removed and the app would work as before just without encryption.

As shown in the code, we use the topic string (“owntracks/<username>/<device_id>”) and the timestamp of the message as tweak. This ensures that every location is encrypted differently for every user, on every device, at every time. Because of that, it is not possible to gain any information about the location of a publisher if he encrypts his coordinates. Even if he publishes the exact same location over a certain amount of time, which would lead an observer to assume that this could be that publisher’s home or working place, every location will be encrypted completely differently. As a consequence for using the timestamp as part of the tweak, this value is obviously not allowed to be encrypted, because the recipient has to know this value to be able to decrypt the location. Considering the fact that an adversary could just observe the time when a location message is published, it is not really a privacy problem to let the timestamp be in clear text.

7.4.3 FPE Helper Class

For the actual implementation of the encrypt- and decrypt-methods as well as the methods for the user password management, we created a helper class with the name FormatPreservingEncryption in the support folder of the project.

In our implementation three types of OwnTrack messages are supported to be encrypted: location, waypoint and transition messages. Any other message type, as for example a beacon or command message will be ignored in this class.

In these supported message types there are various payload element types respectively variables in the JSON string. The following elements are currently supported and will be encrypted, whereas other elements stay in plaintext. Elements which we do not encrypt at the moment, for instance the accuracy of the reported location, are in our opinion not a privacy risk and the encryption of such elements would only cost resources without a good security benefit.

Element	Description	Format
alt	altitude measured in meters above sea level	0-10000
batt	device's battery level in percent	0-100
cog	heading (course over ground) in degrees, 0 = North	0-359
desc	description of a waypoint	String (Base64)
lat	latitude	"-?(90 ([0-9][1-8][0-9])\.\.[0-9]{1,8})?"
lon	longitude	"-?(180 ([0-9][0-9][0-9]1[0-7][0-9])\.\.[0-9]{1,8})?"
wtst	timestamp of the waypoint creation (note: this is not the same timestamp as used in the tweak)	0-4131648000L

Table 23: Encrypted elements in our implementation

All the elements above hold different formats - consequently all these elements need different types of message spaces from which the values can be encrypted with different FPE ciphers. Following a few lines of code out of our helper class which shows how different format types are handled.

The battery level from 0 to 100 percent is a value which can be represented with 7 bits and starts by zero. Therefore our tiny FPE cipher, the Knuth Shuffle, is the optimal cipher for this case.

```
private static final BigInteger BATT_MAX = BigInteger.valueOf(100);
private static final KnuthShuffleCipher BATT = new KnuthShuffleCipher(BATT_MAX);
```

For the altitude above the sea level we defined a maximum of 10'000 meters. This should be enough to cover most of the use cases, even if one would publish a location on Mount Everest. To represent this maximum decimal value in binary numbers, 14 bits are needed. So this is a case for the FFX cipher:

```
private static final BigInteger ALT_MAX = BigInteger.valueOf(10000);
private static final FFXIntegerCipher ALT = new FFXIntegerCipher(ALT_MAX);
```

The longitude is a number from -180 to +180 degrees with fractional digits. For this reason we cannot directly use an integer cipher as we did in the last two examples, but we have to define a string message space with a regular expression on how a longitude has to look like and to use the Rank-Then-Encipher approach:

```
private static final String LON = "--?(180|([0-9]|([0-9][0-9]|1[0-7][0-9]) (\\. [0-9]{1,8})?)");
private static final StringMessageSpace LON_MS = new StringMessageSpace(LON);
private static final RankThenEncipher<String> LON = new RankThenEncipher<>(LON_MS);
```

Descriptions of waypoints are not encrypted with FPE but with a normal AES encryption. We did this because in this case no special format is needed; we just encode the output of the AES encryption with Base64 and got a valid string. This shows also, that even if FPE is implemented as encryption method, it does not have to be used exclusively if there are cases where another method, as for example an AES cipher, is safe enough and provides more speed.

We have declared all message spaces as static constants with the goal that they are initialized respectively built only once and that is by the first use of the helper class. As we could show during the benchmark tests in section 5.2.1, the building of the message space is the most expensive procedure of our library. By declaring them like that, we ensure that this has to be done only once for every start of the app. As a further development one could implement that these messages spaces are only built by the first start of the app and then stored in the SharedPreferences as message space objects.

The encryption-, respectively decryption-method is built with an algorithm which reiterates through all the elements in the given JSON object. If one previously defined element is found, that encryption/decryption is applied. This approach makes it very easy to attach other or new elements that could be implemented in further releases of OwnTracks.

For a closer analysis of what we just described, take a look at the appendix 13.7 where the whole code of the FPE helper class is attached.

7.4.4 Key Storage

One important question in applications which are dealing with passwords is how to store them in a secure manner. This means that only the application itself, in our case OwnTracks, is allowed to read the passwords stored and no other apps.

In the Android operating system a system called Android Keystore exists. In this Keystore private keys can be stored in a container to make it much more difficult to extract them from the device.

However, this Keystore did not work for us. We tried to store our passwords directly or to derive secret keys from the password and store them into the Keystore, but we only received various errors. As it seems in the documentation of the Keystore, it is originally constructed for private/public key pairs and for trusted certificates and not for secret keys used in symmetric encryption as we use them in FPE. Because of that we tried to integrate our derived key as part of a private/public key container (PKCS#12) but that did not work either.

At the end we decided to drop the efforts and to just store the passwords as cleartext in the SharedPreferences of the app. An Android Security Expert gives the following explanation of the security of the SharedPreferences:

“Shared Preferences are stored as a file in the filesystem on the device. They are, by default, stored within the app's data directory with filesystem permissions sets that only allow the UID that the specific application runs with to access them. So, they are private in so much as Linux file permissions restrict access to them, the same as on any Linux/Unix system.

Anyone with root level access to the device will be able to see them, as root has access to everything on the filesystem. Also, any application that runs with the same UID as the creating app would be able to access them. This is not usually done and you need to take specific action to make two apps runs with the same UID, so this is probably not a big concern. Finally, if someone was able to mount your device's filesystem without using the installed Android OS, they could also bypass the permissions that restrict access.” [39]

As a summary we can say the SharedPreferences are not really a secure place for passwords. However, due to the lack of time and the fact that our implementation can be treated as a prototype we decided to leave it like that. For further developments of the FPE feature and plans to send a pull-request to the OwnTracks project, the whole key storing part must be reconsidered again!

7.5 User Manual

This manual gives an instruction on how the encryption feature in OwnTracks can be enabled and how other user's passwords can be stored in the app. Be aware that this manual covers only these points. For a basic user guide of OwnTracks have a look at the OwnTracks Starter Guide [40].

All the pictures shown in this manual are non-modified print screens from the app OwnTracks. No external sources were used.

First a few words about what you are actually doing by enabling the encryption function of OwnTracks.

Let us assume your name is Matthias and you live in Grauholzstrasse 65, Ittigen. By publishing a location report while you are at home, everybody using the same MQTT-Broker as you will see the real coordinates respectively your real position on their maps.

If you enable the encryption of your location as it will be explained in this chapter, all users which use an older version of OwnTracks without the encryption/decryption functionalities or all users which are not in the possession of your password will receive coordinates that are completely wrong.

The following example shows that while you are at home and publish your location, most of the users will think that you are at the east coast of the United States.

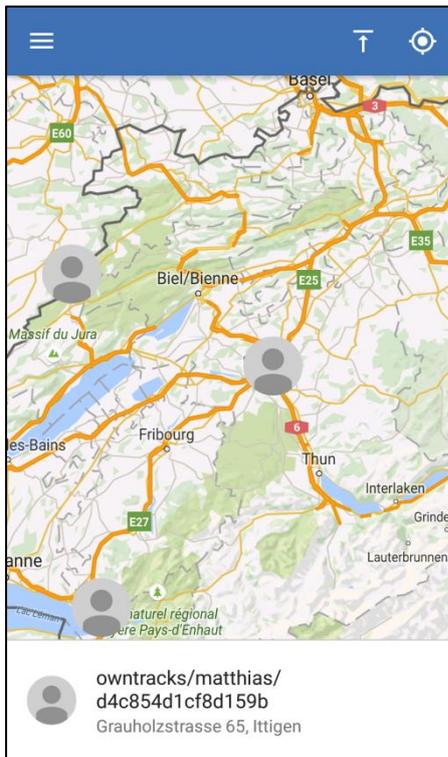


Figure 29: Your location without encryption as seen by all users

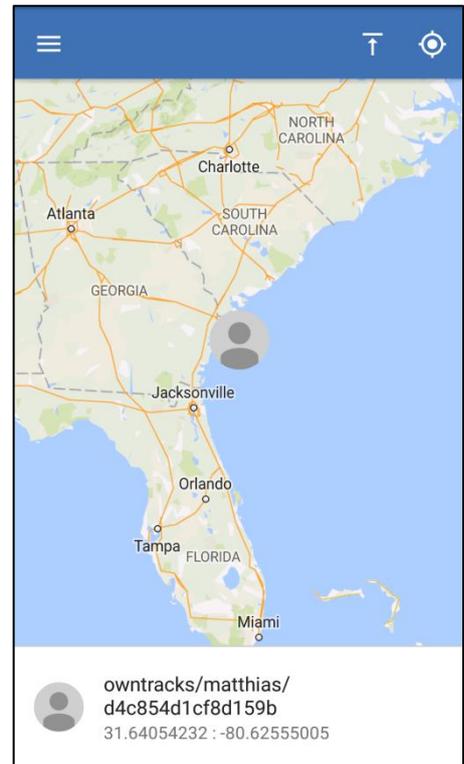
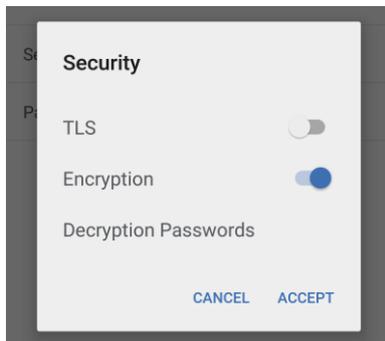


Figure 28: The same location, this time encrypted, seen by all users not in possession of your password

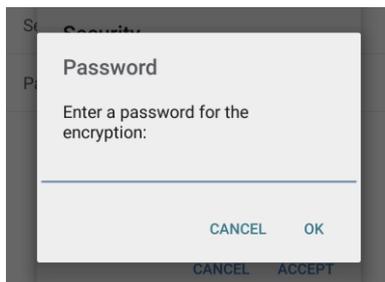
7.5.1 Enable the encryption of locations



Go to the submenu Preferences – Connection – Security.

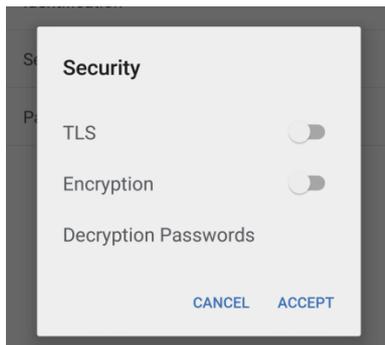
To enable the encryption of your locations, enable the button by switching it to the right.

At the moment where the encryption is activated, the password entering dialog appears.



Here you have to enter a password with which your locations will be encrypted. Be aware that there is no possibility to look up this password again after entering it – you will have to remember it in order to share it with other people.

Congratulations, your shared locations are now encrypted! As a next step you have to share your password with all the people that are allowed to know your real position, so that they are able to decrypt your locations.



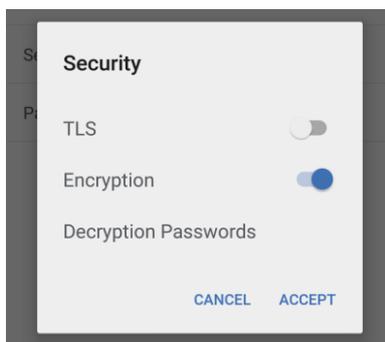
For disabling the encryption, go to the submenu Preferences – Connection – Security and switch the encryption button to the left.

Remember to tell your friends you gave your stored password that you have disabled the encryption; otherwise they will decrypt your no longer encrypted locations and get wrong coordinates.

For changing your encryption password, simply disable and enable the encryption again.

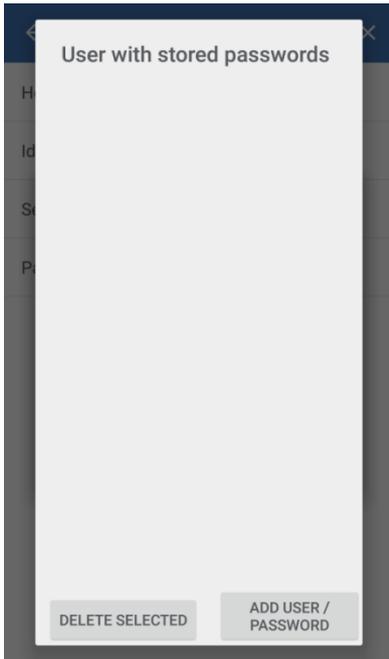
The password entering dialog will appear and you can enter your new password.

7.5.2 Manage password from other users



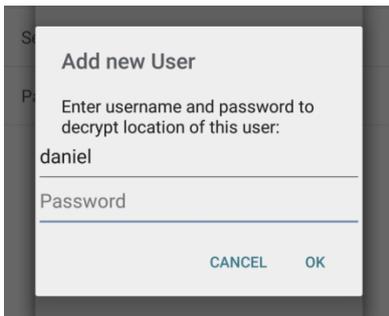
Go to the submenu Preferences – Connection – Security.

Press the button “Decryption Passwords”.

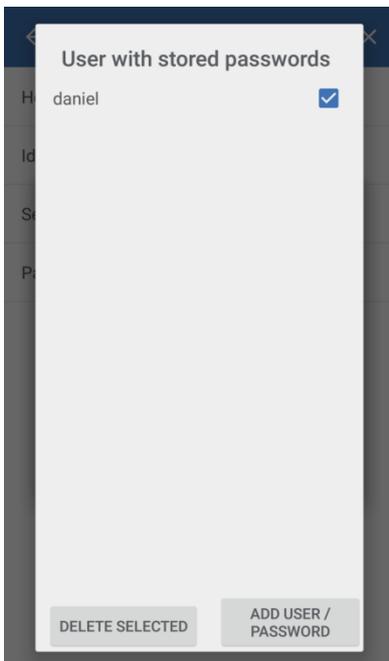


A new dialog with all the already stored user passwords appears. In this example no users are stored yet.

To add a new user, press the button "Add User / Password"



The "Add new User" dialog appears where you can enter the username and the password of the other user.



You have now successfully stored the user Daniel with the password he gave to you.

All locations from Daniel will be decrypted from now on.

If Daniel tells you that he no longer encrypts his locations or if he changes his password, you can simply check the box right to his name and push "Delete selected" at the bottom to remove his password from the list.

Daniel's locations are not decrypted anymore.

8 Summary

We have now seen a number of theoretical as well as practical aspects of FPE. Below there is a short résumé about the subjects considered in this paper.

8.1 Project Management:

Since this thesis is not only about a product but also about the project lead, we documented how we have organized our project in this chapter. We described the roles and the resources. The project did not produce any additional costs. We have listed the detailed project goals for the Java FPE Library, the tutorial and the demo app on Android. All of these mandatory targets including most optional objectives have been met. In addition we attached the original schedule with the deviation which arose. In general, we were able to maintain the schedule. Additionally, the problems are listed and we give a conclusion about the project progress.

8.2 Introduction to Format Preserving Encryption

At the beginning of the chapter, FPE is classified to belong to the field of IT security and it is explained that with FPE only part of cryptography can be covered. For the additional important goals such as integrity, authentication and non-repudiation other constructs must be used.

Using the example of a bank which uses ATMs over leased lines, we demonstrated why it might make sense to encrypt only parts of the message and to preserve the format in the encryption. We discussed the origin of FPE and argued that the idea is not new and over time there have been many different solutions. Next we gave a mathematical definition, namely that a symmetric encryption FPE $E: X \times K \rightarrow X$, where K is the key and X the message space of plain- and ciphertext and E a reversible function, which performs a permutation. We also made a distinction with other related technologies like the format Transforming Encryption (RTD) and the tokenization.

In another section the applications of FPE are shown. One can differentiate between the application where the format is maintained in order to circumvent restrictions by the computer system and the application where the format is maintained to hide the encryption from the user.

8.3 FPE Schemes

In this chapter we show a possible categorization of FPE variants, which was proposed by P. Rogaway: The division on the size of the message space in tiny space (approximately up to ten bits), small space (up to 128 bits) and large-space (for more than 128 bits) message spaces. For the library we have selected and implemented one scheme of each size. This chapter also includes the explanation why we opted for the corresponding schema and how it works.

The tiny space scheme is based on the Knuth shuffle, a technique for mixing elements. We described how we make this process deterministically with a pseudo random number generator (PRNG) to obtain a permutation for the encryption.

For the small space scheme FFX first explain the underlying technology of the Feistel cipher. The balanced Feistel works with an even input length and the unbalanced Feistel which can also handle inputs of odd length. There is also the possibility to work on a non-binary alphabet. Based on this, the definition of the three variants of FFX is given: A2 (binary alphabet), A10 (decimal alphabet) and radix (variable alphabet and allowed message size enlarged).

Under the point of the large space scheme we introduce the two standardized schemes XCB and EME2 and say why we opted for EME2. Both schemes have been standardized by a task force which advocates for safety in the storage area.

Next in this chapter, two generic FPE schemes, which cannot be classified into the above categories, are described. These are cycle walking and rank-then-encipher. Cycle walking is used when the output of a FPE encryption may end up outside the message space, but the probability is rather slim. With the rank-then-encipher scheme a message space is converted into another enumerable message space. These two schemes come into use in the library.

8.4 FPE Library

This chapter provides a technical documentation of the library. After an overview of the classes, a detailed insight into the individual classes is given. Furthermore, the used libraries are listed. The only external library is Dk.brics.automaton which is used to define the string message space with an automaton or regular expression. Then a brief overview of the legal aspects is given: all used components are open source or not protected, except FFX on which HP has a patent. This section also shows the performance of our library. Based on a benchmark it is shown that for a FPE encryption quite a bit more CPU time is required than for a classic AES encryption. Also for the creation of a message space some time must be allowed depending on the size and equipment. At the end of this chapter we take a look at the safety of the library. In conclusion, we can recommend the library in order to achieve an additional security but we cannot recommend it as single solution in an environment with critical data.

8.5 Tutorial

In this chapter it is shown how a subsequent implementation of FPE could look like by way of an example. We had to ask ourselves the question, where the encryption layer is integrated, which Java data types are converted in which messages space and how we deal with existing, unencrypted data. From this use case we have seen that a subsequent encryption of data is not as complicated as one might think and actually worked quite well. We have seen that there is some additional code but almost no changes in the current code and in the database.

8.6 Demo app

The aim of this chapter was to provide an application which uses FPE in order to demonstrate a practical example of use. On the basis of the Android app OwnTracks we show how it is possible to encrypt location data with FPE. The advantage of this solution is that the data can still be saved on a public server, but it can only be read by authorized persons. And all existing apps in which the encryption is not integrated can still deal with these locations; however they are not the real ones.

8.7 Conclusion

In conclusion, we can say that FPE is an interesting technology, to which perhaps too little attention has yet been paid and that in some applications its use would make sense. Our library is, as far as we know, the only implementation in Java. The implementation of FPE into a known, open library as Bouncy Castle or the direct integration into the operating system API would be beneficial for the spread of the technology. Since our library is not fully tested, we can recommend it only with reservations. For sensitive data we recommend to use a broadly based implementation as the C library MIRACL of Certivox or a dedicated application as offered by Voltage Security. With the help of two examples we have shown that a subsequent integration of an encryption is not impossible and with FPE it can be realized relatively easily.

9 List of illustrations

All illustrations without an indication of source are self-made.

Figure 1: Cover picture	1
<i>Background: http://3.bp.blogspot.com/.../s1600/matrix-binary.jpg</i>	
<i>Credit Card: http://www.silverxcard.com/wp-content/uploads/2012/09/visa-credit-card.jpg</i>	
<i>Lock: http://www.gtcomm.net/blog/wp-content/uploads/2014/01/securing-your-data.png</i>	
Figure 2: Planned Estimation of Work Volume	10
Figure 3: Communication between bank and ATM	16
Figure 4: Balanced Feistel	21
Figure 5: Feistel on Non-Binary Alphabets	22
Figure 6: Unbalanced Feistel	22
Figure 7: Alternating Feistel	22
Figure 8: Procedure of EME2 <i>http://eprint.iacr.org/2004/125.pdf, Page 6</i>	27
Figure 9: Encryption and decryption with cycle walking	28
Figure 10: FPE-Library Class Overview	31
Figure 11: FPE Cipher Classes	32
Figure 12: Message Space Classes and Key Class	33
Figure 13: Start page of the application	44
Figure 14: Overview of the client's assets	45
Figure 15: Transactions for all accounts	45
Figure 16: Package encUserTypes	49
Figure 17: Decrypted Data in Web Application	50
Figure 18: Web application successfully decides between encrypted and unencrypted data	51
Figure 19: OwnTracks Logo <i>http://owntracks.org/</i>	52
Figure 20: OwnTracks Locations	52
Figure 21: OwnTracks Map	52
Figure 22: MQTT Logo <i>http://mqtt.org/new/wp-content/uploads/2011/08/mqttorg-glow.png</i>	53
Figure 23: Communication between OwnTracks clients and MQTT-Broker	53
Figure 24: Content of the JSON in clear text, encrypted with AES or FPE	54
Figure 25: Original Security Menu compared to our additions	55
Figure 26: Changes in the logic of outgoing messages	55
Figure 27: Changes in the logic of incoming messages	56

Figure 28: The same location, this time encrypted, seen by all users not in possession of your password 59

Figure 29: Your location without encryption as seen by all users 59

10 Contents of the table

Table 1: Involved persons and their roles	6
Table 2: Resource requirements	6
Table 3: Goals for the Java FPE-Library	7
Table 4: Goals for the Tutorial	7
Table 5: Goals for the Android Demo App	8
Table 6: Milestones	8
Table 7: Timetable	9
Table 8: Meetings with the advisor	11
Table 9: Meetings with the expert	11
Table 10: Advantages and disadvantages of three tiny space FPE schemes	19
Table 11: Example of a Knuth shuffle	20
Table 12: Overview of the FFX parameters	24
Table 13: Number range message space	29
Table 14: String message space	29
Table 15: Enumeration message space	30
Table 16: Juridical aspects of the FPE-algorithms	40
Table 17: Devices used for the benchmark	41
Table 18: FPE-cipher benchmark	42
Table 19: Message space building benchmark	42
Table 20: Comparison P inside and outside round function	43
Table 21: Domains, message spaces and ciphers of the demo application	46
Table 22: Encrypted Data in Database	50
Table 23: Encrypted elements in our implementation	57
Table 24: Glossary	67
Table 25: Tiny-space FPE schemes composed by Philipp Rogaway in [1]	74
Table 26: Small-space FPE schemes composed by Philipp Rogaway in [1]	75
Table 27: Large-space FPE schemes composed by Philipp Rogaway in [1]	76
Table 28: Prefix cipher permutation table with seven example entries	77
Table 29: Representation of digits in the factorial number system	77
Table 30: Use permutation numbering to calculate the 315-th permutation	78

11 Glossary

Term	Definition
Advanced Encryption Standard (AES)	Today's most recommended and safest block cipher with a block length of 128 bits and key length of 128, 192 or 256 bits.
Automaton	Model to design sequential logic in computer science.
Block cipher	Encryption scheme that encrypts a plaintext block of a certain length to a ciphertext block of the same length.
Ciphertext	Output value of an encryption.
Cryptography	Science that secures information systems against unauthorized reading and altering.
Cycle Walking	FPE technique where the output of an encryption is encrypted over and over until it fits in a given value range.
Deterministic Encryption	With the same key, such an encryption returns always the same ciphertext to the same plaintext.
Deterministic Finite Automaton (DFA)	Particular kind of an automaton with a finite amount of states where every computation for an input string is unique.
EME2	FPE-scheme for input lengths greater than 128 bits.
Feistel cipher	Generic construction for designing block ciphers.
FFX	FPE-scheme based on a Feistel cipher for variable input lengths.
Format Preserving Encryption (FPE)	Encryption schemes where the ciphertext has the same format as the plaintext. Possible format can be number, string, etc.
Hibernate	Java framework for mapping a domain model to a relational database.
Initialization Vector / Tweak	Additional plaintext data in an encryption to prevent deterministic encryption.
Internet of Things (IoT)	Idea of connecting all possible things like household object together on the Internet.
JavaScript Object Notation (JSON)	Represents data object as attribute-value pair strings.
Knuth Shuffle	Algorithm for random permutation on a finite set.
Legacy Systems	Outdated system still in operation for certain reasons.
Maven	Software by Apache for automating the build process.
Message Space	Domain that contains all possible values from a given format.
MQTT	Publish-subscribe based "light weight" messaging protocol [38]
Plaintext	Input value of an encryption.

Pseudorandom Number Generator (PRNG)	Algorithm for generating random numbers by a small set of initial values which makes it not a true random.
Rank-Then-Encipher (RtE)	FPE-scheme where the distinct number (rank) of an element in a message is encrypted which gives another rank in the same message space.
Symmetric Encryption	Encryption scheme that uses the same key for the encryption as for the decryption.

Table 24: Glossary

12 Bibliography

The following section gives a compilation of interesting issues on the subject of FPE. We do not refer to all of this from this paper. There are also some references which provide interesting additional background knowledge.

The references are separated into the categories papers, websites/blogs, videos, implementations of FPE and applications which use FPE.

The content of the Internet sources are verified by us until the 11.06.2015. We cannot guarantee that the content remains the same after this date.

12.1 Papers

[1] A Synopsis of Format-Preserving Encryption

*Rogaway, Phillip, March 27, 2010,
Simple introduction from a leading researcher in the field of FPE.*
<http://web.cs.ucdavis.edu/~rogaway/papers/synopsis.pdf>

[2] Format-Preserving Encryption

*Bellare, Mihir, Ristenpart, Thomas, Rogaway, Phillip, Stegers, Till, November 3, 2009
Formal introduction to FPE from the co-founders including security proofs.*
<https://eprint.iacr.org/2009/251.pdf>

[3] The FFX Mode of Operation for Format-Preserving Encryption

*Bellare, Mihir, Rogaway, Phillip, Spies, Terence, February 20, 2010,
Definition of the FFX FPE scheme.*
<http://csrc.nist.gov/groups/ST/toolkit/BCM/documents/proposedmodes/ffx/ffx-spec.pdf>

[4] Addendum to "The FFX Mode of Operation for Format-Preserving Encryption"

*Bellare, Thomas, Rogaway, Phillip, Spies, Terence, September 3, 2010,
Second version of the FFX FPE scheme with some improvements.*
<http://csrc.nist.gov/groups/ST/toolkit/BCM/documents/proposedmodes/ffx/ffx-spec2.pdf>

[5] Using datatype-preserving encryption to enhance data warehouse security

*Brightwell, Michael, Smith, Harry, 1997,
Older FPE approach and predecessor of FFX.*
csrc.nist.gov/nissc/1997/proceedings/141.pdf

[6] Guidelines for Implementing and Using the NBS Data Encryption Standard

*National Bureau of Standards, April 1, 1981,
First, DES-based, FPE approach.*
<https://www.thc.org/root/docs/cryptography/fips74.html>

[7] FPE White Paper by Voltage Security

*Spies, Terence,
Paper from co-founder of FPE in which three FPE methods are described.*
<https://www.voltage.com/pdf/Voltage-Security-WhitePaper-Format-Preserving-Encryption.pdf>

[8] A Note on the Implementation of Format Preserving Encryption Mode

*Scott, Michael,
Comparison of three FPE schemes.*
http://cdn2.hubspot.net/hub/230906/file-20129878-pdf/downloads/certivox_labs_fpe.pdf

[9] Data-Centric Security vs. Database-Level Security

*Voltage Security, July 7, 2014,
Voltage shows the benefit of their SecureData solution - which encrypts or tokenizes the data -
over encryption at database level.*
www.voltage.com/wp-content/uploads/Voltage_TB_Transparent-Data-Encryption_FINAL.pdf

[10] VAES3 scheme for FFX

*Vance, Joachim, May 20, 2011,
An addendum to [3]. A parameter collection for encipher strings of arbitrary radix with sub key
operation to lengthen life of the enciphering key.*
<http://csrc.nist.gov/groups/ST/toolkit/BCM/documents/proposedmodes/ffx/ffx-ad-VAES3.pdf>

[11] BPS: a Format Preserving Encryption Proposal

*Brier, Eric, Peyrin, Thomas, Stern, Jacques, April, 2010,
Alternative to the FFX FPE scheme.*
<http://csrc.nist.gov/groups/ST/toolkit/BCM/documents/proposedmodes/bps/bps-spec.pdf>

[12] Follow-up on NIST's Consideration of XTS-AES as standardized by IEEE Std 1619-2007

Matthew V. Ball, Sun Microsystems, 2009
http://csrc.nist.gov/groups/ST/toolkit/BCM/documents/comments/XTS/follow-up_XTS_comments-Ball.pdf

[13] PRP Mode Comparison IEEE P1619.2

David McGrew, Cisco
Comparison of all the modes proposed for IEEE 16.19.2
<http://grouper.ieee.org/groups/1619/email/pdf00047.pdf>

[14] EME*: extending EME to handle arbitrary-length messages with associated data

Shai Halevi, Philip Rogaway, May 27, 2004
<http://eprint.iacr.org/2004/125.pdf>

[15] The Extended Codebook (XCB) Mode of Operation

David A. McGrew, Scott Fluhrer, October 25, 2004
<http://eprint.iacr.org/2004/278.pdf>

[16] On Some Weaknesses in the Disk Encryption Schemes EME and EM2

Cuauhtemoc Mancillas-Lopez, Debrup Chakraborty and Francisco, Rodriguez-Henriquez
<http://delta.cs.cinvestav.mx/~debrup/iciss09.pdf>

[17] Another Look at XCB

Debrup Chakraborty, Vicente Hernandez-Jimenez, Palash Sarkar
Paper about the vulnerabilities and possible attack on XCB mode.
<https://eprint.iacr.org/2013/823.pdf>

[18] On Generalized Feistel Networks

Hoang, Viet Tung, Rogaway, Phillip, February, 2010
Security proof of well-known generalized Feistel networks.
<https://eprint.iacr.org/2010/301.pdf>

12.2 Websites / Blogs

[19] Wikipedia article on Cryptography

<http://en.wikibooks.org/wiki/Cryptography/Introduction>

[20] Wikipedia article on symmetric encryption

http://en.wikipedia.org/wiki/Symmetric-key_algorithm

[21] Wikipedia article on block cipher

http://en.wikipedia.org/wiki/Block_cipher

[22] Wikipedia article on block cipher mode of operation

http://en.wikipedia.org/wiki/Block_cipher_mode_of_operation

[23] Wikipedia article on deterministic encryption

http://en.wikipedia.org/wiki/Deterministic_encryption

[24] Wikipedia article on AES

http://en.wikipedia.org/wiki/Advanced_Encryption_Standard

[25] Wikipedia article on FPE

http://en.wikipedia.org/wiki/Format-preserving_encryption

[26] Wikipedia article on disk encryption theory

http://en.wikipedia.org/wiki/Disk_encryption_theory

[27] Wikipedia article on IEEE P1619

http://en.wikipedia.org/wiki/IEEE_P1619

[28] Voltage Security

<https://www.voltage.com/pdf/Voltage-Security-WhitePaper-Format-Preserving-Encryption.pdf>

[29] Encrypting credit card numbers using FFSEM

Explains FFSEM (predecessor of FFX) by means of encrypting a credit card number number.
<http://www.di-mgt.com.au/crypto-ffsem.html>

[30] Knuth shuffle / Fisher-Yates shuffle

The Knuth shuffle (or Fisher-Yates shuffle) can be used to create a tiny space FPE scheme.
http://en.wikipedia.org/wiki/Fisher-Yates_shuffle#The_modern_algorithm

[31] You Don't Want XTS

A blog about disk encryption and critics of the AES-XTS mode
<http://sockpuppet.org/blog/2014/04/30/you-dont-want-xts/>

[32] IEEE 1619 Security in Storage Working Group (SISWG)

The IEEE Security in Storage work group (SISWG) is working on standards related to encrypted storage media, including both encryption and key management.
<http://siswg.net/>

[33] FTE in the Tor project

FTE is used by the Tor Project to circumvent deep packet inspection.
<https://www.torproject.org/docs/pluggable-transport.html.en>

[34] FTE in the fteproxy

Proxy based on FTE to provide transport-layer protection.
<https://fteproxy.org/about>

[35] HP Voltage Security Patents

<https://www.voltage.com/technology/standards/patents-and-research/>

[36] Mailing List about EME2 Patents

<http://www.saout.de/pipermail/dm-crypt/2013-February/003165.html>

[37] Letter of Knuth about Patents

http://www.pluto.it/files/meeting1999/atti/no-patents/brevetti/docs/knuth_letter_en.html

[38] MQTT

<http://mqtt.org/>

[39] Android SharedPreferences Security

<http://stackoverflow.com/questions/9244318/android-sharedpreference-security>

[40] OwnTracks Starter Guide

<https://github.com/owntracks/owntracks/wiki/Getting-started>

[41] Encryption in MySQL

Supported AES modes.
<http://mysqlserverteam.com/understand-and-satisfy-your-aes-encryption-needs-with-5-6-17>

[42] Wikipedia Article on Transparent Data Encryption

http://en.wikipedia.org/wiki/Transparent_Data_Encryption

[43] Encryption Hierarchy in Microsoft SQL-Server

<https://msdn.microsoft.com/en-us/ms189586>

[44] Security of the Microsoft Data Protection API

Vulnerability of DPAPI data protection in Windows Server 2000 - 2012.
<http://www.passcape.com/index.php?section=blog&cmd=details&id=34>

[45] Spring MVC 4 Quickstart Maven Archetype

Maven archetype we used as basis for our tutorial.
<https://github.com/kolorobot/spring-mvc-quickstart-archetype>

[46] JPMorgan Demo Web Application

Demo web application to demonstrate the integration of FPE.
<https://github.com/Rizja/JPMorganDemo/>

[47] Regex for a timestamp

We use the Regex of Arthur Lorent for our timestamp message space.
<http://stackoverflow.com/questions/1057716/regular-expression-to-validate-a-timestamp>

[48] Wikipedia Article on International Bank Account Number (IBAN)

http://en.wikipedia.org/wiki/International_Bank_Account_Number

[49] JP Morgan sees 76 million customer accounts hacked

<http://www.bbc.com/news/business-29470381>

[50] DFA library dk.brics.automaton

<http://dk.brics.automaton>

12.3 Videos

[51] Online Cryptography Course- Introduction to FPE

Session on FPE from an online cryptography course by Dan Boneh (Stanford University).
<https://class.coursera.org/crypto-preview/lecture/45> (12 min)

[52] Winter School on Cryptography Symmetric Encryption

Session on FPE recorded at the Bar-Ilan University in Israel held by Thomas Ristenpart (University of Wisconsin - Madison).
<https://www.youtube.com/watch?v=MT5ketZ-jLw> (1h 14min)

12.4 Implementations of FPE

[53] MIRACL from v7.0.0

MIRACL (Multiprecision Integer and Rational Arithmetic Cryptographic Library) is a crypto library. It contains an implementation of BPS proposed by Brier, Peyrin and Stern. Language: C, Licence: commercial or AGPL
<https://www.certivox.com/miracl>

[54] Botan Library from v1.9.17

Botan is a crypto library. It contains an implementation of an integer FPE using the scheme FE1 from the paper "Format-Preserving Encryption" (<http://eprint.iacr.org/2009/251>) by Mihir Bellare, Thomas Ristenpart, Phillip Rogaway und Till Stegers.

Language: C++, Licence: BSD2

<http://botan.randombit.net/manual/fpe.html>

[55] DotFPE v0.8.1.1

Port of the FPE part from the Botan Library from C++ to C#.

Language: C#, Licence: BSD, Release: October 2012, Last Update: October 2012

<http://dotfpe.codeplex.com>

[56] libffx

Libffx is a Python implementation of the FFX mode of operation for FPE.

Language: Python, Licence: GPL, Release: January 2014, Last Update: May 2014

<http://github.com/kpdyer/libffx>

[57] IEEE 1619 Reference Implementations

Reference implementation of EME2

Language: C, Licence: GPL, MIT License Release: October 2007, Last Update: April 2013

<http://sourceforge.net/projects/crypto1619>

12.5 Applications which use FPE

[58] Voltage Security

Voltage Security has several products for securing data like emails, files, payment transactions, ecommerce transactions and other data in databases, applications, data warehouses and cloud environments.

<http://www.voltage.com>

[59] Protegrity

Provides like Voltage comprehensive solutions for protecting databases, files, applications and Hadoop solutions.

<http://www.protegrity.com/>

[60] StratoKey

Acts as gateway between the user and several cloud products like Google Docs, Dropbox, Sales Force, Confluence and others and protects the data with FPE.

<https://www.stratokey.com/>

[61] XYPRO

Provides a solution based on Voltages SecureData extended to provide protection for the HP NonStop server and peripheral systems.

<https://www.xypro.com>

[62] Perspecsys

AppProtex provides Encryption of data between user and cloud. The gateway can be deployed as hosted or managed solution. For some providers an additional adapter interfaces is available to preserve functionalities, such as the ability to search or sort data.

<http://www.perspecsys.com>

[63] BestCrypt

BestCrypt is a commercial disk encryption program for Windows, Linux and Mac OS X, developed by Jetico. It is a software suite that provides container encryption and partition encryption as well some additional encryption methods and complimentary data erasure utility BCWipe. (Wikipedia)
<http://www.jetico.com/products/personal-privacy/bestcrypt-container-encryption>

[64] TrueCrypt

TrueCrypt is a discontinued source-available freeware utility used for on-the-fly encryption (OTFE). It can create a virtual encrypted disk within a file or encrypt a partition or (under Microsoft Windows except Windows 8 with GPT) the entire storage device (pre-boot authentication). On 28 May 2014, the TrueCrypt website announced that the project was no longer maintained and recommended users to find alternative solutions. (Wikipedia)
<http://truecrypt.sourceforge.net/>

13 Appendix

13.1 Definition of Bachelor Thesis Project

The image shows a document with the following content:

Definition of Bachelor Thesis Project

for	Matthias Liechti Rizja Schmid
Division	Computer Science
Advisor	Dr. Reto Koenig

Format Preserving Encryption

There is too much private data floating around in the cloud databases. Using format preserving encryption (fpe), this data can be encrypted without violating the input validation or the record structure of a database.

Within this bachelor thesis, a typical service from a big provider shall serve as an example in order to demonstrate how to store private data without losing privacy.

The following points have to be achieved:

- Providing a library which easily allows to apply fpe in an existing context
- Providing a detailed developers tutorial on how to apply fpe in an existing context
- Providing one implementation of an application that has been enriched by fpe

Start of Project	16. February 2015
End of Project	12. June 2015

13.2 Tiny-space FPE schemes

method	can encrypt on	description	security
Knuth shuffle [15, 18, 29, 40, 59]	$\mathcal{X} = [N]$ where N is small	Shuffle $[N]$ by repeatedly choosing an element from a decreasing prefix and moving it to the end. Slow setup then fast, table-driven encryption. Map X to its position after the shuffle.	Provably secure with ideal bounds.
Permutation numbering	$\mathcal{X} = [N]$ where N is very small	Map key K to a number $k \in [N!]$ and encrypt with the k 'th permutation π_k on $[N]$. One AES call enough to usually determine permutation on $N \leq 34$ points (32-bit arithmetic enough for $N \leq 12$, 64-bit arithmetic enough for $N \leq 20$).	Provably secure with ideal bounds.
Prefix cipher [8]	$\mathcal{X} = [N]$ where N is small	Use ordering of $E_K(0), \dots, E_K(N-1)$ to determine permutation. Slow setup then fast, table-driven encryption. Simpler but uses more coins than Knuth shuffle.	Provably secure with ideal bounds.

Table 25: Tiny-space FPE schemes composed by Philipp Rogaway in [1]

13.3 Small-space FPE schemes

method	can encrypt on	description	security
Alternating Feistel [1, 32]	$\mathcal{X} = \{0, 1\}^n$ for any desired n	Like unbalanced Feistel but based on ladder drawing of Feistel. Round functions' domain and range alternate between contracting and expanding.	Provably secure to nearly the size of the domain of the round function with larger domain [8, 26, 32].
Brightwell-Smith scrambling [9]	$\mathcal{X} = \Sigma^n$ for any desired Σ and n	Characters successively modified by a multi-step process involving modular additions, rippling, and the use of DES in OFB mode. Later elaborated and named DTP [34].	Cryptographically unsophisticated scheme that seems unlikely to be secure. No provable security results are claimed.
Classical Feistel [16]	$\mathcal{X} = \{0, 1\}^{2n}$ for any desired n	Classical (~1971), extensively studied approach. Variable number of rounds (eg, 3–32). Can use a high-assurance PRF as the round function.	With enough rounds, provably secure to about 2^n queries [36, 49]. With enough rounds, known attacks are completely ineffective [50]. Also true of all Feistel variants described here.
FIPS 74 scrambling [45]	$\mathcal{X} = \Sigma^n$ for any desired n	Characterwise-modular addition of plaintext and a key-dependent, DES-based pad.	Trivially attackable.
FFSEM [68]	$\mathcal{X} = \{0, 1\}^{2n}$ for any desired n ; or $\mathcal{X} = [N]$ for any desired N	Former NIST submission from Voltage Security that combines classical Feistel and cycle walking. Has been replaced by FFX.	Inherits classical Feistel results.
FFX [7]	$\mathcal{X} = \Sigma^n$ for any desired n	NIST submission uses alternating or unbalanced Feistel. Highly parametrized. Parameter sets A2 and A10 concretize enciphering binary and decimal strings.	Provably secure with bounds associated to alternating or unbalanced Feistel [26, 39, 42].
Numeric Feistel [5, 8]	$\mathcal{X} = [N]$ where $N = a b$ for $a, b \geq 2$	Like unbalanced or alternating Feistel but everything is done mod a or mod b instead of using binary strings. Intended to make cycle walking rare.	Provably secure with about the same bounds as the analogous string-based methods [5, 8, 26].
Recursive-merge shuffle [21]	$\mathcal{X} = [N]$ where N is arbitrary	Trace the trajectory of (just) card x in a shuffle suggested by [14, 42]. Samples from hypergeometric distribution to do so. Impractically slow.	Provably secure with ideal bounds.
Thorp shuffle [71]	$\mathcal{X} = [N]$ for even N	Split deck in half and let cards fall left-right or right-left depending on a coin. Coincides with maximally unbalanced Feistel if $N = 2^n$.	Provably secure to $N^{1-\epsilon}$ queries (with $O(1/\epsilon)$ passes) [26, 39]. Weak security notions already achieved with two passes [39].
Unbalanced Feistel [63]	$\mathcal{X} = \{0, 1\}^n$ for any n	Generalized Feistel scheme where split is not into even halves. Source-heavy and target-heavy subtypes.	Provably secure to nearly the size of the domain of the round function [26, 42]. Weak attacks [51, 52]. Source-heavy preferred.

Table 26: Small-space FPE schemes composed by Philipp Rogaway in [1]

13.4 Large-space FPE schemes

method	can encrypt on	description	security
ABL4 [38]	$\mathcal{X} = \{0, 1\}^*$	Unbalanced Feistel with a layer of ECB. Slow, eclipsed by later work.	Provable security claimed, but no known writeup.
CMC [25]	$\mathcal{X} = \{0, 1\}^{mw}$ where $m \geq 2$	An encrypt-mix-encrypt scheme where encryption is CBC and the mix is very lightweight. Not parallelizable.	Provably secure with good bounds.
EME [24]	$\mathcal{X} = \{0, 1\}^{mw}$ where $m \geq 1$	An encrypt-mix-encrypt scheme where encryption is ECB and the masking is lightweight.	Provably secure with good bounds.
EME2 [22, 27]	$\mathcal{X} = \{0, 1\}^{\geq w}$	Refinement of EME to extend the domain. Formerly named EME*. In draft standard IEEE P1619.2.	Provably secure with good bounds.
HCH [11]	$\mathcal{X} = \{0, 1\}^{>w}$	Follow-on to HCTR, an efficiency and security improvement to it.	Provably secure with good bounds.
HCTR [10, 72]	$\mathcal{X} = \{0, 1\}^{\geq w}$	A hash-counter-hash scheme where the hash is relatively heavy.	Provably secure with good bounds.
HEH [61] HEH* [60]	$\mathcal{X} = (\{0, 1\}^w)^+$ $\mathcal{X} = \{0, 1\}^{\geq w}$	A hash-encrypt-hash follow-on to TET; improves efficiency, makes VIL.	Provably secure with good bounds.
HMC [41]	$\mathcal{X} = \{0, 1\}^{>w}$	Follow-on work to HCTR improves bounds and efficiency.	Provably secure with good bounds.
NR [42, 43]	$\mathcal{X} = \{0, 1\}^{mw}$ where $m \geq 1$	A hash-encrypt-hash scheme with a relatively heavy hash and an encrypt of ECB mode, say. Not fully specified.	Provably secure with good bounds.
PEP [12]	$\mathcal{X} = \{0, 1\}^{mw}$ where $m \geq 1$	Like TET, follow-on to NR. Poor speed, eclipsed by other modes.	Provably secure with good bounds.
TES [62]	$\mathcal{X} = \{0, 1\}^{\geq 2w}$ where $m \geq 1$	A hash-encrypt-hash scheme using only the forward direction of the blockcipher. Tweakable, VIL.	Provably secure with good bounds.
TET [23]	$\mathcal{X} = \{0, 1\}^{mw}$ where $m \geq 1$	A hash-encrypt-hash scheme that concretizes NR. Tweakable, parallelizable, FIL.	Provably secure with good bounds.
Type-1 Feistel [73] Type-2 Feistel [73] Type-3 Feistel [73]	$\mathcal{X} = \{0, 1\}^{mw}$ for desired m	Generalized Feistel schemes used in blockciphers like CAST-256, RC6, and MARS.	Provably secure with good bounds [26, 73].
VIL [6]	$\mathcal{X} = \{0, 1\}^{\geq w}$	Early scheme that helped introduced and formalize the problem.	Provably secure for CPA security, but does not achieve CCA security.
XCB [37]	$\mathcal{X} = \{0, 1\}^{\geq 2w}$ or $\mathcal{X} = \{0, 1\}^{\geq w}$ with nonce AD	A hash-counter-hash scheme with a relatively heavy hash. In draft standard IEEE P1619.2.	Provably secure with good bounds, although the writeup leaves some questions about the precise claims.

Table 27: Large-space FPE schemes composed by Philipp Rogaway in [1]

13.5 Prefix cipher

The prefix cipher uses the ordering of the encrypted elements to do the permutation. The initialization of the permutation-table is slow. Afterwards the encryption and decryption can be done with a fast lookup in the table.

To explain the scheme, we consider as an example the encryption of a weekday. We now have to decide which day decrypts to which other day with the use of a symmetric key. We do that by encrypting every element $x \in X$ with a symmetric cipher E . We then sort by the encrypted value $E(x)$ and use the position y in the new order for the permutation.

In the following example the encryption is made with AES and the key $k = \{0\}^{128}$. After the encryption of every element and the sorting, the plaintext “4” maps to the ciphertext “2” and vice versa.

y	x	E(x) ▼
0	2	42C07771A57CB56665CB3B796B189A70
1	1	6331C0B8DCF72CEEF41D09F461399140
2	4	6862D130AB3DACEB330037A767450332
3	0	9B9AF18901273E33B99C3338DE77B0A6
4	3	A5829BF144C026105712CEDBCAFA7A04
5	6	AFCFEE5489EE27B1A64BCDDDBA6D93D8
6	5	CF53A94AAC82691DED8DFC742E861509

Table 28: Prefix cipher permutation table with seven example entries

13.6 Permutation numbering

The permutation numbering is based on the factorial number system (also called factoradic). The factorial number system is a mixed radix numeral system – the numerical base varies within an integer from position to position. The base of the i -th digit from right is given by i .

i-th digit from right	...	7	6	5	4	3	2	1
radix		7	6	5	4	3	2	1
digits allowed		0-6	0-5	0-4	0-3	0-2	0-1	0
place value		6!	5!	4!	3!	2!	1!	0!
place value decimal		5040	120	24	6	2	1	1

Table 29: Representation of digits in the factorial number system

For example the decimal 17 is represented as $17 = 2 * 3! + 2 * 2! + 1 * 1! + 0 * 0! = 2_3 2_2 1_1 0_0$.

The permutation numbering is used to do a uniquely defined mapping between the integers $0, \dots, n! - 1$ and permutations of n elements in lexicographical order. This mapping is called Lehmer code or inversion table.

In the permutation numbering FPE scheme we have n elements and therefore $n!$ possible permutations. The symmetric key k is mapped to a permutation and thus has to be between $0 \leq k \leq n! - 1$. The resulting permutation we then use to do the mapping in the encryption/decryption.

For the explanation we use again the example of the weekdays. We have 7 elements and thus 5040 possible permutations. Therefore the key k has to be between $0 \leq k \leq 5039$. To get the permutation for the weekdays with the key $k_{315} = 315 = 0 * 6! + 2 * 5! + 3 * 4! + 0 * 3! + 1 * 2! + 1 * 1! + 0 * 0! = 0_6 2_5 3_4 0_3 1_2 1_1 0_0$ we proceed as follows:

- Starting from left, take the i -th digit of the key in the Lehmer code representation. For each digit do:
1. In the original sequence take the number at this position.
 2. Remove it and add it at the end of the permutation.

Index i	Lehmer Code k_{315}	Original	Permutation k_{315}
0	0 ₆ 2 ₅ 3 ₄ 0 ₃ 1 ₂ 1 ₁ 0 ₀	0 1 2 3 4 5 6	0
1	0 ₆ 2 ₅ 3 ₄ 0 ₃ 1 ₂ 1 ₁ 0 ₀	1 2 3 4 5 6	0 3
2	0 ₆ 2 ₅ 3 ₄ 0 ₃ 1 ₂ 1 ₁ 0 ₀	1 2 4 5 6	0 3 5
3	0 ₆ 2 ₅ 3 ₄ 0 ₃ 1 ₂ 1 ₁ 0 ₀	1 2 4 6	0 3 5 1
4	0 ₆ 2 ₅ 3 ₄ 0 ₃ 1 ₂ 1 ₁ 0 ₀	2 4 6	0 3 5 1 4
5	0 ₆ 2 ₅ 3 ₄ 0 ₃ 1 ₂ 1 ₁ 0 ₀	2 6	0 3 5 1 4 6
6	0 ₆ 2 ₅ 3 ₄ 0 ₃ 1 ₂ 1 ₁ 0 ₀	2	0 3 5 1 4 6 2

Table 30: Use permutation numbering to calculate the 315-th permutation

13.7 OwnTracks Helper Class FormatPreservingEncryption

```

package org.owntracks.android.support;

import android.util.Base64;

import ch.bfh.fpelib.Key;
import ch.bfh.fpelib.RankThenEncipher;
import ch.bfh.fpelib.intEnc.FFXIntegerCipher;
import ch.bfh.fpelib.intEnc.KnuthShuffleCipher;
import ch.bfh.fpelib.messageSpace.StringMessageSpace;
import org.json.JSONException;
import org.json.JSONObject;

import java.math.BigInteger;
import java.nio.charset.Charset;
import java.security.GeneralSecurityException;
import java.util.Arrays;
import java.util.HashSet;
import java.util.Iterator;

import javax.crypto.Cipher;
import javax.crypto.spec.IvParameterSpec;
import javax.crypto.spec.SecretKeySpec;

public class FormatPreservingEncryption {

    //Format (regex or max value) definitions for JSON elements to be encrypted
    private static final String TYPES_IMPLEMENTED = "(location|waypoint|transition)";
    private static final BigInteger ALT_MAX = BigInteger.valueOf(10000);
    private static final BigInteger BATT_MAX = BigInteger.valueOf(100);
    private static final BigInteger COG_MAX = BigInteger.valueOf(359);
    private static final String LAT_REGEX = "-?(90|([0-9]|[1-8][0-9])(\\.([0-9]{1,8})?))";
    private static final String LON_REGEX = "-?(180|([0-9]|[0-9][0-9]|1[0-7][0-9])(\\.([0-9]{1,8})?))";
    private static final BigInteger TST_MAX = BigInteger.valueOf(4131648000L);

    //Initialization of various FPE-ciphers for each JSON element depending on the format
    private static final FFXIntegerCipher ALT = new FFXIntegerCipher(ALT_MAX);
    private static final KnuthShuffleCipher BATT = new KnuthShuffleCipher(BATT_MAX);
    private static final KnuthShuffleCipher COG = new KnuthShuffleCipher(COG_MAX);
    private static final StringMessageSpace LAT_MS = new StringMessageSpace(LAT_REGEX);
    private static final RankThenEncipher<String> LAT = new RankThenEncipher<>(LAT_MS);
    private static final StringMessageSpace LON_MS = new StringMessageSpace(LON_REGEX);
    private static final RankThenEncipher<String> LON = new RankThenEncipher<>(LON_MS);
    private static final FFXIntegerCipher TST = new FFXIntegerCipher(TST_MAX);

    /**
     * Empty private constructor to avoid instantiation of this helper class
     */
    private FormatPreservingEncryption() {}

```

```

/**
 * Creates a new JSONObject from a JSON string
 * @param jsonMessage JSON message as string
 * @return JSON object
 */
public static JSONObject getJsonMessage(String jsonMessage) {
    try {
        return new JSONObject(jsonMessage);
    } catch (JSONException e) {
        e.printStackTrace();
        return null;
    }
}

/**
 * Encrypt predefined elements in the JSON object with FPE techniques
 * @param json JSON Plaintext
 * @param key Randomly computed key
 * @param tweak Arbitrary bytes to prevent deterministic encryption
 * @return JSON object with encrypted elements
 */
public static JSONObject encrypt(JSONObject json, Key key, byte[] tweak) {
    if (!json.optString("_type").matches(TYPES_IMPLEMENTED)) {
        //if message type not supported return original
        return json;
    }
    JSONObject encryptedJson = new JSONObject();
    Iterator<String> entries = json.keys();
    while (entries.hasNext()) {
        String entry = entries.next();
        Object value;
        try {
            switch (entry) {
                case "alt":
                    value = ALT.encrypt(BigInteger.valueOf(json.getLong(entry)), key, tweak).intValue();
                    break;
                case "batt":
                    value = BATT.encrypt(BigInteger.valueOf(json.getLong(entry)), key, tweak).intValue();
                    break;
                case "cog":
                    value = COG.encrypt(BigInteger.valueOf(json.getLong(entry)), key, tweak).intValue();
                    break;
                case "desc":
                    try {
                        Cipher aes = Cipher.getInstance("AES/CBC/PKCS5Padding");
                        Key aesTweak = new Key(tweak);
                        aes.init(Cipher.ENCRYPT_MODE, new SecretKeySpec(key.getKey(16), "AES"),
                                new IvParameterSpec(aesTweak.getKey(16)));
                        byte[] encrypted = aes.doFinal(json.getString(entry).getBytes(
                                Charset.forName("UTF-8")));
                        value = Base64.encode(encrypted, Base64.DEFAULT);
                    } catch (GeneralSecurityException e) {
                        throw new RuntimeException("Unexpected exception. " + e.getMessage());
                    }
                    break;
                case "lat":
                    value = LAT.encrypt(json.getString(entry), key, tweak);
                    break;
                case "lon":
                    value = LON.encrypt(json.getString(entry), key, tweak);
                    break;
                case "wtst":
                    value = TST.encrypt(BigInteger.valueOf(json.getLong(entry)), key, tweak).longValue();
                    break;
                default: //all other entries are not encrypted
                    value = json.get(entry);
                    break;
            }
            encryptedJson.put(entry, value);
        } catch (JSONException e) {
            throw new RuntimeException("Unexpected exception. " + e.getMessage());
        }
    }
    return encryptedJson;
}

```

```

/**
 * Decrypt predefined elements in the JSON object with FFE techniques
 * @param json JSON Ciphertext
 * @param key Randomly computed key
 * @param tweak Arbitrary bytes to prevent deterministic encryption
 * @return Decrypted JSON object
 */
public static JSONObject decrypt(JSONObject json, Key key, byte[] tweak) {
    if (!json.optString("_type").matches(TYPES_IMPLEMENTED)) {
        //if message type not supported return original
        return json;
    }
    JSONObject decryptedJson = new JSONObject();
    Iterator<String> entries = json.keys();
    while (entries.hasNext()) {
        String entry = entries.next();
        Object value;
        try {
            switch (entry) {
                case "alt":
                    value = ALT.decrypt(BigInteger.valueOf(json.getLong(entry)), key, tweak).intValue();
                    break;
                case "batt":
                    value = BATT.decrypt(BigInteger.valueOf(json.getLong(entry)), key, tweak).intValue();
                    break;
                case "cog":
                    value = COG.decrypt(BigInteger.valueOf(json.getLong(entry)), key, tweak).intValue();
                    break;
                case "desc":
                    try {
                        Cipher aes = Cipher.getInstance("AES/CBC/PKCS5Padding");
                        Key aesTweak = new Key(tweak);
                        aes.init(Cipher.DECRYPT_MODE, new SecretKeySpec(key.getKey(16), "AES"),
                                new IvParameterSpec(aesTweak.getKey(16)));
                        byte[] decoded = Base64.decode(json.getString(entry).getBytes(
                                Charset.forName("UTF-8")), Base64.DEFAULT);
                        value = new String(aes.doFinal(decoded), Charset.forName("UTF-8"));
                    } catch (GeneralSecurityException e) {
                        throw new RuntimeException("Unexpected exception. " + e.getMessage());
                    }
                    break;
                case "lat":
                    value = LAT.decrypt(json.getString(entry), key, tweak);
                    break;
                case "lon":
                    value = LON.decrypt(json.getString(entry), key, tweak);
                    break;
                case "wtst":
                    value = TST.decrypt(BigInteger.valueOf(json.getLong(entry)), key, tweak).longValue();
                    break;
                default: //all other entries are not decrypted
                    value = json.get(entry);
                    break;
            }
            decryptedJson.put(entry, value);
        } catch (JSONException e) {
            throw new RuntimeException("Unexpected exception. " + e.getMessage());
        }
    }
    return decryptedJson;
}

/**
 * Stores username and password in the SharedPreferences
 * @param username Username to store
 * @param password Password according to username
 */
public static void setPassword(String username, String password) {
    deletePassword(username); //First remove old password of user if exists, before storing the new one
    HashSet<String> userPWs = new HashSet<String>(Preferences.getSharedPreferences().getStringSet(
        "userPasswords", new HashSet<String>()));
    userPWs.add(username + "$" + password);
    Preferences.getSharedPreferences().edit().putStringSet("userPasswords", userPWs).commit();
}

```

```

/**
 * Deletes username and password of a given user from the SharedPreferences
 * @param username User to delete
 */
public static void deletePassword(String username) {
    HashSet<String> userPWs = new HashSet<String>(Preferences.getSharedPreferences().getStringSet(
        "userPasswords", new HashSet<String>()));
    for (String userPW : userPWs) {
        if (userPW.contains(username)) {
            userPWs.remove(userPW);
            break;
        }
    }
    Preferences.getSharedPreferences().edit().putStringSet("userPasswords", userPWs).commit();
}

/**
 * Returns the password of a given user
 * @param username User of the password
 * @return Password of the user
 */
public static String getPassword(String username) {
    HashSet<String> userPWs = new HashSet<String>(Preferences.getSharedPreferences().getStringSet(
        "userPasswords", new HashSet<String>()));
    String password = "";

    for (String userPW : userPWs) {
        String userPWLowerCase = userPW.toLowerCase();
        if (userPWLowerCase.contains(username.toLowerCase())) {
            password = userPW.substring(userPW.indexOf("$") + 1);
            return password;
        }
    }
    return null;
}

/**
 * Returns all users as string array
 * @return String array with all usernames
 */
public static String[] loadUsers() {
    HashSet<String> userSet = new HashSet<String>(Preferences.getSharedPreferences().getStringSet(
        "userPasswords", new HashSet<String>()));
    String[] users = userSet.toArray(new String[userSet.size()]);

    for (int i = 0; i < users.length; i++) {
        users[i] = users[i].substring(0, users[i].indexOf("$"));
    }
    return users;
}
}

```