B.Eng. Thesis
Bachelor of Engineering

DTU Compute
Department of Applied Mathematics and Computer Science

Decryption Key Management System for Ulfberht

Christian Feilberg Hansen

Herlev and Kongens Lyngby
2014 and 2015

Brunata A/S
Group R&D

Vesterlundvej 14
2730 Herlev, Denmark
Phone +45 7777 7010
brunata@brunata.dk
www.brunata.dk

DTU Compute
Department of Applied Mathematics and Computer Science
Technical University of Denmark

Matematiktorvet
Building 303B
2800 Kongens Lyngby, Denmark
Phone +45 4525 3031
compute@compute.dtu.dk
www.compute.dtu.dk

Summary(English)

Brunata needs to use meters that encrypt their data. These meters each encrypt their data with their own unique encryption key. The number of meters needed, potentially number in the hundreds of thousands. This will also mean that thousands of encryption keys for these meters will have to be stored and managed.

Ulfberht - RIS is Brunata's system for receiving telegrams from a number of different meters, the system validates and interprets each of these telegrams. In order for RIS to validate and intepret the encrypted data that these meters send, the data will have to be decrypted.

The solution to this has to be implemented into Ulfberht - RIS. The solution consists of three parts

- Storage and management for the encryption keys.
- Decryption of Data.
- Integration into Ulfberht - RIS.

Part 1 of the solution consists of a place store the encryption keys and a interface for managing the keys. Part 2 of the solution handles the decryption of the data and uses part 1 to retrieve the keys needed for decryption. Part 3 of the solution is responsible for sending the encrypted data to part 2.

Summary(Dansk)

Brunata har brug for at bruge målere der krypterer deres data. Hver af disse målere krypterer deres data med deres egen unikke krypterings nøgle. Der er potentielt brug for flere hundrede tusinder af målere. Det betyder samtidig at der også er brug for at validere og fortolke flere hundrede tusinder af krypteringsnøgler.

Ulfberht - RIS er Brunata's system til at modtage telegrammer fra mange forskellige målere, systemet validerer og fortolker hvert af disse telegrammer. For at RIS kan validere og fortolke disse nye målere, der krypterer deres data, så skal data'en dekrypteres.

Løsningen til dette skal implementeres ind i Ulfberht - RIS. Løsningen består af 3 dele:

- Opbevaring og forvaltning af krypterings nøglerne.
- Dekryptering af data'en.
- Integrationen ind i Ulfberht - RIS.

Del 1 af løsningen består af et sted at opbevare krypterings nøglerne og et interface til at forvalte nøglerne. Del 2 af løsningen består i at dekryptere data'en og bruge del 1 til at hente de krypterings nøgler der er brug for for at dekryptere data'en. Del 3 af løsningen er ansvarlig for at sende krypteret data til del 2.

Acknowledgements

- Anne Elisabeth Haxthausen, DTU Supervisor, Associate Professor in the Software Engineering section, DTU Compute, Technical University of Denmark.
- Morten Heebøll, Embedded Systems Engineer, Brunata Supervisor, Brunata A/S
- Bo Visfeldt, Project Owner on my project, System Architect, Brunata A/S
- Søren Kofoed Nielsen, Head of Laboratory, Brunata A/S

  For help with Requirements for encryption of radio telegrams.
- Søren Boisen, Senior Systems Developer, Brunata A/S
- Peter Rahlff Friis, Systems Developer, Brunata A/S

Contents

CHAPTER 1

Introduction

This chapter introduces the project and explains Brunata's business. It also elaborates on the background for the project.

1.1 Readers Guide

There are 7 main chapters of this report:

- Introduction - Chapter 1

  This chapter contains an introduction and background for this project.

- Process - Chapter 2

  Chapter about the processes of this project.

- Inception - Chapter 3

  This chapter establishes risks and project scope.

- Elaboration - Chapter 4

  This chapter elaborates on the project scope and establishes requirements.

- Construction - Chapter 5

  This chapter describes the construction of the system, as well as the testing of the system.

- Reflections - Chapter 6

  This chapter reflects upon the project and its sprints.

- Conclusion - Chapter 7

  This chapter concludes upon the project.

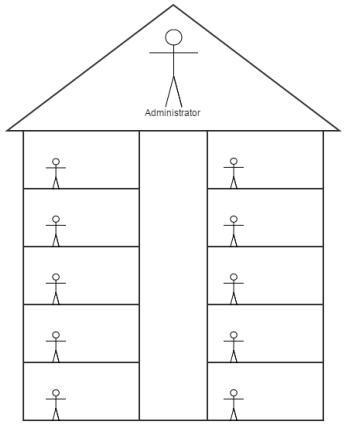

Figure 1.1: Apartment Building

1.2 Brunata A/S

Brunata is a Danish, family owned business, with headquarters in Herlev. They have almost a 100 years of experience in consumption measurement and consumption allocation accounts for the housing sector. Their primary source of revenue is delivering heat cost allocation accounts to housing associations and firms. The allocation accounts are made based on data collected from meters, installed at the customers. Brunata uses both meters developed and produced by themselves, as well as meters from third party manufacturers.

1.3 Brunata's Business

Brunata's Business is best explained by an example: In figure 1.1 is an apartment building, it has ten tenants and one administrator, the administrator is the potential customer of Brunata. The administrator divides the heating bill equally among the ten tenants.

One of the tenants does, in a period, not use any heat. The tenant still pays just as much as the other tenants, this is not fair.

This is where Brunata comes in. Brunata can via Heat allocation meters distribute the bill, so it correlates with each tenant's actual heat consumption.

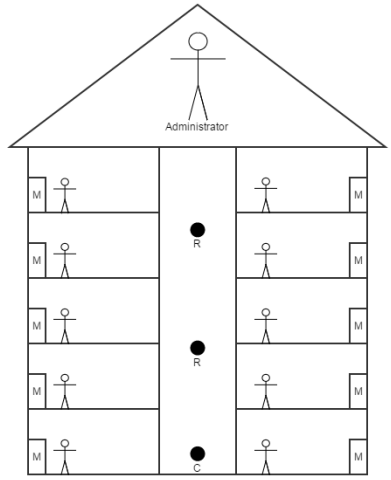

Figure 1.2: Apartment Building with meters installed

1.4 How it Works

Brunata has installed meters in an apartment building. There are 3 different parts as seen on figure 1.2:

- Meters installed on radiators - M
- Radio Receivers - R
- Collector - C

All the meters broadcast a radio signal, which the Radio Receivers receive. The format of this signal is Brunata's own IMR protocol, which is not an open standard. The Radio Receivers then sends its data to the Collector. The Collector then has the responsibility of sending the data to Brunata. From the data Brunata receives, a Heat Allocation Account is created, this describes how much heat each tenant has used.

1.5 Ulfberht

Brunata's current system for receiving meter data has been deemed ready to replace by Brunata. Brunata has therefore launched project Ulfberht. Ulfberht is a highly scalable system for receiving and processing meter data and consists of 3 distinct parts, these are described in the next 3 subsections.

Reading Input System

This system is responsible for receiving all data from meters. It processes the data and sends it to the Storage And Processing layer, described in the next subsection.

Storage And Processing

This system is responsible for holding the data processed by the Reading Input system. It also has to make the data available for the Reading output System.

Reading output System

This systems main responsibility is to make data from the Data Storage system available for the systems, calculating heat allocation accounts.

1.6 The Project

As specified, Brunata needs to use individually encrypted meters, and this needs to be supported in Ulfberht.

Currently most of the meters in use, are of Brunata make and uses Brunata's own closed protocol. Brunata already uses meters from third party manufacturers, most of these use an open protocol, wireless m-bus, described in the next section 1.7.

Both Brunata and the third party meters use a broadcast mechanism for transmitting their data.

Brunata uses their own closed protocol for transmitting data, IMR protocol, this protocol is only understood by entities with knowledge of this protocol. The 3rd party meters Brunata utilizes, uses the wireless m-bus protocol, which can be looked up and potentially understood by anyone. The 3rd party meters therefore use encryption on these meters, so only entities with the encryption key can interpret the data. This is illustrated in figure 1.3. Brunata, who has the encryption key, can understand all three meters, and the 3rd party entity can only understand the unencrypted WMBus meter.

That is the reason encryption on the meter data is needed. On the few third party meters Brunata operates with, the data is already encrypted, but these are not supported in Ulfberht yet. The goal of this project is to support encrypted meters in Ulfberht for Brunata. There are three parts to this:

- Storage and management for the encryption keys.
- Decryption of Data.

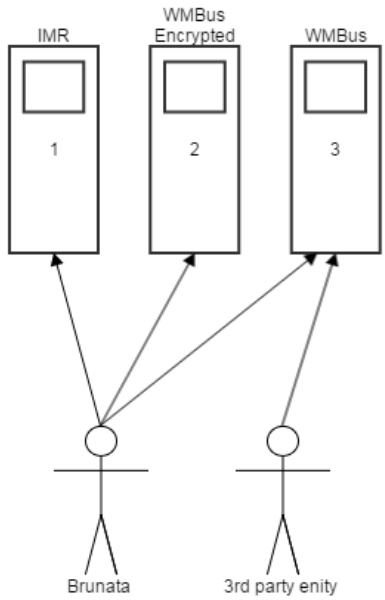

Figure 1.3: wireless m-bus vs. wireless m-bus Encrypted vs. IMR

- Integration into Ulfberht - RIS.

1.7 Wireless M-Bus

Wireless m-bus is a european standard for wireless remote reading of consumption meters.

CHAPTER 2

Process

This chapter discusses and decides on the process of the project. It elaborates on different Development methods, both those chosen and those in contention.

2.1 Project Management

All of Brunata's software projects are set up as Scrum projects. Naturally this project is also set up as a Scrum project.

Bo Visfeldt is Project owner and Morten Heebøll is the Scrum Master. Next section will elaborate on Scrum.

2.2 Scrum [SS13], [Wik14d]

Scrum is an iterative, agile software development framework for managing software development. The key principle is the realization that the requirements will change during the development process. In Scrum there are several roles:

- Project Owner

  The voice of the customer, who is responsible for ensuring that the Development Team delivers value to the business.

- Development Team

  Consists of 3 - 9 people, with cross-functional skills, who are responsible for delivering potentially shippable increments(PSI's) of the product.

- Scrum Master

  Facilitates the Scrum process and ensures that the Scrum process is followed and used as intended.

Sprint

In figure 2.1 is a graphical representation of the process of Scrum. A Product Backlog is a collection of tasks, when the tasks are completed the product

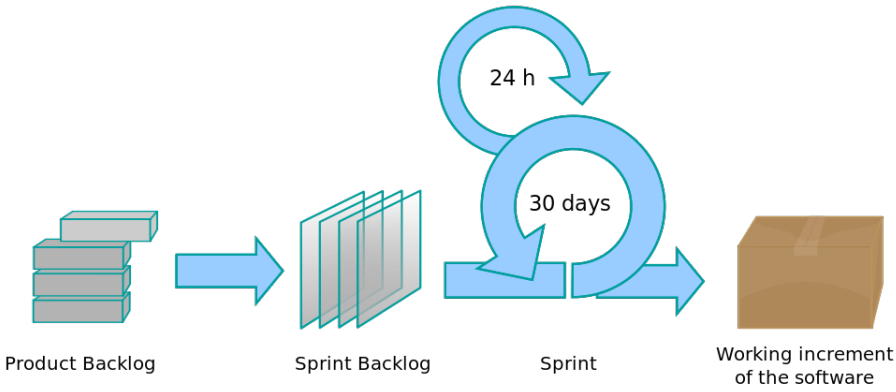

Figure 2.1: Scrum process overview [Wik14d]

is considered finished. A Sprint Backlog is a collection of tasks, which are meant to be completed during the sprint.

A sprint is the basic unit of development in Scrum. A sprint is usually timeboxed, with a duration varying from one week to one month. The duration and the content of the sprint is determined before the sprint starts in a planning meeting, which is described in the next subsection.

Planning

The first activity in the sprint. At this meeting the upcoming sprint is planned, the goals of the meeting are:

- Determine what work will be done.
- Estimate of how much time each task will take.
- Prepare the sprint work backlog, which is a prioritized list of tasks to be done.

Daily Scrum

A meeting held each day during the sprint, it is time boxed to 15 minutes. During the meeting every team member will answer 3 questions:

- What did I do yesterday that helped the Development Team meet the Sprint Goal?
- What will I do today to help the Development Team meet the Sprint Goal?
- Do I see any impediments that prevents me or the Development Team from meeting the Sprint Goal?

The answers to these questions helps the team collaboration and helps identifying problems early on.

Grooming

This is not a core scrum practice, however it is widely used in Brunata. It is a meeting in the middle of the sprint, where the sprint backlog is reviewed. Some tasks may be cut away, others may just be reduced or modified.

Sprint Review

A meeting held at the end of the sprint, which serves to inform the stakeholders of the project status. Review of both the completed work and planned work that was not completed. The completed work will presented and maybe demo'ed to the stakeholders.

Sprint Retrospective

A meeting held at the end of the sprint after the Sprint Review, in which team members will reflect upon the sprint that has passed and evaluate upon it. Two main questions are asked:

- What went well during the sprint?
- What could be improved in the next sprint?

This meeting serves to improve upon the process and tailor it to the Development Team.

How Scrum is implemented in this Project

Scrum is widely used in Brunata's Software projects and is also used in project Ulfberht.

It was set forth by Brunata that this project should also follow Scrum.

A project plan with four sprints was created, see figure 2.2 and for big version: figure B.1 in the appendix.

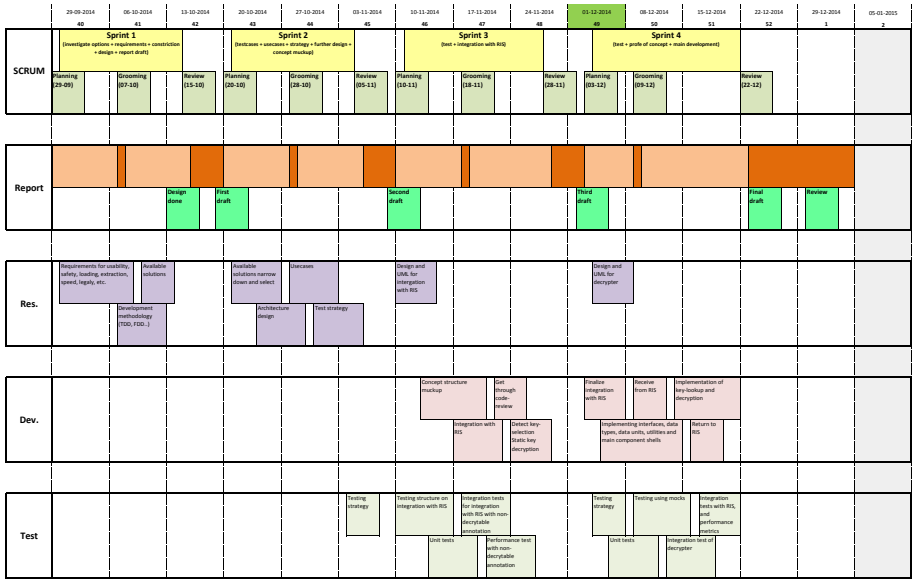

Figure 2.2: Final Project Plan, Big version: B.1

Before every sprint there will be a planning meeting, where the Scrum Master and the Project owner will attend. There will be a grooming in the middle of the sprint and a review and a retrospective at the end of the sprint.

2.3 Development Methodology

A development method helps you run the project. It sets the structure and helps to break down very difficult tasks to more manageable chunks.

Test driven development

The main focus of Test driven development is securing quality and eliminating bugs.

The basics of this development method is to write test code before writing functionality code. The cycle of this development method is:

- Write the test.

- Run the test to assure that it fails.
- Write code that makes the test pass.
- Refactor code for the purpose of removing duplication, optimizing code and following code standards.

This development method focuses on writing quality code. [Lar04].

Feature driven development

Feature driven development is defined by its five main activities:

- Develop overall model.

  Figure out the scope of the project.

- Build feature list.

  What features should the product support.

- Design by feature.

  The features are designed.

  Designs are validated via a design inspection.

- Build by feature.

  The features are built.

  The features must be validated via unit tests and code inspection.

The development method focuses heavily on features. [Wik14c].

Extreme Programming

"Extreme programming (XP) is a software development methodology which is intended to improve software quality and responsiveness to changing customer requirements" - [Wik14b].
The main focus of Extreme Programming is customer satisfaction and communication between developers and customers is central.

In Extreme Programming the customer is not the one who pays for the system, but rather the users who are going to be using the system. The customer is of vital importance in Extreme Programming and one should always be at hand, available for questioning. Extreme Programming takes into account that requirements could change at any time and it plans for it.

Domain driven design

Domain driven design is centered around understanding and investigating the domain and furthermore focusing on the essence of the domain. It mostly focuses on projects with high complexity.

In the center of the development method is the main goal, all activities should support this goal. [Wik].

Conclusion

Based on my research into different development methodologies, described in the previous 3 subsections, the best development method for this project, is a mix Feature driven development and Test Driven Development. Test Driven Development is choosen because it focuses on quality of code, which is of high importance to Brunata.

In the project features will be added to an existing system and since Feature driven development is heavily feature focused, it fits the project well. Feature Driven Development will be the dominant and aspects of Test Driven Development will be used.

2.4 Test Strategy

For each Feature:

1. Define acceptance criteria.
2. Write the functionality.
3. From the acceptance criteria write tests to test the functionality. If the tests fail, fix the functionality so it passes.
4. Optimize and Refactor the code, so it adheres to code standards. It is expected that every function has a unit test and features should have integration tests, when applicable.

The framework used will be NUnit [Nun14].

CHAPTER 3

Inception

In this chapter the justification for the project will be established and the scope of the project will be defined. Risks will be identified.

3.1 Justification

The main justification for this project is, that Brunata needs to start using individually encrypted meters. This will require support systems for handling these. It will require a way to decrypt meter data and it will require a storage for decryption keys, these two are not necessarily two individual systems.

3.2 Risks

The end product needs to handle large quantities of decryption keys, potentially numbering in the thousands. This presents a specialized case. Research will determine whether an out of the box storage system can be found. If a suitable out of the box key storage cannot be found, Brunata will have to develop one. This is a quite large endeavour and will not be within reach in the time allotted for this project.

3.3 Project Approach

This project is setup in Scrum and in chapter 2 about Process this is elaborated upon.

CHAPTER 4

Elaboration

In this chapter legal and usability requirements will be defined, as well as general requirements pertaining to the integration with the established system.

4.1 About Encryption

There are several ways of having encrypted meters, see figure 4.1.

- Type encryption

  All meters of the same type, from the same supplier is encrypted using the same key.

- Batch encryption

  A collection of meters, typically of the same type and same supplier, is encrypted using the same key.

- Individual encryption

  One meter has one encryption key.

As for the encryption itself there are two main types of encryption: Public Key Encryption and Symmetric encryption as seen on figure 4.2.

Symmetric key encryption is pretty straight forward, the same key is used for both encryption and decryption. Public key encryption has two keys: the public key, which anyone can use to encrypt data with, but only those with the private key will be able to decrypt the data and read it. Another feature of Public Key Encryption is authentication. Messages can be encrypted with the private key and can then only be decrypted with the public key, this ensures that the receiver knows that the message came from an entity with the private key.

Read more about encryption here [Wik14a]

4.2 Legal Requirements

What impact will Legal requirements have on the project.

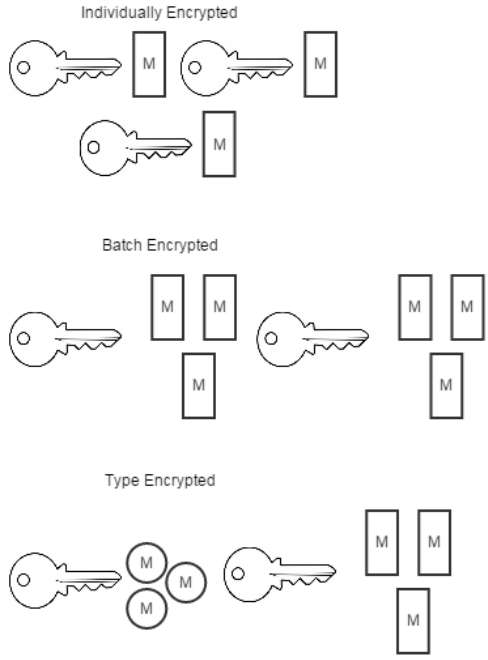

Figure 4.1: Meter encryption types

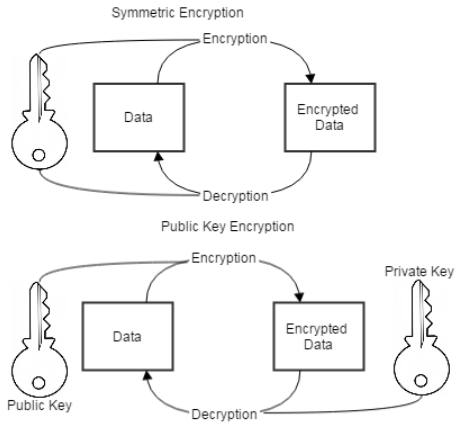

Figure 4.2: Symmetric vs Asymmetric Encryption

Are Decryption Keys Personal Information?

The keys provide access to personal meter readings, but does that make the keys themselves personal information? It can be argued that because the keys unlock personal information, in the form of meter readings, the key itself is also to be considered personal information. On the other hand encryption is a part of transporting the personal information securely. Also if the person the personal information pertains to has rights to the data, he can be given access to the data, without providing the keys.

There is therefore no reason to give the decryption keys to the person the data pertains to. Another argument for not allowing the person access to the decryption keys, is that the person, even with the decryption keys, will not have direct access to the data, since the person, in most cases, do not have the necessary equipment and knowledge to extract the data directly from the meter.

The keys themselves should not be considered personal information, and only the owner of the keys, in this case the owner of the meter, have rights to the keys.

In order for Brunata to have access to the keys, it has to be in the service agreement, in most cases between Brunata and the Housing Association, who, in the majority of cases, is the owner of the meters and therefore also the keys.

Who is the legal owner of the decryption keys

The owner of the decryption keys is the entity that also owns the meters, this is has been established by Brunata. If Brunata wants to service a customer, with meters that encrypt data, there has to be an agreement, allowing Brunata to use these keys.

Requirements from EU's Measuring Instruments Directive

Since the actual directive is heavy judicial reading, A Software guide from WELMEC [wel12] is used, specifically section 7 about transmission of measurement.

Completeness of transmitted data

"The transmitted data must contain all relevant information necessary to present" or further process the measurement result in the receiving unit." [wel12]

This requirement is not inside my project scope, since it pertains to what data the meter must send.

Protection against accidental or unintentional changes

"Transmitted data shall be protected against accidental and unintentional changes." [wel12]

[wel12] further explains that a way to achieve this, you can accompany the data with a checksum, using the CRC-16 algorithm. Checksum is to be calculated by the sender and the receiver, then the receiver checks its result against the senders calculated checksum.

Integrity of data

"The legally relevant transmitted data must be protected against intentional changes with software tools." [wel12]

Similar to the requirement above, however this requires to check if someone intentionally changed the data. To achieve this [wel12] again recommends CRC-16 Checksum, but here the initial vector of the CRC-register or the generator polynomial, must only be known only by the programs generating and verifying the checksums, if these are to be transmitted they must be treated as keys and is subject to the requirement: Confidentiality of keys. Another acceptable solution put forward is using a hash algorithm(SHA-1 or RipeMD-160) in combination with a encryption algorithm, such as RSA(768 bits) or Elliptic Curves(128-160 bits). This can also be achieved by using the transmission protocol HTTPS.

Authenticity of transmitted data

"For the receiving program of transmitted relevant data, it shall be possible to verify the authenticity and the assignment of measurement values to a certain measurement." - [wel12]

[wel12] suggests that each dataset has a unique identification number. Each dataset should also have information about the origin of the measurement data. Since this pertains to the meter itself, this requirement is mostly outside the scope of my project. The receiver of the data set should check all data for plausibility, this pertains to the input system and is also outside my project scope.

Confidentiality of keys

"Keys and accompanying data must be treated as legally relevant data and must be kept secret and be protected against compromise by software tools." - [wel12]

[wel12] suggests that secret keys are stored in a Hardware part that can be physically sealed, my interpretation of this, is that the keys can be stored on a server, that is placed in a server room that can be locked and has very

limited access. This is already the case with Brunata' server room, so this wont be a problem. Another suggestion is that the keys are stored in binary format in the executable code of the relevant software, this opens up for keys being stored in the meters.

Handling of corrupted data

"Data that are detected as having been corrupted must not be used." - [wel12]

[wel12] suggests that in case corrupted data is detected, it tries to restore from redundant data, if available. If this fails it should give a warning to the user of the problem. The data should then either be marked with either "not valid" or delete the corrupted data set.

Transmission delay

"The measurement must not be inadmissible influenced by a transmission delay." - [wel12]

[wel12] suggests an implementation of a transmission protocol that supports this, also the data itself must not be influenced by transmission delays.

Availability of transmission services

"If network services become unavailable, no measurement data must get lost." - [wel12]

In Brunata' case the meters are not interruptible, in this case [wel12] suggests, that if it has no network service, that it continues measuring and sends the cumulative data, when the network service is back.

Discussion of possible future legal requirements

There has been rumours, that the EU will be setting up directives, that require meter data to be encrypted. As of now, there are no concrete requirements, regarding encryption from the EU, this has been verified by Brunata.

- Worst case will be, that each meter has to have individual asymmetric encryption, meaning no two meters, of the same type, encrypt their data using the same key.
- Best Case is that meter data will only be required to be encrypted, but that there would be no requirement, saying that two meters can't use the same key. This is very unlikely, since you could encrypt all meters with the same key, meaning if you break the encryption for one, you break if for them all.

- The most likely case will be individual symmetric encryption, since having asymmetrical has no tangible benefits.

  Short of brute forcing the encryption, your best way of getting the key, will be through reverse engineering the key from the meter. Chances are, if you already have access to the meter, you already have the right to access the data it protects, otherwise you broke in and stole the meter. So basically with symmetric encryption you have two vulnerable points, the meter itself and the key storage.

  In asymmetrical you only have 1 vulnerable point in the key storage. I would not think it feasible to steal meters to get their keys, so since it takes less computing power encrypting with symmetric encryption compared to asymmetric encryption, symmetric seems the most likely candidate for encrypting meters individually. Another requirement the EU might put up is the type of symmetric encryption. The smart thing would be if they used one of the existing algorithms.

Conclusion

There is not much in the legal requirements researched, that can be acted upon in this, since the requirements described pertains to the transmission of the data itself, the components that transmit the data and the system that receives the data. This project is about handling keys and they decryption of the data.

4.3 Usability Requirements

As this is a new system in Brunata, the Responsibility of key loading and key administration is currently undefined.

- First case:
  Brunata orders meters, that encrypt data transmission. The department that has responsibility for ordering meters, will also have the responsibility for populating the system with the keys for the meters they order. In Brunata this is Supply Chain management department.

- Second case:
  Brunata takes over meters, that encrypt their data, from a competitor to Brunata. The department that has the responsibility for taking over the meters, will also have the responsibility for populating the system with the decryption keys, belonging to those meters. In Brunata this would be the sales department.

- Third case:
  Brunata already has meters that encrypt data where keys are not present in the system. In this case it is not clear who would have the responsibility. Brunata has a Team of service technicians, who have the responsibility for servicing the existing meter installations. In Conclusion, it would only be natural, that the service technicians have responsibility for populating the system with the old decryption keys.
- Administrator:
  Now the three prior cases are all users, administrator is another discussion. The administrator should assign rights to users, who has access to what, who can delete, who can update, who can insert keys. Now Brunata already has an IT department that has responsibility for running several systems, both purely internal systems and business systems. It would only be natural, that they also administer this system, ofcourse with support from R&D when problems occur.

4.4 Criteria for Key Storage System selection

Currently Brunata has 1.8 million meters that we hear from, if I accept the premise that in the future all meter data will need to be individually encrypted, Brunata will need a key storage to support at least 1.8 million Encryption keys. Furthermore Brunata has a wish of growing substantially, this means that any encryption key storage has to scale to support 2-3 times the current number meters.

Support for AES 128 bit Counter mode Encryption. This is a requirement of a meter Brunata plans to use, Multical 21. [Kam13]

Furthermore the system should support a wide range of algorithms, so that when brunata decides to use a new type of meter, the system will have decryption and key handling support for the new meter.

Brunata has not put up any requirement for the platform the system should run on, but for the purpose of this project, I will have to be able to evaluate the system.

No new hardware will be ordered to support the project, therefore any solution that can only be ordered as a full Hardware Solution is disqualified.

The code that needs to connect to the system is C# .Net, there needs to be support for this.

4.5 Key Storage System Candidates

Since brunata has no key storage system, criteria for system selection are needed.

Choosing

I have not chosen a Key Storage System. I have concluded that the resources involved with testing the different solutions, would be too much work to fit into the remaining project time. So I have decided not to include it in the remainder of my project. However the most promising system I have read about is Alliance Key Manager. The reason is that they offer the most diverse implementation methods including Cloud, Software and Hardware appliance. Alliance key manager is also the only one that has written that the maximum number of keys it stores is unrestricted.

4.6 Use Cases

There are 3 roles in the system. Ulfberht is the role of the system that this subsystem must integrate into. Admin is the role of those administrating the system. User is the role of those working with the system.

- Ulfberht:
  - Decrypt meter data.
- Admin:
  - Add decryption keys.
  - Remove decryption keys.
  - Edit decryption keys.
  - Assign user rights.
    Right to Add decryption keys.
    Right to Remove decryption keys.
    Right to Edit decryption keys.
- User:
  - Add decryption keys.
  - Remove decryption keys.
  - Edit decryption keys.

See Figure 4.6 for Usecase Diagram.

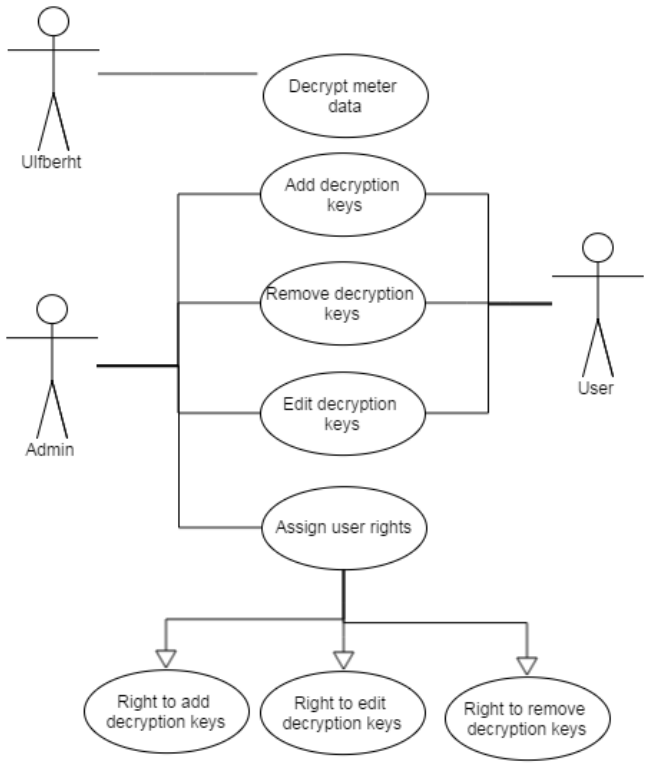

Figure 4.4: Use Cases for Decryption Key Management System

4.7 System Design and Architecture

When deciding how to facilitate decryption into Ulfberht, the most important thing was not to halt any processes while waiting for a decryption key, to decrypt data. That is why every time the Interpretation layer meets an encrypted telegram, it delegates the decryption task to the Decryption Gateway via the message bus. The Decryption Gateway consists of 2 components, one containing the decryption algorithms, the Decryptor, one, whose job is to allocate the encryption key, the Encryption Key Finder. For illustration see Figure 4.7

4.8 Requirements for integration with RIS

- Must not halt the main process in RIS.

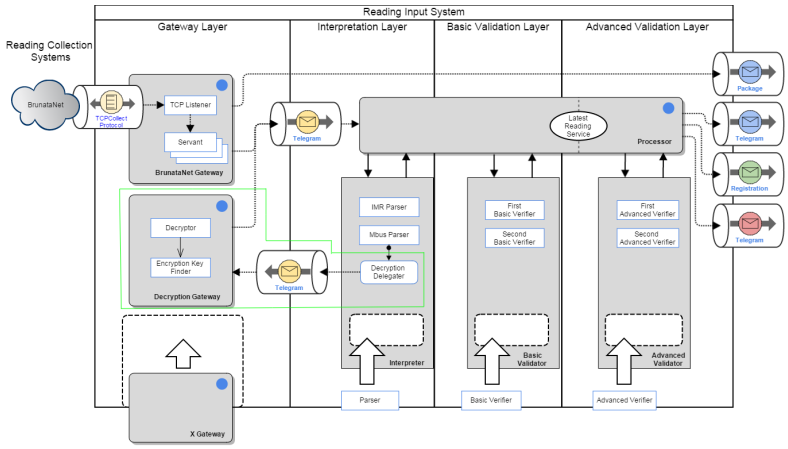

Figure 4.5: Design for Decryption Key Management System

- Must not break the processing of telegrams that currently process correctly.
- Cannot require significant changes in established data structures.
- Implement Ulfberht ShutdownHandler and ConsoleShutdownHelper, this is to support the overall system Coordinator.

Key Storage System	Apparent Suitability	Evaluation Available	Implementation types	Max No. of keys
Ideal Solution	Yes	Yes	Software Appliance	No limit
Oracle Key Vault Links See A.1	Yes	Yes	Software Appliance	-
Oracle Key Manager Links See A.2	Yes	No Eval found	As far as I can see Software Appliance	- -
SafeNet KeySecure Links See A.3	Yes	Yes	Hardware and Software Appliance	- -
Alliance Key Manager Links See A.4	Yes	Maybe, they are stalling	Cloud, Software and Hardware Appliance	Unrestricted
Bell ID Key Manager Links See A.5	No, it is for chip cards	- -	- -	- -
HP Enterprise Key Manager Links See A.6	Yes	No Eval found	Hardware Appliance	- -
RSA Data Protection Links See A.7	Yes	No Eval found	Cloud, Software and Hardware Appliance	- -
StronKey Links See A.8	Not supported anylonger	- -	- -	- -
Thales KeyAuthority Links See A.9	Yes	No Eval found	Hardware Appliance	- -
Vormetric Key Vaulting Links See A.10	Yes	No Eval found	Hardware Appliance	- -

Table 4.3: Evaluation Matrix

CHAPTER 5

Construction

This chapter describes the construction of the system. It Elaborates on both Design and implementation as well as the testing of the system.

5.1 Integration with RIS - Design

The processor has a package of 25 telegrams, the objective is to find out which of these are encrypted, so that they can be sent to the Decryption Gateway.

Flow for one encrypted telegram, diagram shown on figure 5.1:

1. The processor sends the telegram into the Interpreter.
2. The Interpreter tries to parse the telegram with the parsers.
3. A parser reports back to the Interpreter that the telegram payload is encrypted.
4. The Interpreter reports back a Status code to the Processor, which indicates if it needs to be send to the Decryption Gateway for decryption.

5.2 Integration with RIS - Implementation

RIS is the receiver of all telegrams and it is in this system the encrypted data will have to be decrypted. For each telegram which has encrypted data, there is a need for a key and for the sake of data security, the keys to the encryption will have to be stored separately from the system, so naturally in the key retrieval process there will be wait time on the delivery process.

So in order not to halt the process of processing telegrams, the decryption system is implemented in its own process. To support this, RIS will have to sort out any telegrams with encrypted data and send it to the decryption system. This is the first focus of the integration with RIS. The decryption system will also need information about how to decrypt the data, it has to know the manufacturer, the meter type and the meter number in order to retrieve the key and decrypt the data. Since RIS already have to detect

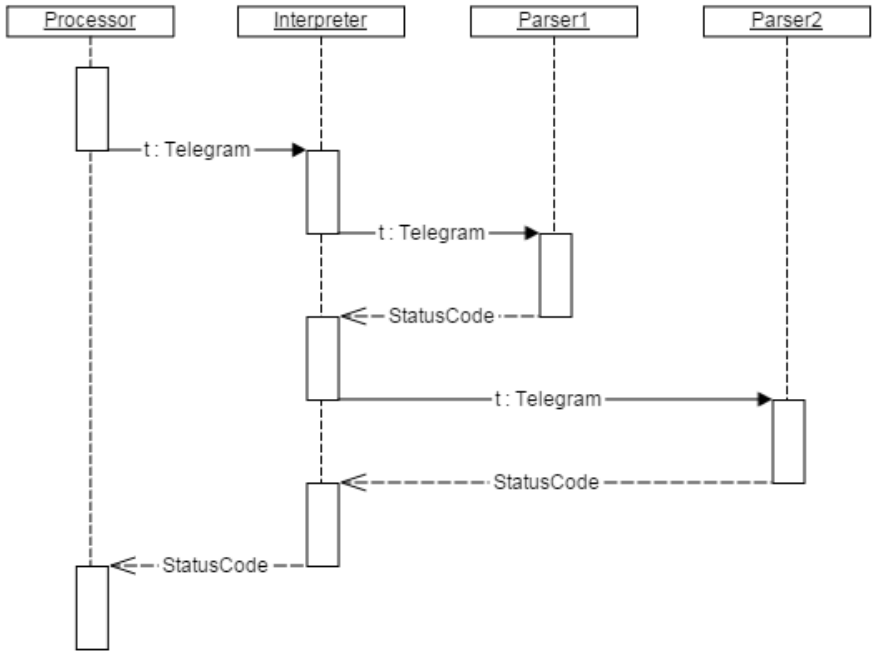

Figure 5.1: Integration With RIS Flow diagram

whether or not the telegram has encrypted data, it might as well, from the telegram payload, retrieve the relevant information and find out what parts of the telegram is encrypted.

In RIS the parsers will have the responsibility to detect whether or not a telegram is to be sent to the decryption system. Most, if not all telegrams, with encrypted data will be wrapped in a IMR protocol wrapper, this wrapper also contains information about where the telegram came from, this is annotated in the Metadata. The telegram payload will then have to be unwrapped from the IMR wrapper. This will be done in the class IMRWrappedExtractor.

IMRWrappedExtractor will then send the unwrapped telegram to other parsers, most telegrams will at this point be Wireless M-Bus(WMBus) telegrams and this is the only telegram type that will be implemented. The parser will then decide whether it is a WMBus telegram, that has encrypted data. There are 4 different returns available to the parser:

1. ENCRYPTED

   Encrypted data detected, meaning send to decryption system.

2. OK

   No encrypted data detected, but it does recognize the telegram and has parsed it.

3. NO_MATCH

   Does not recognize the telegram.

4. INVALID_TELEGRAM

   The telegram is corrupted, this is an error state.

5.3 Decryption Gateway - Design

In figure 5.3 we enter a state where the Decryption Manager has a telegram, t, that it wants decrypted. With KeyFinderDB it finds the key pack needed for decryption. The KeyPack holds information pertaining to how the telegram is encrypted including key and an identifier of which Decryptor to use. The Decryption manager finds in the DecryptorLookup what decryptor to use and sends the telegram and the key pack, kp, to the decryptor, in this case the AES128BitDecryptor, and the decryptor returns the decrypted telegram, dt, to the Decryption Manager.

5.4 Decryption Gateway - Implementation

The Decryption Gateway is responsible for decrypting the encrypted data in telegrams. The decryption could also have been implemented directly in the processor of RIS, however there are certain advantages to having the process in its own process. It simplifies the process and separates the responsibility. The retrieval of keys potentially relies upon other systems, such as a Key Storage system, getting a key from an external Key Storage results in IO wait, which would halt the processing in the processor, by having it separated from the RIS Processor, the key retrieval process does not halt normal operation of the RIS processor.

For the purpose of implementing the Decryption Gateway, a Concept Diagram was made as seen on figure 5.4. The Decryption Manager will receive packs of telegrams, via the messagebus, from the RIS processor, it then for each telegram attempts to find a key with the KeyFinders, if that is successful, it attempts to decrypt the encrypted data with the appropriate Decryptor. When all telegrams has been processed, the Decryption Manager returns the telegram pack to the RIS processor via the messagebus.

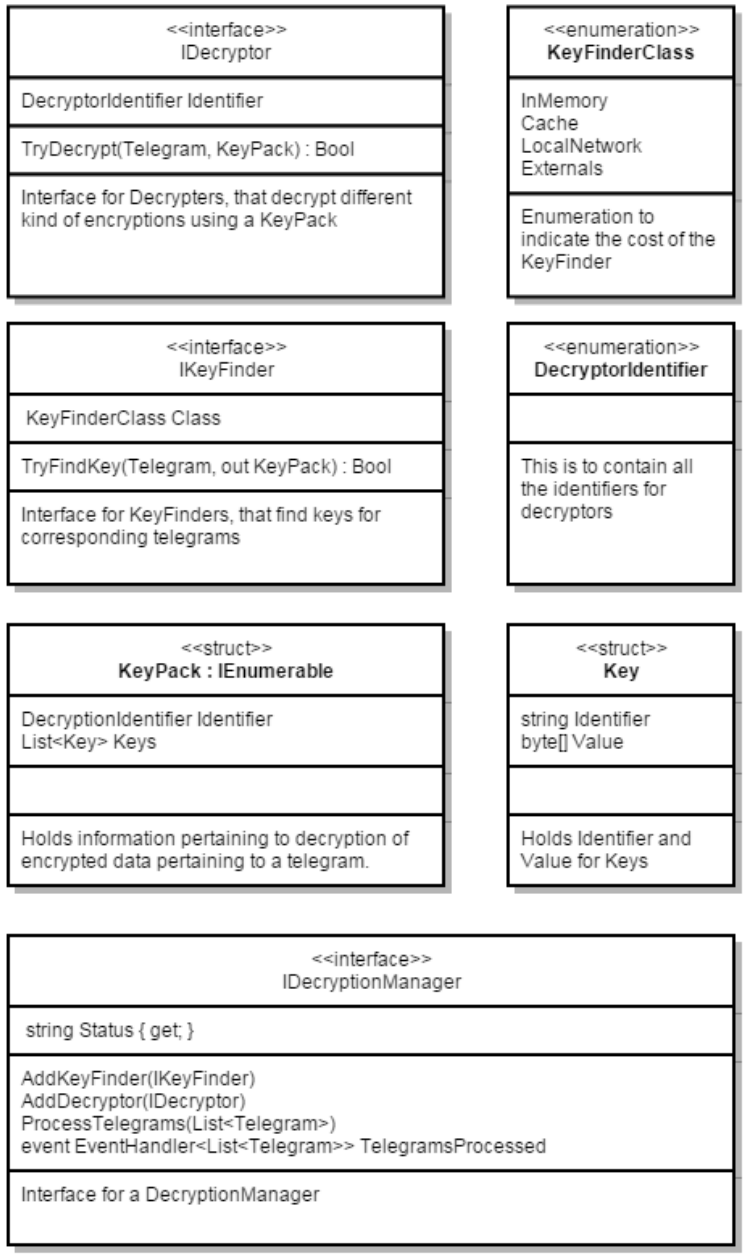

Figure 5.2: Class Diagram of Decryption Gateway API

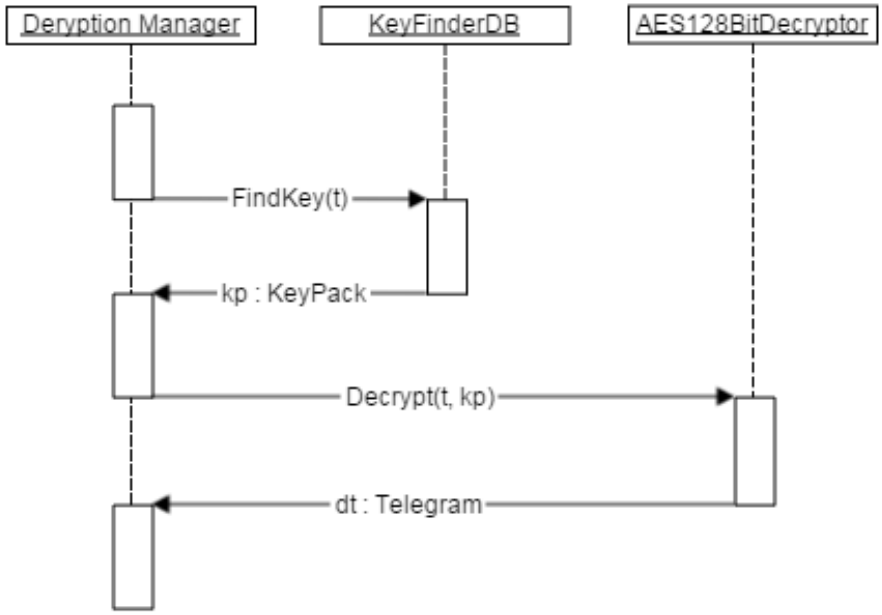

Figure 5.3: Flow Diagram of Decryption Gateway

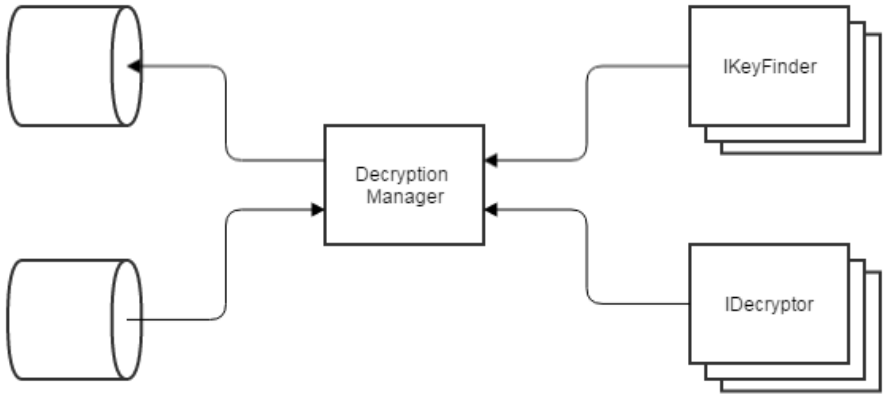

Figure 5.4: Decryption Gateway Concept Diagram

5.5 Decryption Gateway API

The Decryption gateway is implemented using interfaces, the advantages of this is having well defined entry points to the gateway, which also allows for testing against the decryption gateway, without using the actual implementation. It also allows for separation of concerns, so that if a change is made in the implementation of the decryption gateway, it does not interfere with other components.

The API consists of 6 elements as shown in the figure:

- IDecryptor

  The interface for creating a Decryptor. The Identifier identifies the decryption method used in the decryptor and TryDecrypt can then from a Telegram and a KeyPack decrypt the encrypted data in the Telegram.

- IKeyFinder

  The interface for creating a KeyFinder. The Class indicates the cost of finding the key in the implementation. TryFindKey is the method for finding a key, it will return true if successful.

- IDecryptionManager

  The interface for the component responsible for managing the process of decrypting data in a telegram. AddKeyFinder is for adding KeyFinders and AddDecryptor is for adding decryptors. ProcessTelegrams is for processing telegrams with encrypted data, with the goal of decrypting the data. TelegramsProcessed is the event triggered when telegrams have been processed.

- KeyPack

  The data structure for storing key information for decrypting a telegram. The Identifier identifies which Decryptor to use. Keys is the data structure for storing the key information itself.

- Key

  Holds one element of the key with a string, Identifier, to identify what is stored in the Value.

- DecryptorIdentifier

  Enumerator for identifying which Decryptor to use.

5.6 Testing

Two kinds of testing has been implemented, Unit testing and Integration Testing. This section elaborates on the testing techniques and the implementation of these. In this project all testing has been done using NUnit [Nun14]

Unit Testing

A Unit test tests the smallest testable part of the application. The biggest asset to Unit Testing, is being able to locate where an error occurs, this makes it much easier to correct set error. White-box testing, which is also used in this project, takes this even further. Basically a white box test is designed from the source code itself and is designed to test every statement in the code.

Example: When there is a switch case in the code, there will be a test case designed for each case in the switch. Thereby if a test case fails you know which case to check for errors.

Integration Testing

Integration Testing is a level testing where two or more components of the system are put together to test if they work together as intended.

Testing with Mocks of Interfaces [NSu14]

NSubstitute allows for creation of mocks of interfaces. The mocks can be modified in any way needed, for example: A IDecryptor mock can be modified to have a specific Identifier and the TryDecrypt can be modified to return true or false. The advantage of NSubstitute is that it allows for unit testing of components that rely on other components, without using these components.

Test Overview

- IMRWrappedExtractor
  - IMRWrappedInvalid(int positionToModify)

    Tests that the parser returns ParseResult.NO_MATCH, when different bytes in the payload are wrong.
  - IMRWrappedValidTelegram()

    Tests that a valid telegram with encrypted data, parses as ParseResult.ENCRYPTED in IMRWrappedExtractor.

  - IMRWrappedInvalidTelegramMetaDataNotSet()
    Tests that an invalid telegram, parsed through IMRWrappedExtractor, does not get certain MetaData fields annotated.
  - IMRWrappedMetadataCorrect(MetaData.Keys key)
    Tests that a valid telegram passed in IMRWrappedExtractor contains the correct values in MetaData.
  - IMRWrappedIsTimestampSetAndCorrect()
    Tests for correct TimeStamp when a valid telegram is parsed in IMRWrappedExtractor.
  - IMRWrappedEncryptedDataCorrect()
    Tests that a valid telegram parsed in IMRWrappedExtractor has the encrypted data in MetaData and that it is correct.

- WMBusParser
  - WMBusParserValidTelegram()
    Tests that a valid WMBus Telegram parses as ParseResult.ENCRYPTED.
  - WMBusParserValidTelegramCryptoData(MetaData.Keys key)
    Tests that when DecryptedData or EncryptedData is set in the Metadata on a valid telegram, parsing in WMBusParser results in a ParseResult.NO_MATCH.
  - WMBusParserNO_MATCH(int positionToModify)
    Tests that invalid telegrams return ParseResult.NO_MATCH.
  - WMBusParserMetaDataSet()
    Tests for correct MetaData on a valid telegram parsed in WMBusParser.

- DecryptionManager
  - AddKeyFinder()
    Tests the status after Mock KeyFinder is added and checks that it is correct and checks that the status reflects when a duplicate KeyFinder is added.
  - AddDecryptor()
    Tests the status after Mock Dercryptor is added and checks that it is correct and checks that the status reflects when a duplicate Decrypter is added.
  - ProcessTelegramCleanDecryptionManager()
    Tests that a Decryption Manager without any Decryptors or KeyFinders, annotates the telegram correctly in MetaData.

  - KeyFoundAndDecryptionSuccess()

    Tests that a decryption Manager with a Mock Decryptor, which returns true on TryDecrypt and a Mock KeyFinder, which return true on TryFindKey, correctly annotates the MetaData field decrypted as true.
  - KeyFoundAndDecryptionFail()

    Tests that a decryption Manager with a Mock Decryptor, which returns false on TryDecrypt and a Mock KeyFinder, which returns true on TryFindKey, correctly annotates the MetaData field decrypted as false.
  - KeyNotFoundAndDecryptionFail()

    Tests that a decryption Manager with a Mock Decryptor, which returns false on TryDecrypt and a Mock KeyFinder, which returns false on TryFindKey, correctly annotates the MetaData field decrypted as false.
  - KeyNotFoundAndDecryptionSuccess()

    Tests that a decryption Manager with a Mock Decryptor, which returns true on TryDecrypt and a Mock KeyFinder, which returns false on TryFindKey, correctly annotates the MetaData Field Decrypted as false.
  - TwoKeyFindersSuccess()

    Inserting two mock KeyFinders, one that returns false on TryFindKey, one that returns True and a mock decryptor that returns true on TryDecrypt. Tests that the MetaData field Decrypted is true.
  - TwoDecryptorsSuccess()

    Inserting two mock Decryptors, one where TryDecrypt returns false and one where it returns true. Test that the MetaData field Decrypted is true.

- RijndaelDecryptor
  - DecryptionCorrect()

    Tests that Telegrams with encrypted data decrypts correctly, when accompanied by the correct KeyPack.
  - DecryptionWrongKey()

    Tests that Telegrams with encrypted data does not decrypt, when accompanied by the wrong KeyPack.
  - DecryptionInvalidKeyPack

    Tests that Telegrams with encrypted data does not decrypt, when accompanied by an invalid KeyPack.

- InMemoryKeyFinder
  - CorrectKey()

    Tests that a telegram retrieves the correct KeyPack.

- Overall Integration Test: IntegrationTest()

  Tests the integration between the Processor and the Decryption System as a whole. Both with valid telegrams and with telegram with Invalid MeterNumber. It Tests that several MetaData Fields are correct.

- Processor Test: DecryptionIntegration()

  Tests that Telegrams of the kind with encrypted Data gets sent to the DecryptionGateway, while telegrams which is not of that kind gets processed as it should.

Unit Test Example

The example elaborated on in this section is the unit tests on the RijndaelDecryptor. As preparation for the unit tests, predetermined data has been encrypted with keys and inserted into telegram payloads. The keys used for decryption has been organized into KeyPacks, that when used in the RijndaelDecryptor can decrypt the encrypted data in the telegrams. The RijndaelDecryptor has been tested using three test cases:

1. Telegram with encrypted data in conjunction with the right KeyPack.

   TryDecrypt should return true and result in the correct unencrypted data in the DecryptedData field in MetaData.

2. Telegram with encrypted data in conjunction with a wrong KeyPack.

   TryDecrypt Should result in TryDecrypt returning false and the DecryptedData field in MetaData should not be set.

3. Telegram with encrypted data in conjunction with an invalid KeyPack.

   TryDecrypt Should result in TryDecrypt returning false and the DecryptedData field in MetaData should not be set.

The test code can be found in RijndaelDecryptorTests.cs.

Unit Test with Mocks Example

The example elaborated on in this section is the test that tests the DecryptionManager. In order to properly unit test the DecryptionManager, mocks of IKeyFinder and IDecryptor, using Nsubstitute, are utilized. There are 4 test cases for the decryptionManager:

1. A Key is found and a decryptor can decrypt.
2. A Key is not found and a decryptor a can decrypt.
3. A Key is found, but no decryptor can decrypt.
4. A Key is not found and no decryptor can decrypt.

To test these cases mocks that suit these test cases are created.

Integration Test Example

The integration test featured in this example is the overall integration test. This test combines the decryption gateway with the RIS processor. The main feature it tests is that Telegrams of a type that has encrypted data, gets this data decrypted. It further tests that it decrypts correctly and that the fields in the metadata are set as they should be. For the purpose of this test 4 telegram payloads were created. The first 3 has serial numbers that correspond with keys stored in the InMemoryKeyFinder and encrypted data encrypted with these keys, for the purpose of testing whether the decryption was correct. The last telegram has a serial number that does not correspond with a KeyPack stored in InMemoryKeyFinder to ensure that this case is also handled correctly.

The processor and the decryption manager invokes events when they have processed telegrams. The processor Invokes 4 kinds of events:

- EncryptedTelegrams

  When the processor detects telegrams with encrypted data, it invokes this event.

- RegistrationsProcessed

  When telegrams are interpreted Registrations are created.

- TelegramsNotRecognized

  When the Telegrams is not recognized by any parsers, this event is invoked.

- TelegramsProcessed

  When Telegrams has been processed in the processor, this event is invoked, this includes all but those invoked in EncryptedTelegrams.

The Decryption Manager invokes only one event, TelegramsProcessed, which is invoked when telegrams have been processed in the Decryption Manager.

All these events are hooked onto methods on the test class. When the decryption manager invokes the TelegramsProcessed event, it calls ProcessTelegrams on the processor. When the processor invokes EncryptedTelegrams, ProcessTelegrams in the decryption manager is called. This serves to emulate the system setup and test that the integration between the processor and the decryption manager.

To begin the integration test the test telegrams are put into a list and ProcessTelegrams in the processor is called with the list as the parameter. The processor should then parse the telegrams as encrypted, then send them to the decryption manager, which then will process the telegrams and send them back to the processor, which should now be unable to recognize the telegrams and invoke the TelegramsNotRecognized. Lastly the telegrams are tested for the correct MetaData.

5.7 Class Descriptions

Descriptions of the various classes implemented. A Class Diagram, providing an overview over the Decryption Gateway, can be seen in figure 5.7. A Class Diagram, providing an overview over the parsers, can be seen in figure 5.7.

IMRWrappedExtractor

The purpose of this class is to recognize and extract WMBus telegrams, for the purpose of finding encrypted data. It implements the IParser Interface, so it can be used along with the other parsers in RIS. TryParse validates several elements in the telegram before it tries to extract the WMBus telegram:

- Checks if it has encrypted data in its MetaData, which would indicate that it has been through the IMRWrappedExtractor before.
- Checks if the length byte in the payload is the same as the payload length.
- Checks for correct MeterType in the IMRHeader of the payload.

When these checks are done and the telegram is validated, it will extract what is supposedly the WMBus telegram and try to parse it in the WMBusParser.

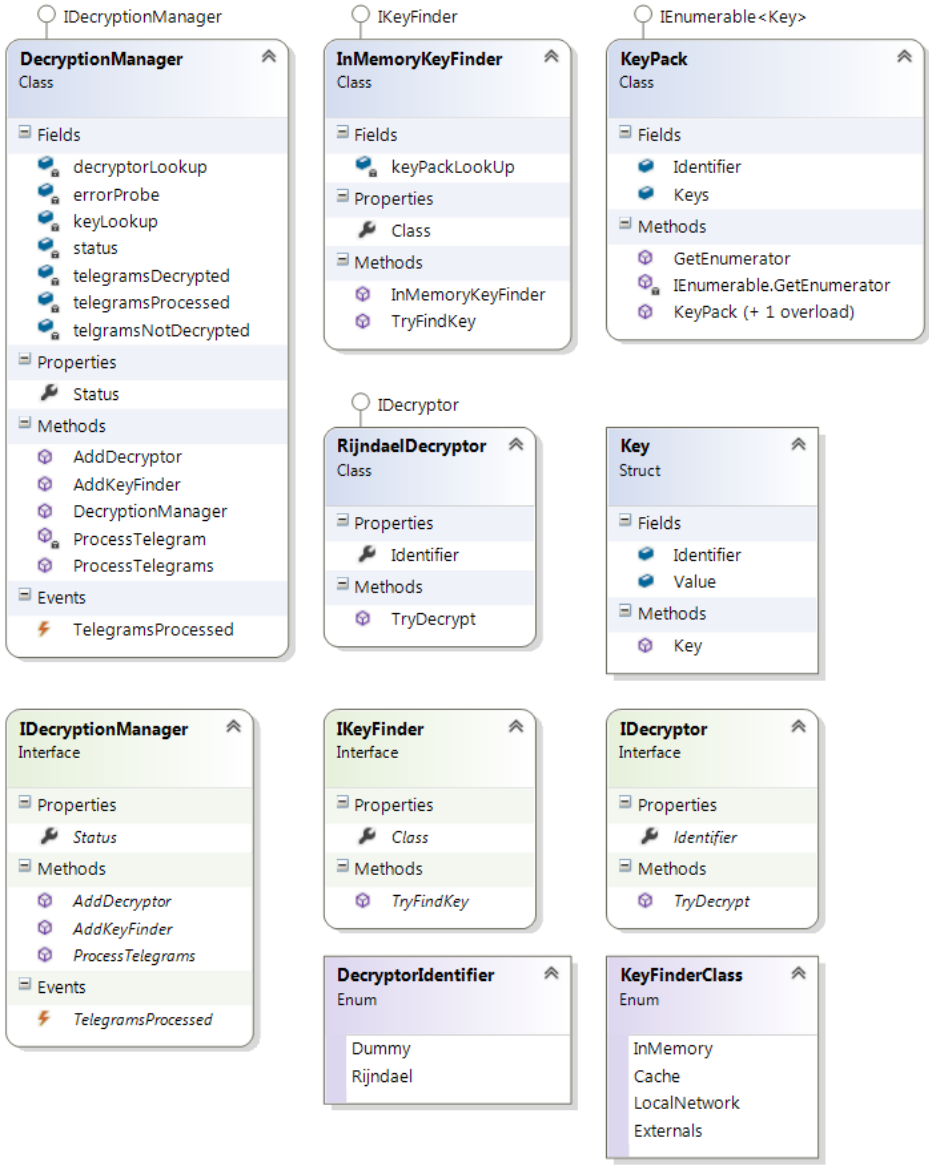

Figure 5.5: Class Overview - Decryption Gateway

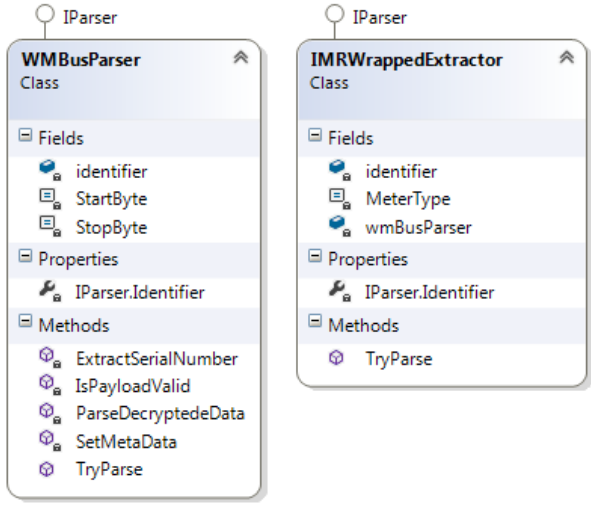

Figure 5.6: Class Overview - Parsers

WMBusParser

The purpose of this class is to validate and parse WMBus telegrams as well as extract encrypted data to be set in the MetaData. TryParse validates several elements in the telegram:

- Checks for parser hint in MetaData, if the hint is "WMBusParser", the telegram has already already been through the Decryption Gateway and had its encrypted data tried decrypted, if decryption was successful, it should parse the decrypted data, parsing of the decrypted data has not been implemented. ParseResult.NO_MATCH will be returned.
- If it has the hint, but not the decrypted data, the decryption in the Decryption Gateway failed, therefore ParseResult.NO_MATCh will be returned.
- It checks that the length byte of the telegram is correct.
- It checks for the correct start byte in the payload.
- It checks for the correct stop byte in the payload.

When all these checks have been made, the encrypted data is extracted and all the pertinent MetaData fields are set and TryParse will return ParseResult.Encrypted.

DecryptionManager

The main responsibility of the DecryptionManager is to manage and orchestrate the decryption process. From receiving telegram packs, to finding KeyPacks, to decrypting the encrypted data in the telegrams, all the way to sending the telegrams back to the processor. The DecryptionManager does not find encryption keys or decrypt any data itself, it delegates these tasks to KeyFinders and Decryptors, these can be injected into the DecryptionManager via AddKeyFinder and AddDecryptor.

In AddKeyFinder, any KeyFinder not already present in keyLookup is added to keyLookup, the keyLookup is then sorted after the Class field in the keyLookup, which indicates the cost of finding a key with the KeyFinder. This way the low cost KeyFinder's are tried before the more expensive ones are tried.

In AddDecryptor, any Decryptor with a decryptor identifier not already present in the decryptorLookup, is added to the decryptorLookup, which is a Dictionary where the key is the identifier and the value is the decryptor.

ProcessTelegrams is the method receiving the telegrams, in lists of telegrams, this method runs ProcessTelegram on each telegram.

ProcessTelegram orchestrates the process of decrypting data in one telegram. The process in this method is as follows:

1. Check for encrypted data, if none exists in the telegram, it sets the metadata to show that its data is not decrypted and then returns.

2. Tries to find a KeyPack, if none is found, it sets the metadata to show that its data is not decrypted and returns.

3. Tries to find a decryptor with the identifier in the found KeyPack. If a decryptor cannot be found it sets the metadata to show that its data is not decrypted.

4. Tries to decrypt the data, if it fails, it sets the metadata to show that its data is not decrypted and returns.

5. If decryption was successful it sets the metadata to show that the data has been decrypted and it returns.

RijndaelDecryptor and InMemoryKeyFinder

The RijndaelDecryptor and the InMemoryKeyFinder was created as a dummy implementation of the system, since no actual implementation was feasible. They are meant as a proof of concept to display how actual Decryptors and KeyFinders can be implemented.

RijndaelDecryptor

This component will decrypt encrypted data stored in a telegram's MetaData if accompanied with the correct KeyPack. The implementation is limited to decrypting data encrypted with RijndaelManaged from the library at System.Security.Cryptography. Also the KeyPack must contain 2 elements in Keys: key and vector, with its accompanied data.

InMemoryKeyFinder

This component serves as the container of 3 KeyPacks, KeyPacks with Keys and Vectors generated with RijndaelManaged, keys which have also been used to create dummy telegrams with encrypted data for testing purposes.

CHAPTER 6

Reflections

This chapter reflects upon the project and its sprints. It also describes the four sprints.

6.1 Sprint One: ”Scratching the surface”

The purpose of the first sprint is to investigate requirements for the project, this includes both usability requirements and legal requirements. A sub purpose of the sprint is investigating candidates for a Key Storage system and finding criteria for selecting final solution. Finally a development methodology must be chosen.

Sprint Thoughts

At the beginning of the sprint, there were certain preconceptions. One was that whatever Key Storage System found, it should run on a Windows server. It soon became clear that this was not possible. No system suitable for the project was running on Windows server. Most systems was linux based

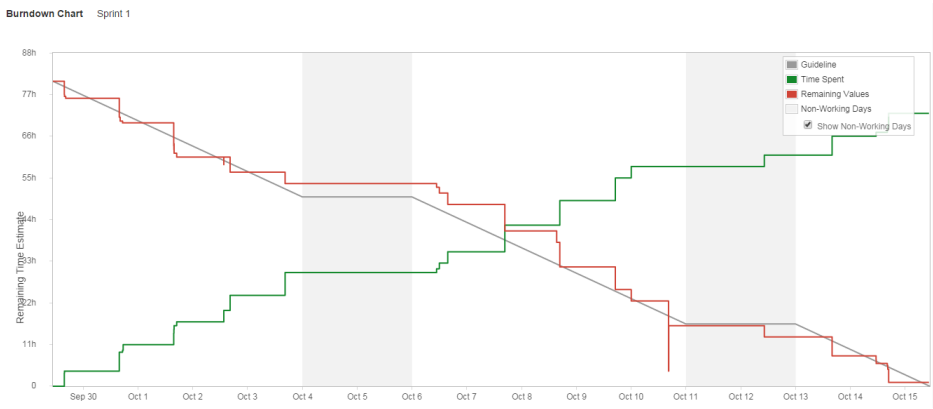

Figure 6.1: Sprint one burndown chart

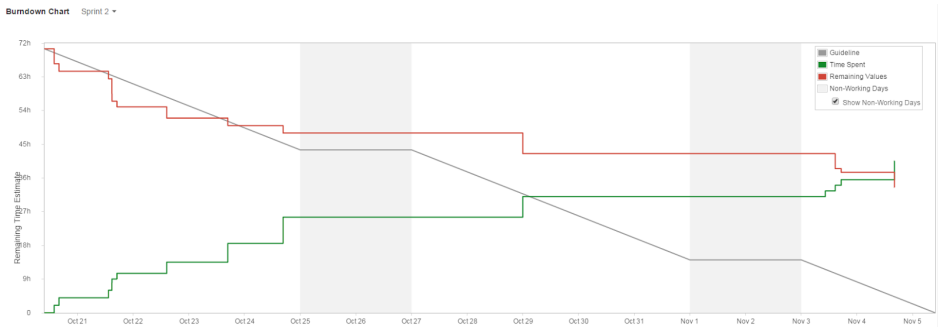

Figure 6.2: Sprint two burndown chart

and running their own OS. Some solutions were cloud based while others could run on WMware, but most came with their own hardware setup, full solutions with software and hardware sold together.

6.2 Sprint Two: "Digging deeper"

This sprint is about further research into the Key Storage tools available. Also planned is use cases, Main architecture design, further investigation into key retrieval methods and lastly there is Test Strategy

Sprint Thoughts

This sprint has been frustrating, filled with blockers and waiting on other entities, both internal and external, to procure resources that I needed. I have had my research blocked, by not having a static IP for testing Oracle Key vault. I have also have my research blocked by missing an evaluation key for Alliance Key Manager. Last blocker was that, I was supposed to talk to Kamstrup and Itron about their key delivery methods, my project owner, Bo Visfeldt(BVI), was supposed to facilitate contact, but it takes some time, due to other company interests. What I did do this sprint was:

- Use cases for the system
- Test Strategy
- Researching candidates for Key Storage and reducing the list of candidates to 2.
- Architecture

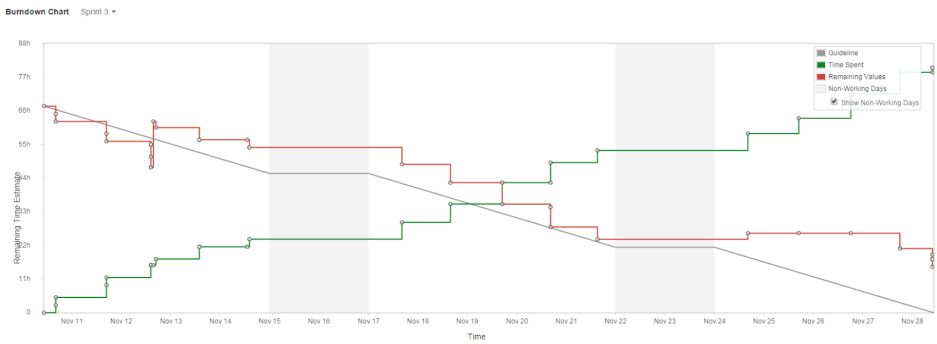

Figure 6.3: Sprint three burndown chart

6.3 Sprint Three: "Integration with Ulfberht"

This sprint is about the integration with RIS, modifying the existing system to sort out telegrams with encrypted data, so that it can be sent to the Decryption Gateway.

Sprint Thoughts

The integration with Ulfberht has gone as planned, the existing system now sorts out the telegrams with encrypted data. In figure 6.3 a burndown chart for the sprint can be seen.

6.4 Sprint Four: "Are we there yet?"

The main task of this sprint is to design and implement the Decryption Gateway, as well as a Decryptor to decrypt data from Kamstrup meters and a KeyFinder, working in memory, to store keys.

Sprint Thoughts

The files containing the keys for the Kamstrup meters, is encrypted, and while kamstrup has provided tools capable for decrypting with these files, via software from Kamstrup, the keys cannot be extracted. In order to provide a proof of concept of the system, RijndaelDecryptor and InMemoryKeyFinder has been made, they serve to show how Decryptors and KeyFinders can be implemented. Also the focus of this sprint changed during the course of the sprint, the focus changed to focus on the documentation of this project.

6.5 Individual vs. Universal decryption keys

Universal Keys

Pros:

- Fewer keys means it can be easily implemented in code(hard coded).

Cons:

- One key for many meters results in a single point of failure.
- If a key is lost many meters will become unusable. (assuming reprogramming is unfeasible)
- Meters cannot be resold without risking the data security of the other meters.

Individual Keys

Pros:

- One key for each meter results in no Single point of failure.
- If a key is lost it only impacts one meter.
- Meters can be resold without risking the data security of other meters.

Cons:

- Many keys will need a storage and management system for these.

6.6 Optimization

The DecryptionManager currently only sends one telegram at a time to a keyfinder, while this is no problem in the current InMemoryKeyFinder, it is not optimal in a KeyFinder implemented to look up in a database. With one telegram at a time one query would be executed for each telegram. Instead of a KeyFinder receiving one telegram at a time, it could receive multiple, so one query would result in multiple keys found, this would reduce the number of queries by a factor of the number of telegrams bundled.

CHAPTER 7

Conclusion

There are many aspects of this project, there are the encryption keys, which must to be managed, there is the encrypted data, which needs to be decrypted and lastly there is the integration with RIS.

The aspect of encryption key management has proven to be the biggest challenge. When reading about it, it is often described as one of the most difficult aspects in the field of Data security, this has throughout the project shown itself to be true. I have first of all not been able to find a suitable key storage system. Most of the available systems for key storage focus on securing databases or personnel credentials. Instead as a temporary measure and for testing a local database has been implemented.

7.1 Development Methodology

This project was set to run with full fledged scrum, this was set forth by Brunata. Early in the project Feature Driven Development and Test Driven Development was chosen as well. Feature Driven Development was chosen with the best of intentions, but in reality it was not used fully, only aspects of it was used, such as "Designs are validated via design inspection". From Test Driven Development only aspects of it was chosen and a Test Strategy was developed, this test strategy was not followed to the letter, since acceptance criteria was not defined for each feature. Scrum is made for a development team from 3-9 individuals, this project only had one, while many aspects of Scrum is useful for a one man development team, full fledged Scrum was too much. The aspects of Scrum that this project did benefit greatly from, was the basic structure of scrum, dividing the work into sprints, the planning and evaluating of the sprints.

7.2 Decryption Gateway

A Decryption Gateway has been implemented, and it can serve as a framework on which to implement actual KeyFinders, hooked up to a key storage and Decryptors, which decrypt actual encrypted meter data.

7.3 Key Storage

A suitable key storage, that match the requirements of this project, was not found. A total of 10 systems was researched and two was tested, Alliance Key Manager and Oracle Key Vault, these did not suit the their purpose. In conclusion, if Brunata is to use Individually encrypted meters, a key storage must be found, or one must be developed.

7.4 Integration with RIS

Two parsers has been implemented in order to facilitate the integration with RIS, IMRWrappedExtractor and WMBusParser. these parsers correctly assures that telegrams with encrypted data, is marked as such, then the RIS Processor has been modified, so that it sends these telegrams to the Decryption Gateway.

7.5 Testing

A myriad of tests has been created for this project, a total of 25, and yet more test cases. The tests confirm that the software written works as intended, both on a component level and on a Integration level.

7.6 Usage

The product produced in this project serves as a proof of concept of how handling individually encrypted meters in Ulfberht can be done. The product as it stands cannot serve a purpose in Ulfberht in its current state, however it can serve as a framework on which to build KeyFinder's that can find keys for live telegrams with encrypted data and Decryptors can be made to decrypt these data with the keys found via the KeyFinder's.

7.7 Future Work

The research into a Key storage system did not come up with any systems suitable for the key storage needs of this project, so for Brunata to be able to handle individually encrypted meters, more research is required and if a storage system still cannot be found, Brunata will have to develop their own.

APPENDIX A

Key Management System Candidates

A.1 Oracle Key Vault

Main Link

http://www.oracle.com/us/products/database/security/key-vault/overview/index.html

Data Sheet

http://www.oracle.com/technetwork/database/options/key-management/overview/ds-security-key-vault-2256707.pdf

A.2 Oracle Key Manager

Main Link

http://www.oracle.com/us/products/servers-storage/storage/storage-software/oracle-key-manager/overview/index.html

Data Sheet

http://www.oracle.com/us/products/servers-storage/storage/tape-storage/034335.pdf

FAQ

http://www.oracle.com/us/products/servers-storage/storage/tape-storage/oracle-key-manager-faq-444635.pdf

A.3 SafeNet KeySecure

Main Link

http://www.safenet-inc.com/data-encryption/enterprise-key-management/key-secure/

A.4 Alliance Key Manager

Main Link

http://townsendsecurity.com/products/encryption-key-management

Data Sheet

http://townsendsecurity.com/sites/default/files/AKM_DataSheet_2.pdf

A.5 Bell ID Key Manager

Main Link

http://www.bellid.com/products/bell-id-id-key-manager/

A.6 HP Enterprise Secure Key Manager

Main Link

http://www8.hp.com/dk/da/software-solutions/software.html?compURI=1336923#.VDPxL_m

A.7 RSA Data Protection Manager

Main Link

http://www.emc.com/security/rsa-data-protection-manager.htm

Data Sheet

http://www.emc.com/collateral/data-sheet/h9092-rsa-data-prot-mgr-ds.pdf

A.8 StrongKey

Main Link

http://strongkey.strongauth.com/

A.9 Thales KeyAuthority

Main Link

https://www.thales-esecurity.com/products-and-services/products-and-services/key-management-systems/keyauthority

A.10 Vormetric Key Vaulting

Main Link

http://www.vormetric.com/products/enterprise-key-management/key-vault

APPENDIX B

Process

B.1 Project Plan

Figure B.1: Final Project Plan

APPENDIX C

Code CD Guide

The CD contains more than just this projects code, it contains my code and its dependecies.

If opened in Visual Studio, open from ReadingInputSystem.sln. This project's code can be found in these directories:

- Decryption Gateway: Ulfberht/ReadingInputSystem/Gateways/DecryptionGateway
- IMRWrappedDetector: Ulfberht/ReadingInputSystem/Processing/Parsing/IMRWrappedDetector
- WMBusParser: Ulfberht/ReadingInputSystem/Processing/Parsing/WMBusParser
- Integration Tests: Ulfberht/Testing/DecryptionGatewayIntegrationTests
- Unit Tests: Ulfberht/Testing/DecryptionGatewayUnitTests
- Tests for IMRWrappedExtractor: Ulfberht/Testing/ReadingInputSystemUnitTests/ParsingUnitTests/IMRWrappedExtractorTest.cs
- Tests for WMBusParser: Ulfberht/Testing/ReadingInputSystemUnitTests/ParsingUnitTests/WMBusParserTests.cs

These files has also been modified in this project:

- Modified for integration with RIS: Ulfberht/ReadingInputSystem/Processing/InputProcessor/Implementation/Processor.cs
- Modified for integration with RIS: Ulfberht/ReadingInputSystem/Processing/InputProcessor/Implementation/Interpreter.cs
- DecryptionIntegration(): Ulfberht/Testing/ReadingInputSystemIntegrationTests/IntegrationTests.cs
- Modified for Integration With RIS: Ulfberht/DataTypes/MetaData.cs

Bibliography

[Kam13] Kamstrup. *Multical 21 Teknisk Beskrivelse*. Kamstrup, 2013.

[Lar04] Craig Larman. *Applying UML and Patterns - An Introduction to Object-Oriented Analysis and Design and Iterative Development*. Prentice Hall, 2004.

[NSu14] NSubstitute. *NSubstitute*. NSubstitute, 2014. http://nsubstitute.github.io/.

[Nun14] Nunit.org. *Nunit*. Nunit.org, 2014.

[SS13] Ken Schwaber and Jeff Sutherland. *The Scrum Guide*. scrumguides.org, 2013. http://www.scrumguides.org/scrum-guide.html.

[wel12] welmec.org. *WELMEC Software Guide*. welmec.org, 2012.

[Wik] Wikipedia. *Domain Driven Design*. Wikipedia.

[Wik14a] Wikipedia. *Encryption*. Wikipedia, 2014. Wikipedia.org/wiki/Encryption.

[Wik14b] Wikipedia. *Extreme Programming*. Wikipedia, 2014. Wikipedia.org/wiki/Extreme_programming.

[Wik14c] Wikipedia. *Feature Driven Development*. Wikipedia, 2014. Wikipedia.org/wiki/Feature-driven_development.

[Wik14d] Wikipedia. *Scrum*. Wikipedia, 2014. Wikipedia.org/wiki/Scrum_(software_development).

{2.7182818284}