

CHALMERS



Secure Collection and Analysis of Mobile Broadband Logs

Master of Science Thesis in Software Engineering & Technology

JÉRÉMY L. H. P. HALLDÉN

ERIK B. L. SVEDLUND

Department of Computer Science and Engineering
Division of Software Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Göteborg, Sweden, 2009

The Author grants to Chalmers University of Technology and University of Gothenburg the non-exclusive right to publish the Work electronically and in a non-commercial purpose make it accessible on the Internet.

The Author warrants that he/she is the author to the Work, and warrants that the Work does not contain text, pictures or other material that violates copyright law.

The Author shall, when transferring the rights of the Work to a third party (for example a publisher or a company), acknowledge the third party about this agreement. If the Author has signed a copyright agreement with a third party regarding the Work, the Author warrants hereby that he/she has obtained any necessary permission from this third party to let Chalmers University of Technology and University of Gothenburg store the Work electronically and make it accessible on the Internet.

Secure Collection and Analysis of Mobile Broadband Logs

JÉRÉMY L. H. P. HALLDÉN
ERIK B. L. SVEDLUND

© JÉRÉMY L. H. P. HALLDÉN, September 2009.
ERIK B. L. SVEDLUND, September 2009.

Examiner: WOLFGANG AHRENDT

Department of Computer Science and Engineering
Chalmers University of Technology
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

Department of Computer Science and Engineering
Göteborg, Sweden, 2009

Abstract

This thesis report contains a description of two successful system developments at the Mobile Broadband Modules department at Ericsson, the first being a system for secure collection of mobile broadband logs; the second being a system for analyzing such logs. Relevant fields studied to solve the different problems imposed by the systems' requirements, such a field being cryptography, and the solutions chosen for the systems and their software design and implementation are presented. The confidentiality of the log collection was fulfilled by implementing a solution using .NET library classes for the cryptographic algorithms RSA and AES. The log collection itself was realized by using various APIs. The analysis of the logs was made by translating network messages via an algorithm using resource files. High performance was reached by optimizing the string usage and high usability by system prototyping.

Sammanfattning

Denna examensavhandling innehåller en beskrivning över två framgångsrika systemutvecklingar på Mobile Broadband Modules-avdelningen på Ericsson. Det första är ett system för säker insamling av mobilt bredbandsloggar; det andra ett system för att analysera sådana loggar. Relevanta fält studerade för att lösa de problem förelagda av systemkraven, exempelvis kryptografi, och de valda lösningarna för systemen och deras mjukvarudesign och implementation presenteras. Konfidentialiteten av logginsamlingen åstadkoms genom att implementera en lösning som använder sig av .NET-biblioteksklasser för de kryptografiska algoritmerna RSA och AES. Själva Logginsamlandet genomfördes genom att använda diverse API:er. Logganalyserandet gjordes genom att översätta nätverksmeddelanden via en algoritm som använder sig av resursfiler. Hög prestanda nåddes genom att optimera sträng-användandet och hög användbarhet genom systemprototypning.

Keywords: Log collection, log analysis, Cryptography, Encryption, Decryption, RSA, AES, .NET, C#, String optimization, String comparison, Mobile broadband

Preface

This report was written as part of a Master Thesis at the department of Computer Science and Engineering at Chalmers University of Technology. We authors, Jérémy Halldén and Erik Svedlund, both have Bachelor of Science degrees in Information Technology and this thesis is part of our Master of Science degrees in Software Engineering and Technology.

The actual project was carried out at the Integration and Verification unit of the Mobile Broadband Modules department at Ericsson, Gothenburg, Sweden.

We would like to thank our supervisor at Ericsson, Johanna Boström, and our examiner at Chalmers University of Technology, Wolfgang Ahrendt, for their guidance and support throughout this thesis. We would of course also like to thank our friends, fellow students, colleagues, and also families, for their everyday support.

Finally, we hope that you as a reader will enjoy this report and find it useful and interesting.

Table of Contents

1	Introduction.....	10
1.1	Background.....	10
1.2	Goal	10
1.3	Methodology	10
1.4	Limitations and Delimitations	11
1.5	Overview	11
1.6	Related Work.....	11
2	Context	14
2.1	Mobile Broadband Modules.....	14
2.2	Technical Description	14
3	Log Collection	18
3.1	Rationale	18
3.2	Pre-study	18
3.2.1	Requirements	19
3.2.2	Confidential Logging and Cryptography.....	19
3.2.2.1	Symmetric Key Encryption	20
3.2.2.2	Public Key Encryption.....	23
3.2.3	Tools Currently Used	24
3.2.3.1	Tool A.....	24
3.2.3.2	Other Tools.....	25
3.2.4	Possible Means of Software Reuse	25
3.2.4.1	Proprietary Software	26
3.2.4.2	Software Components.....	26
3.2.4.3	Libraries.....	27
3.2.5	Conclusions.....	30
3.3	Solution	31
3.3.1	System	31
3.3.2	Logger.....	32
3.3.2.1	Functionality.....	33
3.3.2.2	Design and Implementation	33
3.3.3	Decryptor.....	38
3.3.3.1	Functionality.....	38
3.3.3.2	Design and Implementation.....	38

3.3.4	KeyGenerator	40
3.3.4.1	Functionality.....	40
3.3.4.2	Design and Implementation	40
4	Log Analysis	42
4.1	Rationale	42
4.2	Pre-study	42
4.2.1	Requirements	42
4.2.2	Tools Currently Used	43
4.2.3	Usability.....	44
4.2.4	Performance.....	45
4.2.5	3GPP Message Translation	47
4.2.6	Conclusions.....	48
4.3	Solution	48
4.3.1	Functionality.....	48
4.3.2	Design and Implementation	49
4.3.2.1	Usability.....	50
4.3.2.2	Performance.....	52
4.3.2.3	3GPP Message Translation	52
5	Results	58
5.1	Log Collection	58
5.2	Log Analysis	59
6	Discussion	62
6.1	Methodology	62
6.2	Log Collection	62
6.3	Log Analysis	64
7	Acronyms and Abbreviations	66
8	References	68
Appendix A. EMBLA Software Requirements Specification		
Appendix B. EMBLA Administrator Manual		
Appendix C. EMBLA Logger Customer Manual		
Appendix D. EMBLA Logger Test Specification		

1 Introduction

This introduction explains the goals for this master thesis work and gives some background information motivating the thesis. The methodology used for reaching these goals and some limitations and delimitations are also stated.

1.1 Background

The main reason for this thesis proposal by the Mobile Broadband Modules department [1] at the telecommunications company Ericsson [2] is the need to make their field testing and customer support more efficient, which is especially needed because of recent growth. More information about the department and their products and services will be given in section 2. Two important activities have been identified as candidates for improvement.

Firstly, the collection of mobile broadband logs needed to troubleshoot problems could be improved. The data contained in the logs is seen as confidential and should therefore not be revealed to customers, which today hinders the log collection. Furthermore, too many tools for log collection are being used.

Secondly, the analysis of these collected logs could be improved. Also here, too many tools are being used, many also suffering from poor performance and usability.

1.2 Goal

The main goal of this work is to find solutions to the problems of how to log confidential information with sufficient security for a specific context and how to analyze log data with high enough performance and usability for the same context.

Solutions to these problems are provided by introducing two new systems at Ericsson Mobile Broadband Modules solving the problems in an industrial context. One system allows the department's customers to securely log different kinds of confidential information and the other one facilitates the analysis of the collected logs at the department. These introductions can be realized via both existing commercial systems or via the development of new systems. By solving these problems the end goal is to make the internal work at Ericsson Mobile Broadband Modules more efficient, therefore providing means to better meet customer demands.

1.3 Methodology

This thesis report, and the actual thesis work itself, is divided into two distinct parts corresponding to the two system introductions, one being Log Collection and the other one being Log Analysis. The software development in the phases was influenced by a traditional waterfall model [3], together with some principles and values from agile methods, such as Extreme Programming [3]. Some pair programming was done and communication was valued higher than documentation, allowing the development to better cope with changes due to ambiguity in elicited requirements.

The pre-studies started with an initial requirements elicitation. The elicitations were primarily accomplished by having meetings and workshops with employees and by observing the current workflow. Some informal discussions with employees also took place. Having elicited and specified the requirements, literature studies were performed in order to find possible theoretical solutions to solve the problems. Having decided on theoretical solutions, the

means of introducing a system having those theoretical features were studied. Finally, the systems were designed, implemented and lastly evaluated.

Both systems were designed to make maintenance, and possible further development, as efficient as possible after delivery. An administrator manual (See Appendix B) and a customer user manual (See Appendix C) were also written for the system intended to be used by customers to ease the hand-over of it. Releasing beta versions¹ continuously during development was done to receive quick user feedback in order to better meet the expectations and needs of the users. During system testing, demo versions² were distributed to get the final feedback required for the final adjustments and to evaluate the systems. After that, focus was put on robustness testing and bug fixing. Test specifications were also created and then used by Ericsson employees to test the correctness of the systems on the different operating systems supported (See Appendix D for an example of a test specification).

1.4 Limitations and Delimitations

When searching for solutions to help reach the goals set out for this thesis, many options were discarded because of context specific aspects. A complete study of the fields log collection and log analysis was therefore not performed. Despite this, some general conclusions and guidelines are given.

Regarding the performance optimization in the Log Analysis section, optimizations done by the compiler were not studied. Instead, performance tests were done to make sure that the theoretically best solutions found also were preferable in practice.

1.5 Overview

This report commences by giving some information about the context of the thesis work. The two following sections describe the two phases of the thesis work. Finally, this report ends with the results and some conclusions. A list of the used acronyms and abbreviations is present in the end as well.

1.6 Related Work

The theoretical aspects of the first part of the report, log collection, have their roots in data logging³ and applied cryptography. The theoretical parts of the latter half of the report have their roots in data analysis⁴ and data visualization⁵, or more specifically log analysis and log visualization.

The field of applied cryptography is widely covered in literature. Schneier [4] gives a comprehensive overview to the field of cryptography. Partida and Andina [5] study the suitability and performance of Java for implementing security, but much has happened to Java since it was written, 1999. Shen and Zheng [6] give an overview of the Java Cryptography

¹ Versions with limited functionality

² Versions with full functionality, but still not the final version

³ *"Data logging is the practice of recording sequential data, often chronologically."* [61]

⁴ *"Data analysis is the process of looking at and summarizing data with the intent to extract useful information and develop conclusions."* [63]

⁵ *"Data visualization is the study of the visual representation of data, defined as information which has been abstracted in some schematic form, including attributes or variables for the units of information."* [62]

Architecture. There is a wide body of literature documenting and describing the security of programming languages.

Data analysis, logging, and visualization are also fields about which much has been written. The problem is that, though some general conclusions sometimes are made, most of the literature found is very domain specific. By looking at some definitions of these fields (See footnotes 3, 4 and 5), this is not a surprising fact, since all data is different and each context for analyzing and visualizing it is unique. Unfortunately, no paper regarding this thesis' specific domain was found.

In the field of log analysis, Andrews and Zhang [7] show how log analysis can help checking test results to a broad range of tests. Vaarandi [8] presents an algorithm for finding patterns in event logs. Razavi and Kontogiannis [9] present a framework for doing a similar task.

Regarding data logging, specifically confidentially, not much information has been found except for Ohtaki's paper [10] on how to create encrypted logs which provide a way to disclose only some parts of the log.

In the field of data visualization, Ahlberg and Shneiderman [11] show how data filtering can be enhanced by using progressive user queries that enable more detailed analysis. Roberts [12] describes the advantages of using multiple views in data visualization in order to facilitate data analysis. A more strongly-related paper on data visualization is Takada and Koike's paper on MieLog [13]. MieLog is a generic tool for visualizing log contents by using information visualization and statistical analysis. Unfortunately, the tool is not freely available because of supposed plans to commercialize it.

Literature regarding some technical details and aspects of the two systems exist and have been studied and used for this thesis, but no paper that adequately addresses this thesis' goal has been found, even though similar systems likely exist at other companies.

2 Context

To get a better overview of the environment for this thesis, the department and their activities will firstly be presented, followed by a technical description of the main product.

2.1 Mobile Broadband Modules

Ericsson Mobile Broadband Modules (MBM) offers a module portfolio and various services to portable PCs manufacturers in order for them to offer mobile broadband to their customers. The developed modules support most mobile standards (HSPA, WCDMA, EDGE, GSM) and can therefore also be integrated into other devices suitable for mobile broadband. [1]

The module portfolio consists of the mobile broadband module, a customizable dash board, and a software development kit. The module is the piece of hardware integrated into devices offering mobile broadband connectivity (See Figure 1 for an example of a module); the customizable dash board is an application which allows customization of the connection manager, an application for monitoring the data connection; the software development kit is a kit that enables customers to develop their own connection managers. Mobile Broadband Modules design, develop and market the whole portfolio mentioned. [1]

The team requesting this thesis work is the Firmware⁶ Support team which is part of the integration and verification unit of the department. Their primary task is to work with customers, i.e. telecommunication operators and laptop vendors, to get the mobile broadband modules approved by them and to solve their issues. To be able to troubleshoot modules' firmware issues, logs must be collected and sent to MBM Firmware Support. The log collection is done by a set of different tools and then, most often, sent to MBM via the File Transfer Protocol (FTP). These logs are then analyzed, also here using a number of different tools, so that the issues can be pin-pointed and appropriate actions taken. Some of the different tools used to collect and analyze logs will be presented later in this report.

2.2 Technical Description

The module out on market at the time of publication, F3507g (See Figure 1), is a Peripheral Component Interconnect (PCI) Express Mini Card⁷ module which support the mobile standards HSPA, UMTS, EDGE, GPRS, SMS and UMTS/CSD, and both tri-band⁸ and quad-band⁹, in combination with an ordinary Subscriber Identity Module card¹⁰ (SIM card). With the, at the moment, fastest data service, HSPA, enabled, this module supports downlink speeds up to 7.2Mbps and uplink speeds of up to 2.0Mbps (See Table 1). [14]

The module also has a Global Positioning System (GPS) receiver which can be used in combination with positioning applications. The module does not support voice calls, because of its always-on property, but Voice Over IP¹¹ (VoIP) can instead be used. [14]

⁶ Read-only programs that internally control various electronic devices

⁷ A standard computer expansion card format

⁸ Tri-band describes mobile platforms supporting three operating frequencies

⁹ Quad-band describes mobile platforms supporting four operating frequencies

¹⁰ A removable card used in mobile telephony devices to store data needed for mobile communication

¹¹ General term for a family of transmission technologies for voice communications over IP networks



Figure 1. The F3507g module [15] (Cropped with permission)

Module drivers are based on regular Universal Serial Bus¹² (USB) functionality and the module is, in Microsoft Windows operating systems, presented in the device manager as a multifunctional device offering modem, networking, USB and COM-port¹³ interfaces. The module is also open for application developers via a high level application programming interface (API) and a standard AT command¹⁴ interface. [14]

Newer modules, such as the F3607gw [16], with different functionality and supporting newer telecommunication protocols are under development and have been publicly announced.

¹² Common physical computer interface for serial communication

¹³ A common physical serial communication port

¹⁴ Command set developed by Hayes Microcomputer Products, most often used for communication with modems

Mobile Broadband Module F3507g	
Form Factor:	PCI Express Mini Card
Modes:	HSPA/WCDMA/EDGE/GPRS
Bands:	WCDMA 2100/1900/850MHz GPRS/EDGE 850/900/1800/1900MHz
Speeds:	HSPA Data speed D/L 7.2Mbps U/L 2.0Mbps
GPS:	Supports stand alone and assisted GPS
Antenna:	WCDMA receiver diversity
Connectivity:	Advanced receiver – GRAKE2 USB 2.0
Drivers:	MS Windows XP, Vista, MacOS, Linux
Software:	Customizable connection manager (dashboard)
Certifications:	FCC CE R&TTE PTCRB GCF Operator and infrastructure IOT Microsoft certified driver
Environmental:	RoHS Compliant Halogen free

Table 1. F3507g technical facts [17]

3 Log Collection

This section contains the first part of this master thesis work, the introduction of a system which allows customers to securely log information and provides the modularity needed to gather the different kinds of data needed.

Firstly, the rationale behind the system is described. Secondly, the pre-study performed to acquire the knowledge and insight needed to solve the problem is presented. Finally, the actual solution is presented.

3.1 Rationale

As already mentioned, the collection of mobile broadband logs is seen as an activity that is possible to improve by incorporating confidentiality to the collection of logs and also by reducing the number of tools used for the collection.

When troubleshooting customers' problems, live network testing is often required to obtain the information needed. This means that the collection of that information can only be done while in the actual network. The problem is that the tools used today are impossible to distribute to the customers themselves because of confidentiality issues. The tools would reveal information seen as confidential by Ericsson and therefore require Non-Disclosure Agreements¹⁵ (NDA) to be written. Today, to be able to troubleshoot, they therefore have to send employees to the actual different physical locations to gather the information, in the form of mobile broadband logs. This is seen as harmful to the environment, expensive, and not optimal time-wise for neither the customers themselves nor the support at MBM.

Another reason for this thesis proposal is that the logging of different data (such as driver activity, firmware activity etc.) is today done by different tools which is not seen as an optimal way of working. A modular system could improve the efficiency and also help the process of introducing new employees since knowledge about fewer tools would be needed. Having multiple tools for logging different data sources is also not seen as an optimal way of working since customers might not have the knowledge of what data source to log.

Because of these issues there is a big need for a system that can be used by customers to collect log data without revealing any confidential information. Such system would also be required to log different data sources simultaneously to make logging convenient for customers.

3.2 Pre-study

With the rationale in mind, a pre-study was needed to find a solution to the described issues. The first step was to elicit and specify the requirements possible solutions would need to meet. The next step was to study the different areas needed to find the best solution. Firstly, the most important requirement, that the log data must be kept confidential and inaccessible for the user, was investigated. Secondly, the currently used tools were studied, and then what possible alternative solutions exist which could help solve the problem. This section is concluded by describing and motivating the chosen solution.

¹⁵ A legal confidentiality contract, between at least two parties, to restrict access to materials or knowledge that the parties wish to share amongst each other

3.2.1 Requirements

With the rationale in mind, the fundamental functional requirements of the system were elicited and specified (See Appendix A).

The operating environment where the system is to be used put some requirements on it. Microsoft Windows XP and Microsoft Windows Vista are the operating systems installed on the computers. Also, compatibility with upcoming operating system Microsoft Windows 7 is desirable.

The absolutely most important requirement is that the logs should not be readable by anyone other than MBM employees, even in real-time. Therefore, the log data must be made hidden for the customers, and visible for MBM employees.

The logs of most interest to the Firmware Support team are, unsurprisingly, firmware logs. Different data providers¹⁶ for each module must be possible to log, even simultaneously. The modules have two loggable data providers, one being the access-side, the part of the firmware handling the network signaling, and the other one being the application side, which handles everything else, such as SIM Application Toolkit¹⁷ and GPS amongst others. These data providers also support so called interactive logging, which means that they can receive commands that affect what is being output by them. The startup of the module firmware is also important to be able to log. Other data sources to log might also be of interest to reduce the number of tools used, but none is of equally high importance as the firmware.

As earlier described, the modules can be accessed via for example COM-port interfaces. What COM-port to use, when communicating with the modules, must also be selectable.

Last, but not least, the system must be simple to use since it is to be used by customers, usually not having technical knowledge about the module. It must be simple to select what to log and when to do so. Some feedback regarding the logging is also important since the actual log data is hidden. Because of these requirements and the amount of functionality needed, a graphical user interface (GUI) is required. The program should also be robust and not crash, and give appropriate feedback if the connection to a module is lost etc.

3.2.2 Confidential Logging and Cryptography

To prevent customers using the logging tool from accessing the log data, the data had to be hidden, which can be done in a number of ways.

The perhaps first, and somewhat naive, approach is to simply write to a file hidden somewhere on the file system. This approach was instantly discarded because of poor security. Finding the actual file is enough to get access of the entire logs.

Another approach is to store the collected data in the computer's volatile memory. A problem with this approach is that the memory is readable by anyone. This can be done on many systems by for example putting the computer into hibernation and reading the memory dump created by the operating system on non-volatile memory in such situations. Software for doing this is easily available, an example being Sandman [18]. An adjustment that would solve this

¹⁶ A module data channel, either outputting or receiving data

¹⁷ The application handling the SIM card and its possible applications

issue would be to use some kind of cryptographic algorithm and encrypt the data in real-time. The problem with this approach is that the amount of log data collected in this environment can reach hundreds of megabytes, which is simply too much to store in volatile memory.

A possible viable solution is instead to use a well-established cryptographic algorithm and encrypt the data in real-time to a file instead. Using this method, a limited amount of readable log data is kept in memory for a short time, and if kept reasonably low, this is seen as acceptable. To be able to retrieve log chunks from the memory, one would have to dump huge amounts of memory with very short time intervals, making the retrieval of logs infeasible. Log data from the module is being outputted at high speeds, so the encryption speed is important in order to not have too much log data stored in buffers, both because of performance and security reasons. Different principles in cryptography and some of the most well-known algorithms will now therefore be described to find a suitable solution.

Four main characteristics describing different situations common in cryptography are stated by Trappe and Washington [19], using the well known cryptography placeholder names Alice, Bob and Eve:

- Confidentiality: Eve should not be able to read Alice's message to Bob.
- Data integrity: Bob wants to be sure that Alice's message has not been altered.
- Authentication: Bob wants to be sure that only Alice could have sent the message he received.
- Non-repudiation: Alice cannot claim she did not send the message.

By analyzing the context and the requirements one has, the needed characteristics can be determined. The characteristics then influence what cryptographic principles that are appropriate to use. In this specific case, the only needed characteristic is confidentiality. There is nothing to be gained for a person changing logs, which rules out data integrity, or claiming a log does not originate from him/her, which rules out non-repudiation. Authentication is given by the FTP accounts used by customers to upload logs.

Having this knowledge, it is now possible to study the well-established suiting cryptographic algorithms available, all having their respective advantages and disadvantages. Not only is a strong algorithm required, but also an appropriate key size, since the security is a function of both [4]. Too long keys are not desirable either since they are likely to affect the computation speed. The forthcoming sections describe and evaluate some of these well-established cryptographic algorithms with regard to the requirements for this system.

3.2.2.1 Symmetric Key Encryption

Symmetric key encryption algorithms are algorithms where the same key is used for encryption as decryption (See Figure 2), or there is a simple mathematical relation between them. One advantage of symmetric key algorithms is that they generally are computationally faster than asymmetric ones. They can also be used to encrypt parts (blocks) of a message separately [19]. These characteristics are positive for this context since the speed of the outputted data to encrypt will be high and because the block-by-block encryption allows real-time encryption.

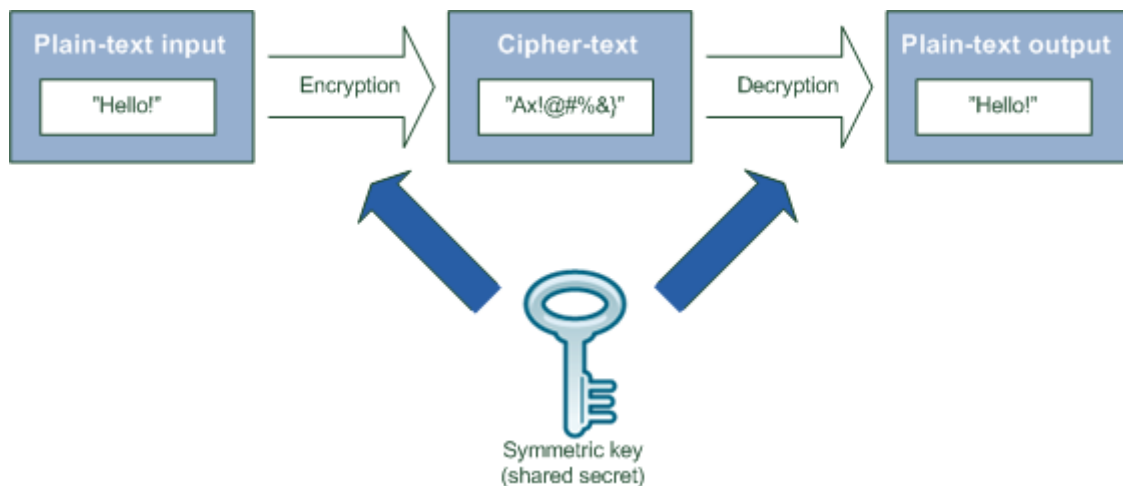


Figure 2. Symmetric key encryption and decryption

There are however also difficulties with a symmetric key algorithm. Both parties must agree on a secret key, which could be difficult to arrange in a secure way [19]. Another major problem is that a symmetric key, or information on how to easily compute it, must be stored on the users' systems to enable encryption and decryption. Storing the key on the customers' systems would allow them to also decrypt and read the logs, which would be devastating.

The two types of algorithms within symmetric key cryptography, stream ciphers and block ciphers [19], will now be described.

3.2.2.1.1 Stream Ciphers

Stream ciphers are algorithms where messages are consumed in small pieces (bits or characters) and the output is pieces of corresponding size. Stream ciphers are generally fast and with low memory usage since only small pieces need to be kept in the memory at the same time. The speed of the most widely used stream cipher Rivest Cipher 4 (RC4) can for example be estimated to around ten times faster than the well-established block cipher Data Encryption Standard (DES). The speed comes at a cost though. Stream ciphers are not as secure as many other common cryptographic algorithms, such as block ciphers and public key systems, which will be described later. The reason for this is that encrypted messages are vulnerable to frequency attacks since each bit/character is encrypted independently. This vulnerability is seen as very negative for this setting. [19]

3.2.2.1.2 Block Ciphers

Block ciphers are algorithms that encrypt a block of characters, i.e. a number of bits, simultaneously. One change in the original message will lead to a lot of potential changes in the encrypted message, depending on the mode of operation used. When encrypting a text larger than the block size, different modes of operation have different effect on how the different blocks are encrypted.

The simplest mode of operation is Electronic Codebook mode (ECB) where each block is encrypted separately using the key. This results in identical blocks being encrypted to identical ciphertexts¹⁸. This is a weakness since the ciphertexts give information about the contents of a message to an adversary. Another mode of operation is Cipher-Block Chaining (CBC), which is

¹⁸ Encrypted text

the most commonly used one. In CBC an XOR¹⁹ operation is performed between the plaintext²⁰ block to encrypt and the previous encrypted block. After the XOR operation the block is encrypted using the key. This results in a chaining effect where a change in one plaintext block results in changes in all following encrypted blocks. It also prevents ECB's issue with identical plaintext blocks resulting in identical encrypted blocks. For this to work the plaintext message must be padded to make the length of it a multiple of the block size. An initialization vector (IV) is used in CBC for performing the XOR operation on the first plaintext block. Using the same IV for encryption of identical messages results in identical ciphertexts and it is therefore common to use randomized IVs. Using randomized IVs also requires the IV to be transmitted to the recipient. One weakness with CBC compared to ECB is that encryption must be sequential due to the chaining of blocks. For this setting, encryption will only be required sequentially making CBC still suitable. CBC would also be suitable due to its security properties. There are also other modes of operation available designed to for example handle errors in the ciphertext by avoiding error propagation. This feature is not required for this setting since data integrity is not an issue, as described earlier. [19]

Using block ciphers with large enough block sizes is secure against frequency attacks, unlike stream ciphers. Block ciphers are in general not as fast as stream ciphers, but still a lot faster than public key systems [19]. The ability to encrypt blocks separately and the performance attributes make block ciphers interesting for the current setting. Two well-established block ciphers will therefore now be described.

Data Encryption Standard (DES)

DES was the first encryption standard endorsed by the U.S. government [20]. Its development was initiated in 1973 and it was in 1977 made the official data encryption standard by the National Bureau of Standards²¹ (NBS). DES was at the time of its creation considered secure, but with modern computational power brute force attacks²² has become feasible and DES is therefore not of interest for this system. The need for other standards with support for larger key sizes triggered the development of algorithms such as Rijndael [19], which is described later in this section.

Advanced Encryption Standard (AES)

Advanced Encryption Standard is a cryptography standard issued by the U.S department of commerce agency National Institute of Technology (NIST). It was created to replace its predecessor DES since it had become feasible to break with modern computational resources. The algorithm chosen for AES was an algorithm called Rijndael, designed by, and named after, Vincent Rijmen and Joan Daemen. The Rijndael algorithm was chosen as standard not only because of its security, but also because of its high performance speed, low memory requirement and agility in terms of keys. Three of the key sizes included in the algorithm were included in AES (128 bits, 192 bits and 256 bits). These three key sizes are considered adequate for U.S federal government applications [21]. AES handles blocks of 128 bits each [22]. AES is expected to eventually replace DES as the most commonly used block cipher because of its

¹⁹ Logical operation for exclusive disjunction

²⁰ Unencrypted text

²¹ A measurements standards laboratory today known as The National Institute of Standards and Technology (NIST)

²² A method of breaking a cryptographic algorithm by trying a large number of possibilities, often keys

strong features and improved security. AES is considered theoretically unbreakable [23], and is also expected to remain secure for a few decades because of its large key sizes. Because of these strong features AES is of much interest for the current setting.

3.2.2.2 Public Key Encryption

Public key encryption algorithms are algorithms where a user's public key is used for encryption and the same user's private key for decryption. All users have one public and one private key. A user's public key is accessible for other users and is used to encrypt messages to him/her. His/her private key, that is not accessible for anyone else, is then used to decrypt the message (See Figure 3). Public key algorithms rely on having a complex mathematical relation between the public and the private key making it computationally unfeasible to calculate the private from the public.

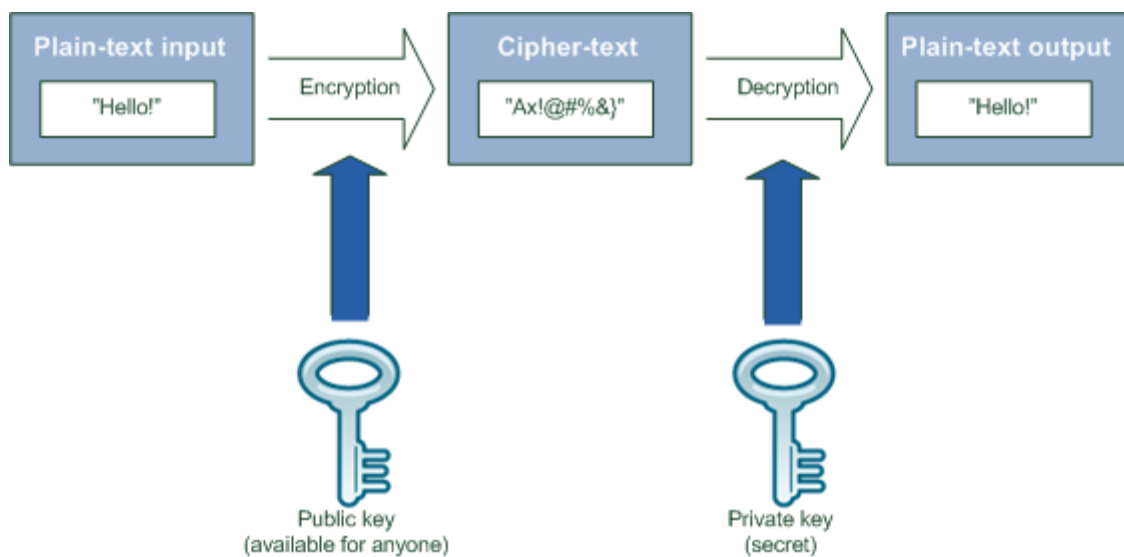


Figure 3. Public key encryption and decryption

Public key systems are considered very powerful, but slower than for example symmetric key algorithms because of their mathematical characteristics. The one most widely used today is RSA (named from Rivest, Shamir and Adleman), which is described in the following section. [19]

Public key cryptography is of interest for the current setting because of the use of different keys for encryption and decryption. This would allow encryption of data in the customers' computers without making decryption possible, because of the need for the private key to decrypt. Using this pattern would only require distributing a public key to customers and having the corresponding private key accessible only at MBM. Customers would not need a private key because of the messages only being sent from customers to MBM and not the other way. Private keys are also used for signing messages, but since that is not a requirement having a public key is enough for customers. Using this pattern would also be cryptographically secure, the only security risk being if MBM's private key is leaked or lost. The negative side of this alternative is that small parts of the logs would have to be encrypted independently to avoid having an entire log in the runtime memory at once. The amount of encryptions required makes the speed demands of the encryption high to be able to log in real-time and thus a public key algorithm unsuitable.

3.2.2.2.1 RSA

RSA is an implementation of a public key system which uses the fact that factorization of large integers into their primes is computationally time-consuming to ensure the complexity of the relation between public and private keys [19]. RSA was designed in 1977 by Rivest, Shamir and Adleman and is still, at time of publication, considered unbreakable when used properly. Since there is no element of randomness in RSA and due to a risk involved with encrypting too short messages, padding is commonly used. With no randomness the encryption would be vulnerable to chosen plaintext attacks, which is trying encrypting different plaintexts and comparing them to the ciphertext to break. If the ciphertexts are equal, the adversary has found the original plaintext. One padding scheme commonly used together with RSA designed to solve these issues is Optimal Asymmetric Encryption Padding (OAEP) [19]. OAEP includes an element of randomness in the padding and makes sure the plaintext to encrypt is of appropriate length [24].

RSA can be used with different key sizes, but to provide sufficient long-term security it is recommended to use at least 2048 bit keys [25]. Like other public key systems RSA is not as fast as symmetric key algorithms and is not suitable for large quantities of data. A common use of RSA is instead to use it for transmission of keys for use in symmetric algorithms (like DES or AES) [19].

3.2.3 Tools Currently Used

To be able to find the best possible solution to this problem, some of the existing tools for collecting logs were studied. The main firmware log collection tool, here called Tool A, and some other tools will now be presented.

3.2.3.1 Tool A

Tool A was created to test mobile equipment based on Ericsson's mobile platforms [26] and is used by MBM to log information from the modules' firmware. Tool A consists of both a server application and a logger application. The server is registered as an ActiveX²³ component and communicates with the module. The logger then communicates with the server to get logs.

The logging is realized by connecting to the modules via COM ports, either standard PC and laptop COM ports, PCMCIA card²⁴ COM ports or 8-port Digiboard²⁵ COM ports. Windows' virtual COM ports can also be used if Tool A is used over Universal Serial Bus²⁶ (USB), IrDA²⁷ or Bluetooth²⁸. [27]

Tool A runs on, amongst others, on the operating systems Microsoft Windows XP and Microsoft Windows Vista, both 32 and 64-bit, which are the operating systems required to be supported [26].

²³ A framework by Microsoft for defining reusable software components

²⁴ Form factor of peripheral computer interfaces by the international standards body Personal Computer Memory Card International Association

²⁵ Boards with multiple asynchronous serial interfaces developed by the company Digi International

²⁶ Serial bus standard to connect devices to a host computer

²⁷ Communication protocol specification, by The Infrared Data Association, for short-range exchange of data over infrared light

²⁸ A wireless protocol for exchanging data over short distances over radio frequencies

Unfortunately, Tool A is a proprietary piece of software and the logging is done to an ordinary text file and directly shown in the GUI, which is the main reason to why this tool cannot be distributed to customers by MBM. MBM has some permission to use the server, which is of interest for this system. The Tool A server is interfaced through a C++ API.

3.2.3.2 Other Tools

There are other loggable data sources that are of interest for MBM when troubleshooting the module and its software. When in need of driver logs and logs for one of the mobile broadband connection managers, a free tool called DebugView [28] is used. DebugView is developed by Windows Sysinternals[29] and logs both kernel-mode (DbgPrint²⁹), which the modules' drivers use, and WinAPI³⁰ debug output (OutputDebugString³¹), which Wireless Manager uses.

Another data source sometimes of interest is the network traffic. Packets from the protocols for network communication, Transmission Control Protocol (TCP) and User Datagram Protocol (UDP), are then logged with the free software tool Wireshark [30].

As with Tool A, both these tools display the log output in their GUI in real-time and both write this information to plain-text non-encrypted text files. This does of course not comply with the confidentiality requirements. Since DebugView may be freely installed and used, one might consider the option of modifying or further developing it to suit MBM's requirements. Unfortunately, DebugView's End User License Agreement (EULA) clearly states that one may not *"work around any technical limitations in the binary versions of the software"*, nor *"reverse engineer, decompile or disassemble the binary versions of the software (...)"*.

3.2.4 Possible Means of Software Reuse

This section presents different possible means of software reuse to help meet the requirements described (See section 3.2.1). To be able to most efficiently introduce the new system, the possibility to use proprietary software, software components, and libraries was investigated. Proprietary software is here considered to be software being legal property of another party with the terms of use being defined by contracts or licenses. Software components are considered to be independent pieces of software that could make up parts of the new system. Libraries are collections of classes and subroutines created to facilitate implementation of other applications.

By evaluating the possibility of using existing proprietary software, components and libraries, i.e. software reuse, the most efficient way of introducing the system could be found. Efficiency here refers not only to time needed for development, but also to aspects such as security and dependability [3]. For software reuse to be efficient it is important that it must be both easier to reuse the artifacts and quicker to find them than it is to develop software from scratch [31]. Useful, complex components tend to be complex to reuse, and much time is needed to understand the abstractions, which might defeat the purpose of reuse [31].

²⁹ Routine in "Windows Driver Kit: Driver Development Tools" that sends messages to the kernel debugger [60]

³⁰ The core set of APIs available in Microsoft's Windows operating systems [59]

³¹ Function in WinAPI that sends strings to the debugger for display

Using already well-documented and well-tested software is more secure than developing from scratch, since implementing cryptographic algorithms without errors is very hard due to their complexity. It is also very difficult to prove the correctness of an implementation [32]. Using existing cryptographic implementations following well-established standards is therefore preferred. Reusing software can also have an effect on the system architecture, which needs to be taken into account.

3.2.4.1 *Proprietary Software*

The log collection system has some unique requirements, such as the communication with the broadband module, the confidentiality aspect, and some usability requirements. This makes it unrealistic to find existing proprietary applications that meet the requirements.

Even if found, there are some negative aspects of adopting proprietary applications. Because of lack of ownership, problems regarding support and maintenance might occur in the future, and the applications might also require unwanted license fees to be paid.

Furthermore, it is sometimes seen as questionable if such an application can be trusted because of lack of access to the source code [33], meaning that the security of the system can not be verified, except by reverse-engineering. *“With proprietary software, it is ultimately a matter of trust [...]”* [34] and *“For proprietary software [...] history tends to show that bad cryptography is much more frequent than good cryptography [...]”* [34]. This is particularly of interest because of the confidentiality requirement for this application, meaning that proprietary software with poor quality could lead to lacking security. Finding a third-party application is therefore excluded.

3.2.4.2 *Software Components*

Another strategy when introducing the new system would be to make use of existing software components. When looking for components, two main parts of the system were identified as candidates: The cryptographic part, and the log collection part. Finding a component for the GUI was discarded because of the nature of GUIs. GUIs most often directly correspond to the functionality of applications, and having unique functionality, such as the communication with the broadband module and the simultaneous logging of specific desired data sources, require a GUI to be implemented from scratch.

Regarding the cryptographic part, there are many implementations of cryptographic algorithms available that could be used as components in the system [35]. A benefit of using existing cryptographic components is that a component that has been thoroughly tested and in use for a while in different systems can be more secure than a new implementation of an algorithm. This is because of the risk that an algorithm’s complexity causes mistakes to be made that lead to poor security, as mentioned earlier.

The problem with some of the existing cryptographic implementations is that they are black-box components, meaning that no information about their internal workings is accessible, which can make them insecure to use since the correctness of their implementation cannot be verified, as described earlier.

There are also white-box components, meaning that information about the internal workings is available, and source code components available. Having this knowledge, correctness

verification and thorough testing could be done to guarantee that the components comply with certain requirements, but with a high degree of cryptographic expertise needed. One might think that white-box components are well-studied, since the source code is open, and therefore secure, but the openness does not guarantee any quality attributes by itself, since it does not imply that experts have verified the source code [34]. There is one famous example of a well-used open source component with a major security flaw, namely a part of the GNU Privacy Guard³² (GnuPG or GPG). This component is included in most GNU/Linux distributions, but despite this, the flaw was present for almost four years [34].

There are also negative sides to using white-box components. One negative side of using such components could be the overhead in integration. Unless the features provided and the properties of the component are perfectly suitable for the system, modifications to either the component or the rest of the system would be required leading to more work overhead. Such modifications also risk negatively affecting the architecture, the performance, and the security of the system. This, because the sum of separate components' properties is not the same as the properties of a system using them combined. Therefore, one needs to be careful when integrating, especially security-critical components, to make sure that the quality attributes of the components are not affected and that no unexpected emergent properties appear because of poor integration [36].

Regarding the log collection, no tools were found which log several of the different data sources of interest. Most tools found were also GUI-based and non-open source, which hinders them from being used as components in this system. Even if possible, the workload for integration and adaptation was seen as, in this case, heavier than to implement the collection from scratch. Also, the likely negative impact on the system architecture was not desired.

3.2.4.3 Libraries

As a combination of both well-tested, high-quality implementations and flexibility, libraries can be good to use. Libraries generally provide higher flexibility than larger components due to the contents of libraries being smaller pieces giving developers more freedom when assembling.

Regarding what programming language to scan for appropriate libraries, the in-house developed tools at MBM are mostly implemented in C# or Java. Since the new system needs to be maintained within the organization, using one of these two languages is preferable. Therefore, these two programming languages' frameworks' match to the problem domain is now presented. Other libraries possible to use with these languages exist, but the native implementations are preferred because they are heavily used and therefore thoroughly tested. Firstly, the Java Standard Edition (Java SE) framework [37] and then the .NET framework [38], used by C#, are presented.

3.2.4.3.1 Java

Java is a well-established object-oriented programming language providing libraries that could be used for the desired functionality. Java's compiler compiles the source code to bytecode³³ that can then be executed by the Java Virtual Machine (JVM). When compiled the code is verified to for example make sure that only code conforming to the Java language specification

³² A free implementation of the OpenPGP standard, a common standard for securing e-mail

³³ Machine-independent code generated by the Java compiler and executed by JVM

and not violating memory management or using illegal typecasts is run. The JVM also provides a level of security by preventing malicious code from corrupting the runtime environment [39] [40]. The JVM is available for different operating systems making Java applications executable on Microsoft Windows, Solaris OS, Linux, and Mac OS [41].

Besides offering security through the platform itself, the Java platform provides frameworks supporting many of the existing common cryptographic algorithms, mechanisms, and protocols to developers [40]. These frameworks are found in the platform's Java Cryptography Architecture (JCA) and Java Cryptography Extension (JCE), which is part of JCA [42]. The frameworks contain implementations of common algorithms, such as for example RSA, AES and DES [39], which were evaluated earlier (See section 3.2.2). There are also pseudo random number generators provided for use in for example key generation. Support for different cipher modes and padding, such as OAEP, is also provided [42].

The APIs for cryptography are designed to allow developers to use multiple interoperable implementations of algorithms, thereby allowing them to use both native implemented algorithms and services or to easily plug-in custom services [40]. Apart from the cryptographic features provided, the platform also contains APIs for other security areas such as authentication, access control, and secure communication.

To verify the security of the cryptographic features provided, the library documentation was studied. The implementations of the algorithms found most interesting, AES and RSA, follow their specifications. One feature interesting to evaluate is the pseudo random number generation used when generating keys. Generation of keys would be of interest in the system developed, but in order to use it, the properties of it have to be analyzed. The functions for generation of seemingly random keys for encryption/decryption in Java use interoperable implementations of sources of randomness. There is also a default random number generator provided (See the [SecureRandom](#) class [43]) that complies with the statistical random number generation tests specified in the Federal Information Processing Standard (FIPS) 140-2, "Security Requirements for Cryptographic Modules" [44]. FIPS are public standards developed by the United States Federal government.

Since Java follows a standard for cryptographically secure random number generation and due to how well-established and thoroughly tested the implementations of the algorithms are, the usage of Java's cryptographic libraries is considered secure.

Another positive aspect of using Java for this development is that Java's class library provides classes facilitating the development of graphical user interfaces through its Java Foundation Classes (JFC), containing amongst others the Swing library. Java is also a language commonly used at the department, which would ease maintenance and possible further development of the system.

3.2.4.3.2 Microsoft .NET

The Microsoft .NET framework is a Microsoft Windows component providing an object-oriented programming environment and a code-execution environment for developing Microsoft Windows applications. The two main parts of the framework are the class library and the common language runtime (CLR). The CLR runs the code and provides services making development easier and more secure, such as for example memory management and thread

management. Source code that targets this runtime is known as managed code. It is also possible to write unmanaged code bypassing this runtime environment if desired. The class library is a set of classes providing common functionality supporting development. [45]

One part of the class library of particular interest for this system is the cryptographic features. Microsoft .NET provides features for, amongst others, symmetric key encryption, public-key encryption, digital signatures, and random number generation. This includes implementations of RSA, AES and algorithms for key generation, all of interest for this system [46]. To verify the security of these implementations, documentation of their internal workings were studied.

The algorithms for AES and RSA are implemented according to their specifications, with the possibility for developers to set variables such as key size. There are also implementations of different cipher modes available that could be used for the block cipher AES, such as CBC and ECB (See CipherMode Enumeration [47]), described in section 3.2.2.1.2. For using RSA there is implemented support for OAEP padding. The key generation feature provided generates seemingly random keys of a chosen size for a specific cryptographic algorithm. The randomness in the key generation uses features from the cryptographic API (CAPI) included in Microsoft Windows. CAPI stores a user specific seed allowing different applications to contribute to the randomness. When an application calls the random function it passes the information it might have, such as keyboard timing input for example. This application-specific information is then combined with both the current seed and system data, such as the system counter and free disk clusters. This seed is then used by the pseudorandom number generator (PRNG). The PRNG in Microsoft Windows Vista service pack 1 and later versions use an implementation of the AES counter-mode based PRNG specified by National Institute of Standards and Technology (NIST), an agency of the United States Department of Commerce, in special publication 800-90 [48]. In Microsoft Windows Vista, without service pack 1, and Microsoft Windows XP a PRNG specified by Federal Information Processing Standard (FIPS) 186-2 is used [49].

Since Microsoft Windows operating systems follow government standards for generation of pseudo random cryptographically secure numbers, the key generation in the .NET framework can be considered secure. Due to how well-established and thoroughly tested the framework is, the implementations of AES and RSA are here also assumed to be correct and secure.

Since the system has high demands on usability, the possibility to quickly develop a good GUI is needed. The .NET framework offers this possibility by providing library classes, through both Windows Forms (WinForms) and Windows Presentation Foundation (WPF), facilitating the development of graphical user interfaces. [50]

There are several positive aspects of the .NET framework for the development of this system. It contains implementations of the algorithms desired designed to give flexibility when developing and also functionality for generating keys for the algorithms in a secure way. Using .NET with C# is common at MBM, so using .NET would also support maintenance and possible further development. The support in the framework for developing graphical user interfaces is also a positive aspect, since it has been proven efficient and easy to use from personal experiences. One negative side of using .NET is that applications developed using it only run on Microsoft Windows operating systems. It is however only a requirement for the system to be developed to run on these operating systems.

3.2.5 Conclusions

The new system is required to handle large amounts of log data without making it accessible to the user. Based on the pre-study for confidential logging (See section 3.2.2), cryptography is chosen to handle the confidentiality. The encryption needs to be able to encrypt large amounts of data in real-time to avoid having to store too much of the logs in plaintext temporarily first. It must also be secure in terms of key handling. The key available in the users' computers should not be able to use for decryption since it could, even if hidden in the code, theoretically be accessible through reverse engineering for example.

AES is, as mentioned earlier (See section 3.2.2.1.2), a block cipher, which would be useful since it allows encryption block-by-block in real-time to solve the issue with having to store logs in plaintext first. It is also secure and very fast. Using a stream cipher would also allow encryption in real-time, but because of the poor security stream ciphers are discarded. The downside of AES for this particular setting is that it uses the same key for both encryption and decryption since it is symmetric. That means that the key used for encryption could also be used for decrypting the logs and, even if hidden in code, the key could theoretically be accessed.

RSA is, as mentioned in section 3.2.2.2.1, a public key algorithm that uses different keys for encryption and decryption, one public and one private. This is a feature that would be very useful for the system to be implemented, and only one pair of keys would be needed since no authentication is required. RSA does not however have the feature to encrypt block-by-block, unlike AES. Using only RSA would mean that either the whole log would have to be stored in plaintext first, or that pieces of the logs would be encrypted separately when receiving the log data. The second alternative would not be fast enough and also would also increase the log sizes, because of the minimum size of ciphertexts and padding.

Both AES and RSA are considered unbreakable and have different useful features for the system, so the decision was made to use a combination of both to get the features needed. AES, in CBC cipher mode, will be used to encrypt the log data block-by-block in real-time to a file when logging. The key, 128 bits long, and IV for AES will be randomized at each run. When a log session has stopped, the randomized AES parameters, the key and IV, will be encrypted with a public 2048-bits RSA key using OAEP padding. The corresponding private RSA key will only be accessible when decrypting. By using this scheme the encryption will be both fast and theoretically unbreakable. The setup of encrypting AES keys with RSA is used in many applications for these reasons [19].

Since existing solutions and tools do not provide sufficient functionality in terms of confidentiality and modularity the decision was made to develop a new system. Investigating the possible means of software reuse for this system, the most appropriate alternative seemed to be to make use of libraries to get both well-tested, high-quality implementations and flexibility.

The .NET framework was chosen as provider for these libraries, mostly because of its features, regarding security and GUI development, but also based on general previous experiences. According to Pilipchouk [51], the cryptographic features of Java and .NET are pretty even, with Java having some a bit more complicated solutions because of the US cryptography export restrictions. Noticeable is that this comparison originates from 2003, and a lot has happened to the platforms since then. Regarding encryption speed, the performance of both platforms'

AES implementations was seen as important. According to Mel and Baker [52] (referenced by [53]), AES performs consistently in a wide range of computing environments. This performance consistency of AES (128-bits) was also shown in performance tests between .NET 2.0 and Java 2 by Francia and Francia [53]. Looking only into security and cryptography, neither of the platforms outstood the other. Looking into the libraries for GUI development, .NET's WinForms and WPF were seen as superior to the libraries provided by Java's JFC. The older WinForms was chosen over WPF, mostly because much more information is present in case support is needed, and because of previous experiences.

Regarding the log data collection, the .NET framework was also seen as superior for this setting. The Tool A server was namely decided to be used, since this re-usage was encouraged by MBM and seen as very beneficial time-wise. The cross-language integration between C# and C++, which the server API is written in, is supported by the .NET CLR [54] and should therefore not cause too many problems. Also, accessing WinAPI was seen as likely to be easier with the .NET framework.

3.3 Solution

Given the conclusions from the pre-study, described in 3.2.5, this section will firstly describe the overall system design. Secondly, the different applications forming the system will be described, together with some relevant design and implementation information.

3.3.1 System

Because the log collection is done by customers and the log reading, and analyzing, is done by MBM employees, separating these tasks into different applications seemed appropriate to facilitate the distribution and key accessibilities. To have all the functionality in one single application would mean that both the encryption and decryption keys could be possible to retrieve for both customers and MBM employees.

The system created was made part of a new application suite named Ericsson Mobile Broadband Logging Applications (EMBLA) and three different applications for log collection were made part of it. The three different applications were named Logger, Decryptor, and KeyGenerator. Logger is used to confidentially log data, Decryptor is used to decrypt the logs encrypted by Logger, and KeyGenerator is used to generate new sets of RSA keys.

This third application, KeyGenerator, was created because of the risk with leaked or lost keys. Because of this, it was seen as necessary to be able to easily switch keys in the applications, even though default keys are included at installation. The solution that was thought to be the simplest for the managers of the system was to have the decryption key as an external file in the decrypting applications and the encryption key as an embedded file in the logging application. The embedded solution is to prevent an attack where a malicious user could replace an external key file with a custom one that he/she has the corresponding decryption key to. Both key files are of the same format as the output from KeyGenerator. The keys are stored in XML³⁴ files, since the format is supported by the used cryptographic classes. Logger will only have a public key for encryption and Decryptor will have the corresponding private key for decryption.

³⁴ A specification for creating custom markup languages

A system overview showing the new flow for the log collection is pictured in Figure 4. Logger will be used by customers to gather log data and produce logs, encrypted with MBM's public key, without ever making plaintext log data accessible. The encrypted logs will then be sent to MBM employees who will use Decryptor with the corresponding private key to decrypt the logs into plaintext files. The RSA keys used have first been generated by a manager of the system with KeyGenerator. The outputted public key is distributed with Logger and the private key with Decryptor.

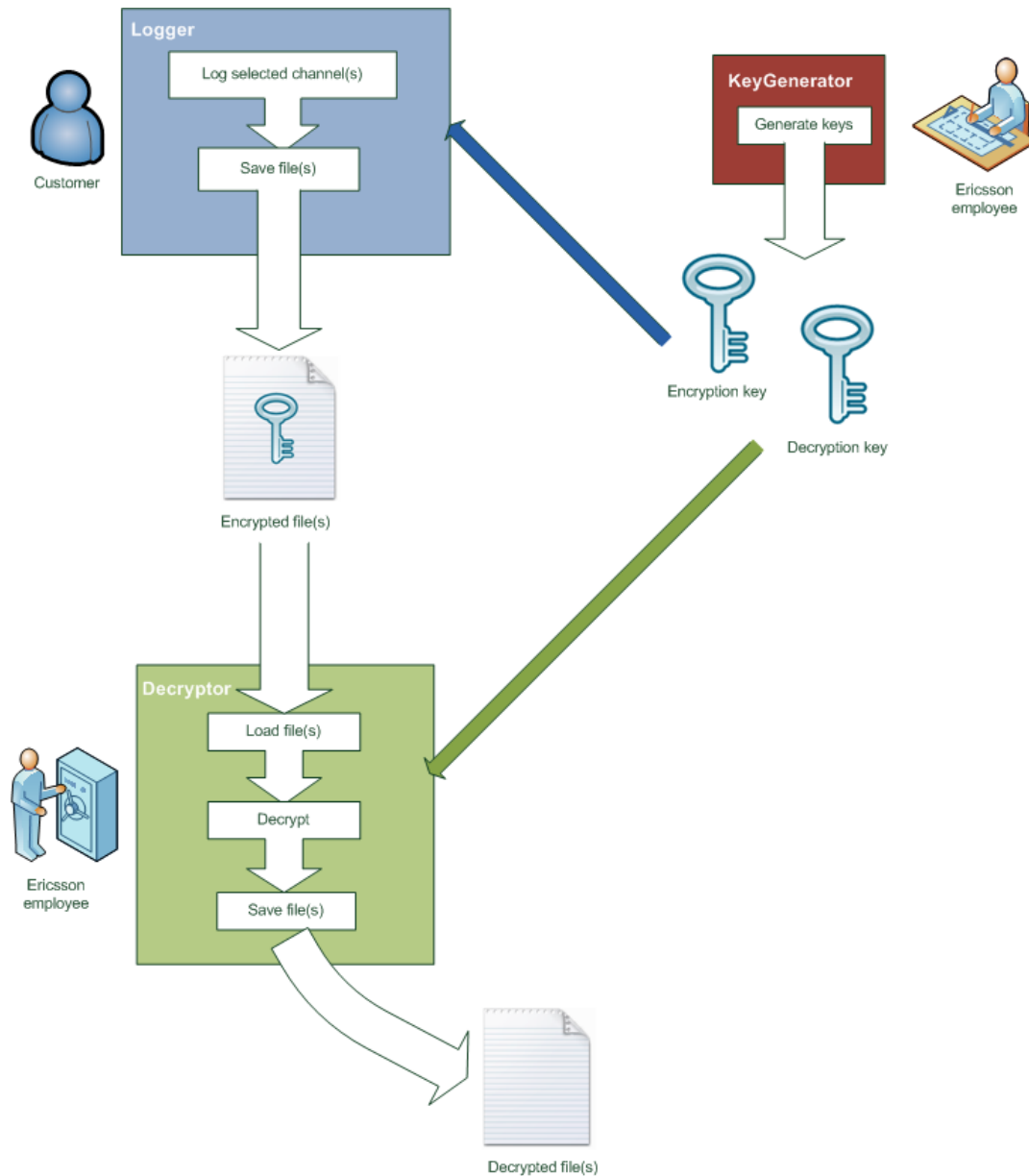


Figure 4. System overview

3.3.2 Logger

As described earlier, Logger is an application used by customers to confidentially log data. This section will describe the main functionality of Logger, its usage and management, and some relevant software design and implementation information.

3.3.2.1 Functionality

Logger's main task is to collect log data and continuously encrypt it as it is collected. It is therefore possible to select what data sources to log and to start/stop logging sessions when desired, the sources being firmware, WinAPI and the network. Simultaneous logging of several sources is also possible.

While logging, statistics, such as current size of logged data and spent time etc., are shown in the GUI making the logging progress more visible. This is especially important since the actual log data is not visible and the user would not know that any data is being logged without statistics. It is also possible to add markers, i.e. custom text messages, in the logs while logging. These markers can be used by the user to indicate that a certain action has been or will be taken, which might make it easier for the reader to locate specific action's impacts in the logs. After a logging session, the user is offered to save the logged data as files, one per data source logged.

To log firmware, the application allows the user to connect to the module via a COM interface. When logging a firmware data source it is also possible to send scripts, series of text commands, to the module via so called interactive data providers.

Often, the actual startup of modules is of interest when analyzing logs. Since the modules are started together with the computer and the operating system, it is not possible to capture that startup. Luckily, a module restart can be triggered by sending AT commands. Logger therefore offers this functionality together with the option of automatically starting to log data sources of choice as soon as technically possible.

3.3.2.2 Design and Implementation

Logger is a Windows Forms application, and not a console application or alike, due to the requirements on usability and the amount of functionality present. The GUI (See Figure 5) has a tree view to the left showing all possible data sources to log. The different icons show what actions are possible to perform on the different tree view nodes, some examples being script sending, resetting, or selecting for logging. On the right side, there is a control section where the starting and stopping of log sessions is made. Below is a sub-section for adding markers to the logs and another showing the statistics. At the bottom of the window, a help section with recommended actions is also present.

MBM's public RSA key used for encryption is embedded at build time from a XML-file and part of the application when installed. It is therefore possible to easily switch the key to another one generated by KeyGenerator if needed.

The application was designed to have high error tolerance. For example, logging is not interrupted regardless of the status of the broadband module, even in the case of a crash. This is important since logging is used to capture erroneous behavior, sometimes including module crashes.

Logger uses the Tool A server application (described in section 3.2.3.1) for communicating with the mobile broadband module. The server is installed together with Logger and is then started by Logger at startup and stopped when exiting.

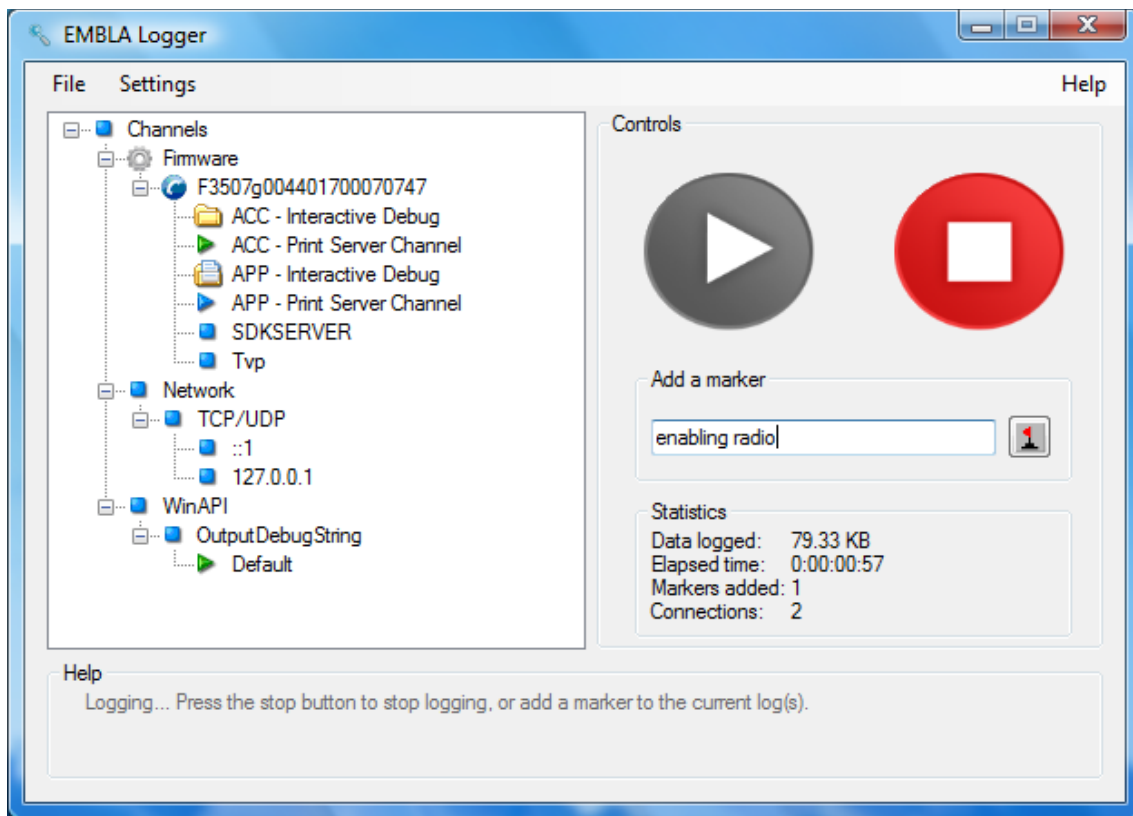


Figure 5. Screenshot of Logger's GUI

The application consists of many different classes with separate functionality structured in different namespaces, providing an intuitive architecture supporting maintenance and further development. A diagram showing the namespaces and their main classes can be seen in Figure 6. This distribution of functionality amongst different modules supports modularity [55]. To achieve modularity, efforts when designing was spent on minimizing the coupling³⁵ and maximizing the cohesion³⁶ of the modules, as suggested by Blundell et al. [55].

One particular part of the architecture that was designed for modularity is the implementation of the data sources used, since it is desirable to relatively easily add or remove new data sources. This is achieved by using the same pattern for the data sources with a monitor class handling the communication with the external data source and a logger class deriving from the base class `ChannelLogger` responsible for the actual logging. This provides an architecture where an implementation handling a new data source can easily be plugged in by implementing a monitor class and a logger class.

³⁵ A metric for the degree to which modules rely on other modules

³⁶ A metric for the singularity of purpose of a module

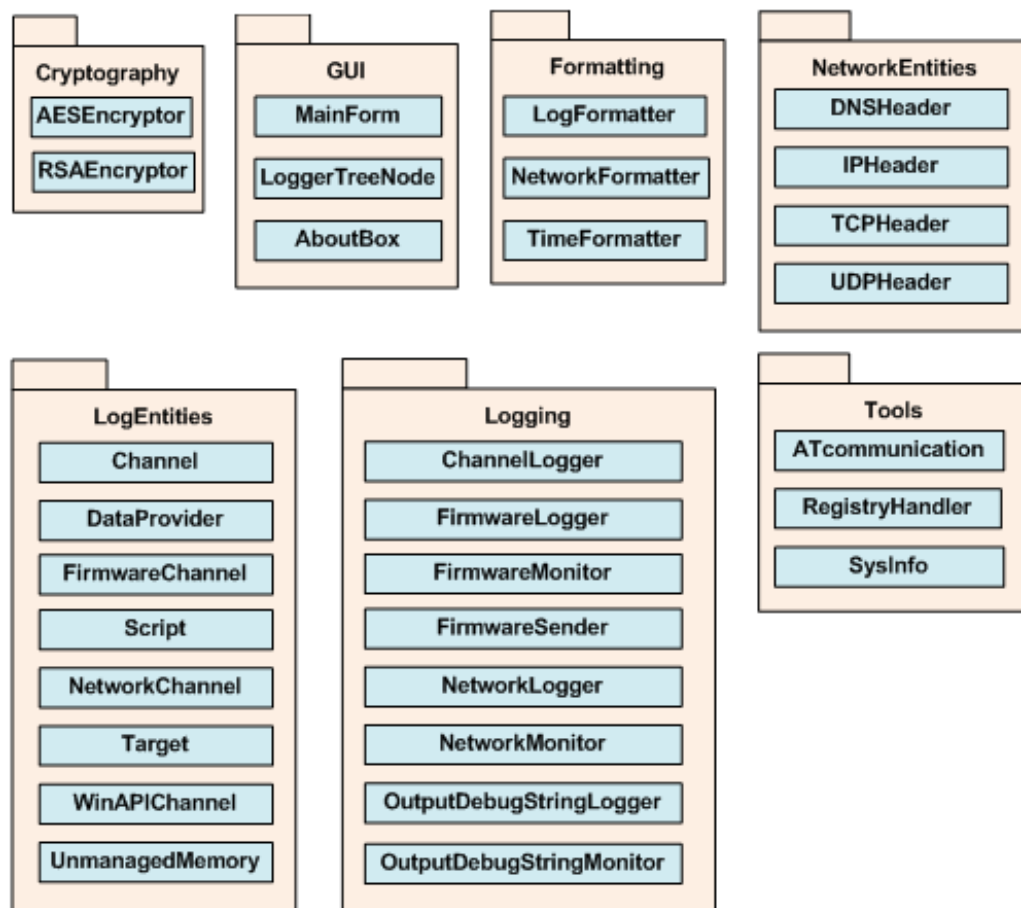


Figure 6. Simplified namespace diagram

3.3.2.2.1 Log Data Collection

The application can log debug messages from the modules' firmware, Windows debug messages from WinAPI, and network communication in the form of TCP and UDP packets. It uses the Microsoft Windows Dynamic-Link-Library³⁷ (DLL) kernel32.dll, exposing applications to parts of WinAPI, for collecting debug messages. Network log data is collected by using the .NET framework's functionality for sockets³⁸ to receive TCP and UDP packets.

To communicate with the firmware, the Tool A server is used and interfaced via a C++ API implemented in a DLL file. Logger uses this API to register as a client to the server by providing a pointer to a callback function. This allows the server to make asynchronous calls to Logger when events occur in the broadband module. The server provides functionality for connecting to different data providers available. When connecting to a data provider, Logger provides another pointer to a callback function. When connected, the server asynchronously calls the callback functions provided when new data to log is available from that data provider, triggering the encryption and logging in Logger. The server application is started by Logger on startup and closed on exit. How C# and C++ are combined to be able to communicate with the server is described in the following section.

³⁷ A library of executable functions or data that can be used by Windows applications

³⁸ Endpoints of bidirectional communication over IP-based networks.

Combining C# and C++

The Tool A server client API used for communicating with the mobile broadband module is a Dynamic-Link Library file (DLL) written in C++. Since the DLL does not support a COM interface, calls to the API are done using .NET's Platform Invoke Services (P/Invoke) functionality, which enables native code, unmanaged code, to be called from managed code. To do this, the function signatures in the API were studied and the native types used by the functions were converted from/to managed C# types.

Using P/Invoke to call functions in unmanaged code from managed code also requires explicit handling of memory allocation, since the unmanaged code cannot write to the managed memory used by .NET's CLR. This is needed in the cases where the server writes its return values to parameters passed when calling it. In these cases, managed memory is allocated before the call and a pointer to that memory is passed as parameter. After the return value has been used, that memory must be freed again. The .NET framework provides support for allocating unmanaged memory and for marshalling³⁹ data from unmanaged memory to objects in managed memory.

This procedure can be unreliable since features such as type safety and garbage collection are lost. To make sure memory is released properly after each time the Resource Acquisition Is Initialization (RAII) idiom is used. RAII is a design pattern in object-oriented programming that ensures resources are released by tying them to the lifespan of objects [56]. By using this idiom the memory is guaranteed to be released even in the case of exceptions when execution is not successful. The RAII idiom is used in this application by allocating memory through a separate class and taking advantage of the "using" keyword in C# which provides support for deterministic finalization since the Dispose() method of the object is guaranteed to be called after the using block. The following is an example of the RAII idiom:

Class (UnmanagedMemory.cs):

```
public UnmanagedMemory(int amountToAllocate){
    strPointer = Marshal.AllocHGlobal(amountToAllocate);
}
public void Dispose(){
    Marshal.FreeHGlobal(strPointer);
}
```

Usage:

```
using (UnmanagedMemory um = new UnmanagedMemory(STRING_BUFFER_SIZE)){
    //Do something with um.strPointer here
}
```

3.3.2.2.2 Confidentiality

As described the pre-study conclusions (See section 3.2.5), cryptography was chosen to handle the confidentiality requirements. AES encrypts log data block-by-block in real-time and RSA is used to encrypt the AES key.

³⁹ The process of preparing data for processing

Since RSA was to be used in all three applications and AES in both Logger and Decryptor, a namespace, package, called `EMBLA.Cryptography` was created. This namespace contains a base class for the RSA classes, called `RSA` and a base class for the AES classes, called `AES`. These base classes have references to instances of their algorithm implementation classes and some algorithm settings. The `RSA` class uses `RSACryptoServiceProvider` as its implementation of the RSA algorithm and the `AES` class uses `RijndaelManaged` as its implementation. Logger has, in its cryptography namespace, an `AESEncryptor` class and an `RSAEncryptor` class. Having presented the cryptography namespace, the encryption workflow will now be described (See Figure 7)

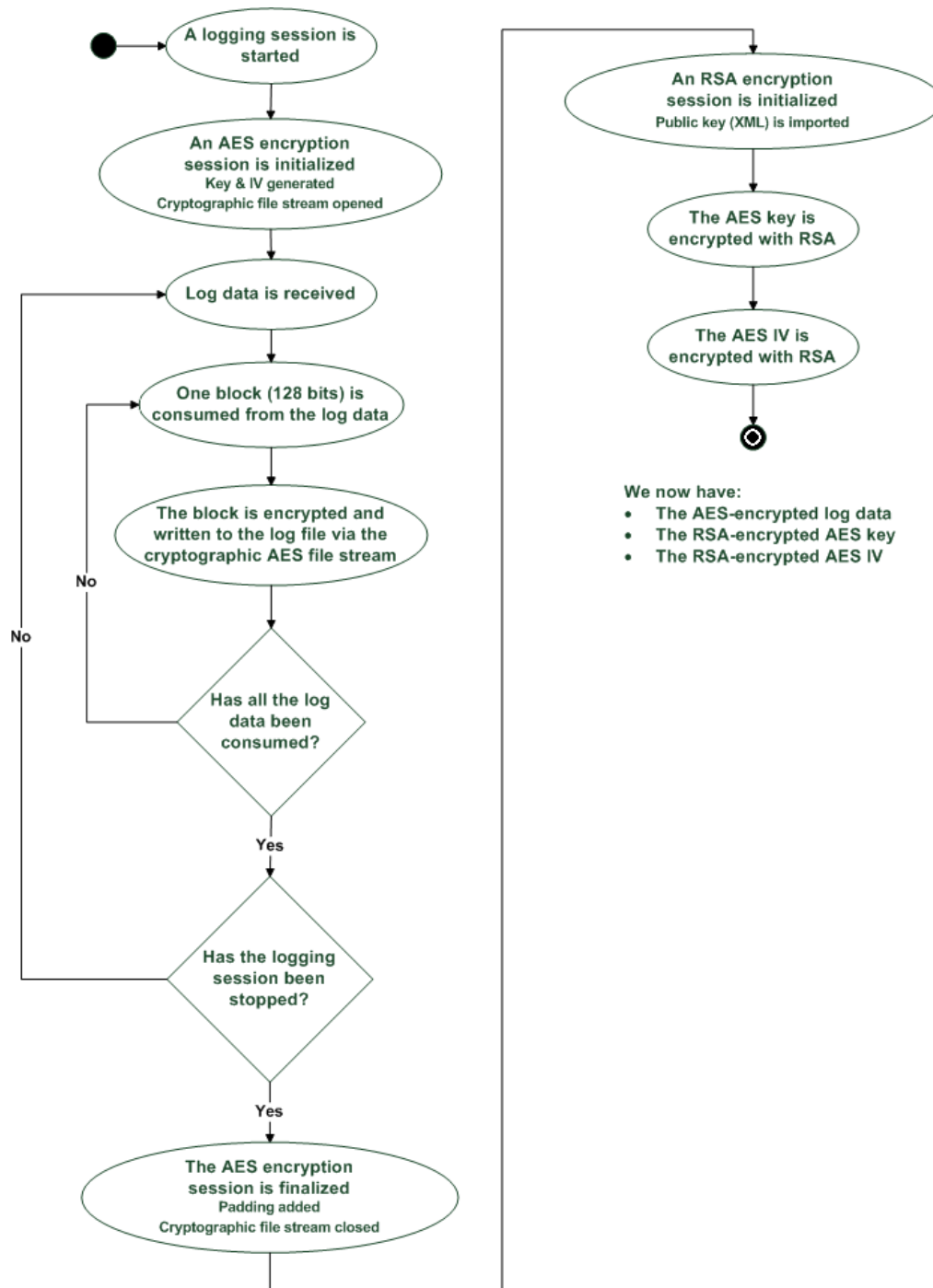


Figure 7. The encryption workflow

The Encryption Workflow

The **AESCryptor** is initialized when a logging session is started. At initialization, new key parameters used for that session are generated and a cryptographic file stream to the log file is opened. The key size used is 128 bits, the block size is 128 bits and the mode is CBC. As log data is received asynchronously, it is consumed by the **AESCryptor** class block-by-block and encrypted to the log file. When a logging session is terminated, the encryption is finalized by adding padding and closing the cryptographic file stream. The next step is now to encrypt the AES key parameters with the public RSA key. For this purpose, the **RSACryptor** is initialized by importing the key from the embedded XML-file in the application. The key-size used here is 2048 bits and OAEP padding is used to prevent certain attacks (See section 3.2.2.2.1).

The Log Scheme

Having the AES-encrypted log data and the RSA-encrypted AES parameters, this data had to be packaged into a single log file, to be easily communicated. A simple log scheme was therefore designed for this purpose (See Figure 8). A new file created after a logging session where the RSA-encrypted AES key parameters, the key and IV, are written to the beginning of the file, both preceded by an integer indicating their size in bytes. This is necessary for being able to parse, unpack, the log file later. By parsing log files by following this scheme, it is therefore possible to extract and decrypt all individual parts of the file.

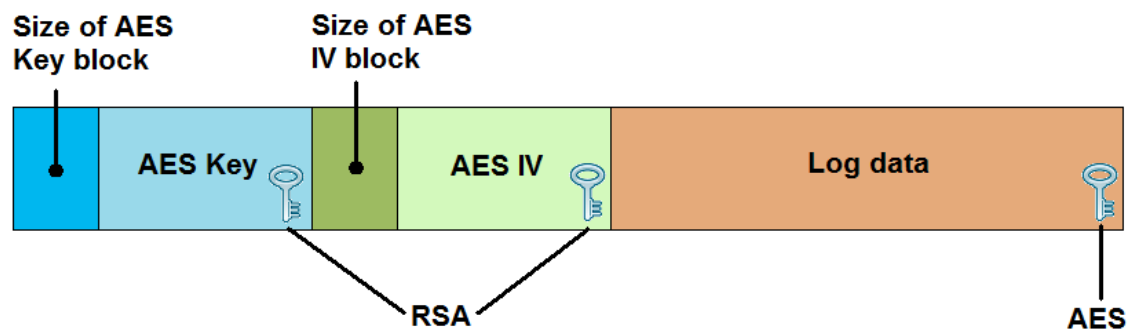


Figure 8. The log scheme

3.3.3 Decryptor

This section will describe the main functionality of Decryptor and its design, together with some relevant software design and implementation information.

3.3.3.1 Functionality

Decryptor's main function is to decrypt the logs encrypted by Logger, allowing multiple encrypted files to be loaded at once for convenience. The decryptions, plain text log files, are then stored to a directory chosen by users. As with Logger, the private key can be exchanged if needed.

3.3.3.2 Design and Implementation

Decryptor is a Windows Forms application. This decision was made because it was seen as preferable from a usability point of view, even though a console application would have been adequate for the functionality provided. To be able to decrypt multiple files at once, and have the possibility to both browse for the files or drag and drop them, a Windows Forms application seemed more appropriate.

Since the amount of functionality available and the information needed for users are limited, the GUI was decided to be kept simple. A screenshot of Decryptor's GUI is seen in Figure 9.

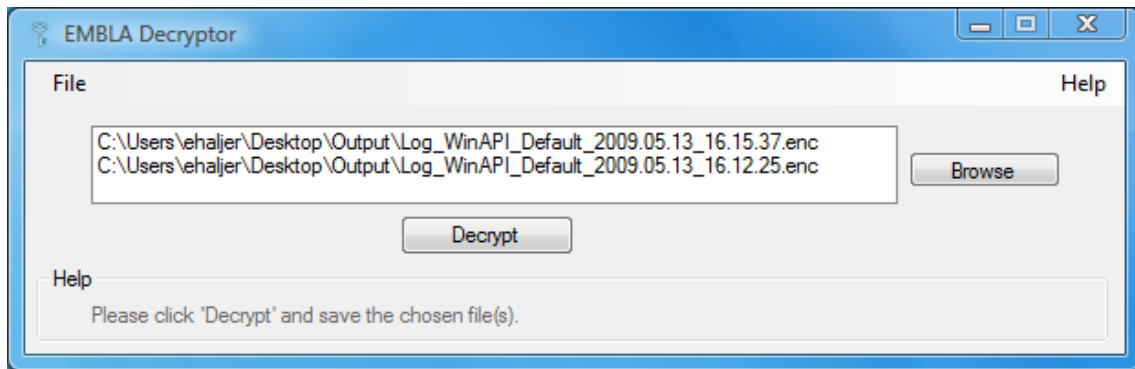


Figure 9. Screenshot of Decryptor's GUI

Regarding the software architecture, a simplified class diagram of Decryptor can be seen in Figure 10. Decryptor also has an exception namespace, resource files and a settings file. [MainForm](#) is the main GUI class controlling the main window shown in Figure 9. When files are loaded and decryption is triggered by users, the files are parsed by the [DecryptionController](#) class according to the log scheme described in section 3.3.2.2.2. If the parsing of the RSA-encrypted AES key and IV was successful, we now have the byte offset in the file to the log data and the AES parameters encrypted with MBM's public RSA key. [DecryptionController](#) now therefore triggers the two RSA-encrypted AES parameters to be decrypted by the [RSADecryptor](#) class, inheriting functionality from the [RSA](#) class in the [EMBLA.Cryptography](#) namespace. Having the offset to the log data and the AES parameters needed to decrypt it, [DecryptionController](#) now triggers the [AESDecryptor](#) class to decrypt the log data. After decrypting, the application prompts feedback about the decryption and queries for a location to save the files. If the file parsing or the decryption fails, the application gives feedback to users.

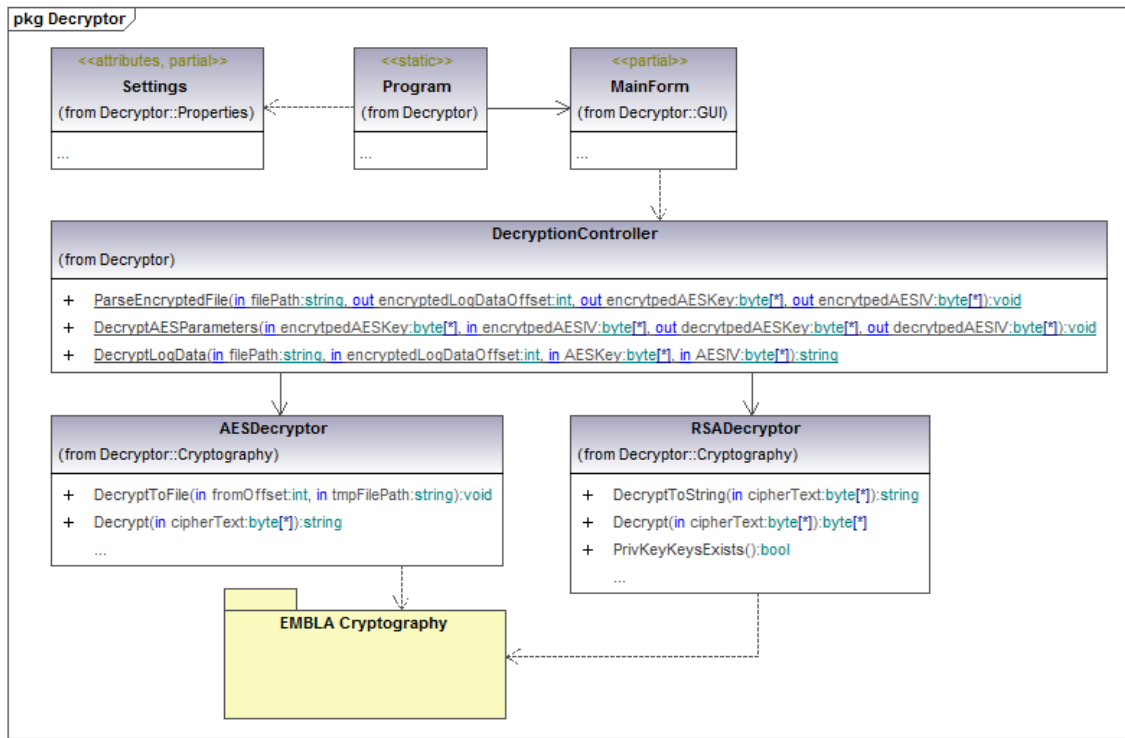


Figure 10. Simplified class diagram

3.3.4 KeyGenerator

This section will describe the main functionality of KeyGenerator and its design, together with some relevant software design and implementation information.

3.3.4.1 Functionality

KeyGenerator's only task is to generate new sets of RSA keys to be used with by Logger and Decryptor. The output is stored in one public key file and one private key file, both in XML-format. For the sake of user-friendliness it is also possible to specify the output directory for the keys via console arguments, and their filenames via the application configuration file.

3.3.4.2 Design and Implementation

KeyGenerator is a Windows console application. The reason for this is the simplicity and single-functionality characteristic of the application. Because of the lack of options for users a console application was found most suitable. The output filenames are specified in the application config.exe file and the output directory is given as console argument to the application. An example execution of KeyGenerator is seen in Figure 11.

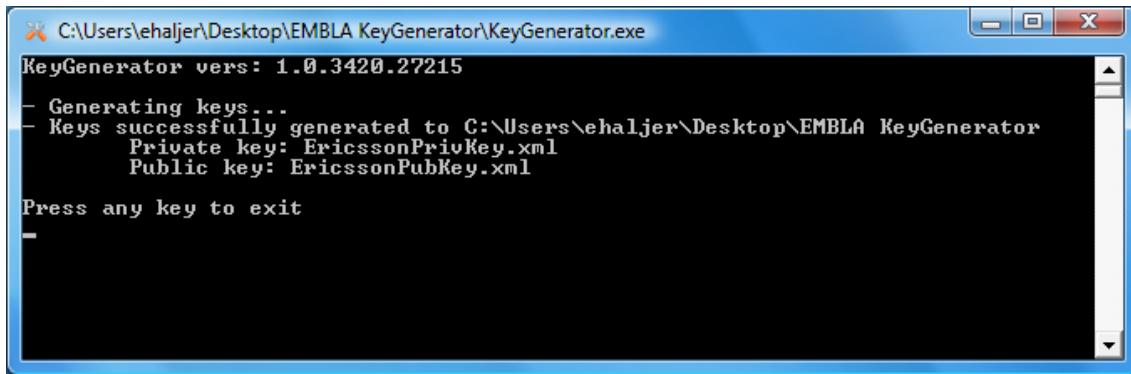


Figure 11. Screenshot of an execution

The software design of the application is very basic because of its simplicity (See Figure 12). The main class, [Program](#), controls the flow of the program and parses possible arguments given. To generate the keys, the [RSAKeyGenerator](#) class is used, inheriting functionality from the [RSA](#) class in the [EMBLA.Cryptography](#) namespace. KeyGenerator also has, as the previous two applications, an exception namespace and a settings file.

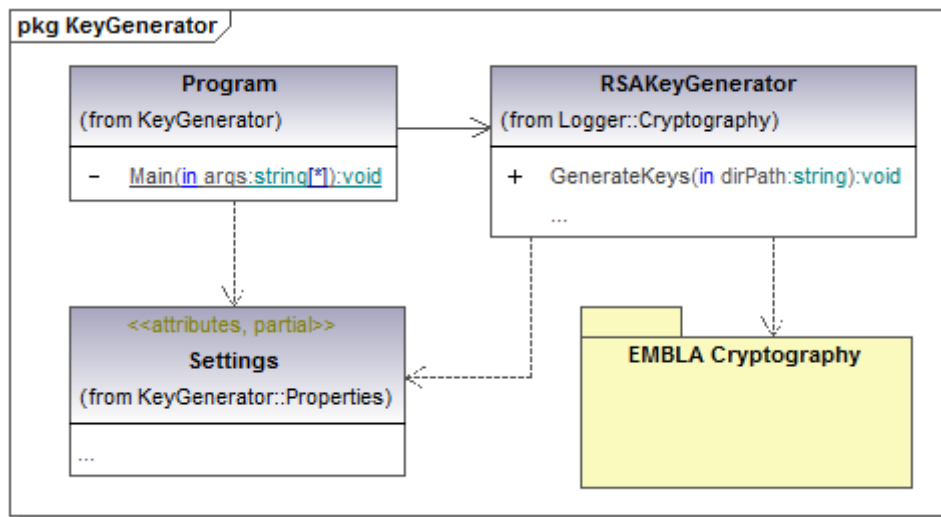


Figure 12. Simplified class diagram

The actual key generation is provided by the [RSACryptoServiceProvider](#) class in the [System.Security.Cryptography](#) namespace. As mentioned in section 3.2.4.3.2, the PRNGs used for key generation in .NET all follow specifications from well-established agencies and are considered secure.

4 Log Analysis

This section contains the second part of this master thesis work, the introduction of a system which facilitates the analysis of the collected logs at the department. The collected logs might originate from systems such as the one introduced in the first part of the report, Log Collection (See section 3).

As in section 3, the rationale behind the system is firstly described. The following section then presents the pre-study made to acquire the knowledge and insight to solve the problem. Finally, the actual solution is presented.

4.1 Rationale

As earlier mentioned, many tools are currently used by MBM to analyze the logs collected by employees or received from customers. The high amount of tools has lead to many different ways of working amongst the employees. Therefore, the communication amongst them of how to solve certain problems suffers. An application tailored according to their needs, making as many of the existing tools as possible redundant, is therefore requested.

Some tools providing some of the requested functionality exist, but they are most often licensed and cost money, or/and suffer from poor performance and poor usability. Some even suffer from poor stability.

The performance is seen as crucial since logs of over 100MB are not uncommon. The analysis of logs of this size therefore takes valuable time away from the employees if not done with high performance. Regarding usability, existing tools are not very user-friendly which disturbs the employees work. The poor performance also increases the sense of poor usability, since many operations are slow and inflexible. Finally, stability is of high importance, not only for usability, but also for not losing any work not yet saved.

4.2 Pre-study

Having described the rationale, the pre-study performed, with the aim of finding a solution to the described issues, is now presented. This section starts with a description of the requirements and the tools currently used. The following sections present the pre-studies performed to find solutions to the most important requirements, namely: Usability, performance and 3GPP message translation. This section is then concluded by describing and motivating the chosen solution.

4.2.1 Requirements

With the rationale in mind, the fundamental functional requirements of the application were decided and documented (See Appendix A).

First of all, users must be able to open any plain text log file and get it displayed by the application. Some of the messages in the firmware that are most often analyzed are log messages containing data according to standards by the 3rd Generation Partnership Project⁴⁰ (3GPP). These 3GPP messages, containing data in hexadecimal format, should be translated according to the standards so that they are more easily understood by employees, by

⁴⁰ A collaboration between telecommunications associations groups specifying 3G mobile phone systems

displaying simplified information about their meaning. To translate messages in detail, another application exists. It is therefore a requirement that this application should be able to interact with that application.

Many times, logs contain a lot of data that is of no interest for the analyst. It is therefore a requirement that the application allows custom made filters to be created and used in order to be able to better focus on certain events etc. Some users also like to mark certain lines and be able to extract them from the application and send them other employees or customers. These marked lines should be displayed in a separate view. It is also important to have some kind of linking between the original log and the other different views in order to be able to easily jump between the different views without losing orientation.

Another desirable requirement is that the above mentioned functionality with filtering and translation should work in real-time when tailing a file, i.e. changes done to an opened log file that is being written to should be visible in the application. A search function must also exist, as well as standard editing commands such as cut, copy, paste, delete, where appropriate.

The application is going to be used at MBM where the operating environment consists of Microsoft Windows PCs with fairly good performance. There are requirements on the speed of the analysis though, since the analyses of big log files with existing tools are seen as slow and lock up the computer while working.

One important finding during the requirements elicitation was that many people have the need for different features and very different ideas about how to use the application. A consequence of this is to make the system as configurable and flexible as possible.

Since the application should be tailored for the needs of the department and many requirements are very specific, searching for existing solutions matching the requirements was discarded. The decision was instead made to develop a system from scratch since it was believed to be faster and more efficient than trying to locate reusable existing software components, frameworks and libraries excluded, and applications. It was also decided to use the .NET framework and C# for the development because of positive personal previous experiences, because they are commonly used at the department, and because of consistency with the log collection system (See section 3).

4.2.2 Tools Currently Used

As written in the rationale, section 4.1, too many tools are used for analyzing logs. Most of the tools are advanced text editors with filters, highlighting and other features. Exactly what tools are used is seen as confidential, but the most commonly used tool for 3GPP message translation will now be briefly presented in order to highlight the problems with it. This tool, from now called Tool B, will also be used as a performance benchmark.

Tool B supports functionality such as: 3GPP message translation, filtering, and searching, and is also linked to an external application for displaying detailed information about 3GPP messages, which is described later in this section. One issue with Tool B is that it does not support tailing functionality. Another issue is that the application is very slow. Opening very large log files with 3GPP message translation and filters enabled takes a lot of time. This problem becomes even more severe since the GUI locks and does not show any progress while loading and

analyzing. The application is also not very stable either and crashes happen occasionally. The application has the fundamental functionality required, but it is not very user-friendly and does not support all the features wanted.

The application displaying detailed information about 3GPP messages is being used by Tool B to show additional information about 3GPP messages that it is not capable of displaying itself. When opening a 3GPP message, the hexadecimal part of the message is translated and the contained information is displayed. The user can choose which external 3GPP message descriptions the application should use allowing the user to switch between different versions of descriptions and easily use new ones when available. This application is seen as a useful tool for analyzing 3GPP messages and is therefore desired to be externally linked to the application to be developed.

4.2.3 Usability

To be able to develop an application with high usability, not only the requirements are important to understand and implement. The user interface (UI) design is of high importance too, since a poor design might hide functionality and features from the users [3]. Therefore, this section will present the pre-study performed regarding both some UI design principles and the UI design process.

When designing user interfaces, one needs to take human factors into account [3]. Sommerville [3] has, based on these factors, presented a list of UI design principles. These principles and their definitions are cited in Table 2. Tidwell also describes this process in the book “Designing Interfaces” [57].

Principle	Description
User familiarity	The interface should use terms and concepts drawn from the experience of the people who will make most use of the system.
Consistency	The interface should be consistent in that, wherever possible, comparable operations should be activated in the same way.
Minimal surprise	Users should never be surprised by the behaviour of a system.
Recoverability	The interface should include mechanisms to allow users to recover from errors.
User guidance	The interface should provide meaningful feedback when errors occur and provide context-sensitive user help facilities.
User diversity	The interface should provide appropriate interaction facilities for different types of system users.

Table 2. User interface design principles [3]

The UI design process itself is also important in order to strive towards the principles wanted. Sommerville [3] presents three core activities for the UI design process: User analysis, system prototyping and interface evaluation, each with their own benefits which can improve the process.

4.2.4 Performance

The functions identified as having poor performance in the currently used tool are, not surprisingly, the filtering and translation of the logs. The filters in the existing tools are plain text filters, which is also what is requested for this application, and therefore require many string operations. The 3GPP message translations also require many string operations since the 3GPP messages themselves must be identified in the logs and then translated. Since all rows in a log file, potentially millions, must be processed, it is crucial to have an efficient handling of strings and string operations to have high performance. Another part identified as crucial for performance is the actual translation of the 3GPP messages, when already identified, which will be described in section 4.2.5.

One aspect of string operations that is important is that strings in C# are immutable⁴¹ (See the [String](#) class [47]). All modifying operations on strings result in new strings being constructed and therefore memory allocated. By minimizing these kind of operations a lot of performance could potentially be gained.

One example of where this is useful to consider is when doing case-insensitive operations to check if a string is contained within another string. Intuitively, these checks are made by making both strings either uppercase or lowercase and then using the [string.Contains\(string value\)](#) method. The problem with that approach is that the [string.ToUpper\(\)](#) and [string.ToLower\(\)](#) methods allocate new memory since they construct new strings. It is a lot more efficient to use the [string.IndexOf\(string value, StringComparison comparisonType\)](#) method, which uses the .NET [StringComparison](#) class [47] to choose if the comparison should be case-sensitive or not. The result of this call can then be compared to -1 to check if the string was contained, since -1 is returned if the string is not found. Below are examples of the intuitive, but slow, approach and the faster preferred approach:

Intuitive:

```
if (s1.ToUpper().Contains(s2.ToUpper())) { ; }
```

Fast:

```
int result = s1.IndexOf(s2, StringComparison.OrdinalIgnoreCase);  
if (result != -1) { ; }
```

The [StringComparison](#) class also provides support for both culture-sensitive and culture-insensitive, ordinal, string comparisons. Culture-sensitive comparison rules can be useful when for example sorting a list of strings containing language specific characters. When using culture specific comparison rules the relative order of all characters, even culture specific ones, is handled. This is not the case when using an ordinal string comparison, where the characters' corresponding integer values are compared which makes it a lot more efficient in terms of speed. For this application, using culture specific rules is not required and ordinal string comparison is instead used to provide higher performance.

To find a rough estimate on the difference in performance between these two approaches, a simple test was constructed. The performance depends on the size and contents of the strings used, so strings with values corresponding to a likely filter word and a log line were used. The

⁴¹ Cannot be modified after creation

test showed that when running the two pieces of code ten millions each the intuitive pattern required 21 seconds of execution time, while the faster only required 9 seconds, which is an improvement of about 60%. This gave a rough indication on the performance that could be gained.

Another example of where string comparisons can be optimized is where the same string value is used many times for comparisons. To be able to easily read the code and see what value is actually used in the comparison, the following pattern is often used:

```
string s1 = "VALUE";  
  
if (s1 == "VALUE"){ ; }
```

The problem with this approach is that a new string is constructed and allocated in new memory with the text "VALUE" each time the comparison is made. If that string is used for many comparisons, this pattern becomes inefficient. A more efficient way is to declare the string once and use its reference for all comparisons, preventing new strings from being constructed each time. A further step in optimizing this scenario is to only declare the string once and use the runtime's internal string pool, which contains a table with references to each unique string value. Multiple declared strings with the same value share the same reference in that table. By using this internal string pool for equality comparisons only references are compared, which is faster than comparing most string values. This approach becomes very efficient when a string value is used for many comparisons and is not modified. The constructing of a string becomes slightly more time consuming though. Below is an example of how the runtime's internal string pool can be used for string comparisons:

```
string s1 = string.Intern("VALUE");  
string s2 = string.Intern("VALUE");  
  
if (object.ReferenceEquals(s1, s2)) { ; }
```

To get an indication of the possible performance gain a simple test was again constructed. The performance of the equality comparisons described is very much depending on the size of the strings compared, so strings of sizes likely to be used in this application were used. The identified candidates for the test values were the keywords used when translating logs, described later in 4.2.5. Therefore, strings of short lengths, 6 characters, were used. When running the two latter mentioned approaches for equality comparisons a hundred million times each, the first more intuitive approach required 6 seconds while the next approach, using the internal string pool, required 4 seconds. Using the internal string pool can consequently give a rough performance gain of 30%. It should also be kept in mind that the difference in performance is much bigger when comparing large strings, so if the keywords used are changed in the future the performance gains could get even bigger.

Another approach possible for handling string operations is to use regular expressions⁴². Support for regular expressions is provided in the .NET framework, via the [Regex](#) class [47], and could be used for identifying strings of interest in log files. Using regular expressions is very flexible and advanced pattern matching can easily be done. For this application though,

⁴² Flexible means in computing for identifying strings and character patterns.

the required operations are only equality comparisons and checking if one string contains another. Much of the functionality in regular expressions is therefore abundant for this application. Using regular expressions for these simple operations required was identified as slower than using the approaches mentioned above, possibly because of the abundant functionality provided, and was therefore discarded.

4.2.5 3GPP Message Translation

To be able to translate 3GPP messages, the first step is to be able to identify them in logs. By using other tools, one can observe that 3GPP messages are log lines containing certain keywords followed by a string in hexadecimal format. Depending on the keyword, the hexadecimal string should be translated differently. The information on how to translate the hexadecimal strings was found to exist in the form of two resource files, text files, used at other parts of Ericsson, which are based on information from 3GPP. Since these files are used in other applications and the information in them is seen as satisfying, these files were decided to be beneficial to use in the developed application.

The first file, the rules file, contains rules for how to parse the hexadecimal strings, but bit-wise. These rules are formatted into so called scripts (See Figure 13), which correspond to a single keyword. The scripts themselves consist of several steps, each denoted by a step number, an action, the number of bits to handle, a value, and an action to perform if the current action succeeds and an action if it fails.

[KEY WORD] <UL CCCH:>					
0	CHECK	1	1	1	2
1	SKIP	36	0	NEXT	INVALID
2	SKIP	1	0	NEXT	INVALID
3	CHKTBL	1	13	NEXT	INVALID
4	CHKTBL	2	2	END	INVALID
[SCRIPT END]					

Figure 13. An example of a script

The different actions types are: Check, skip, check table and call. The check action is a binary comparison between the value and the value represented by the number of bits to handle, from the current position in the binary message. The skip action means that one should skip forward the number of bits specified from the current position in the message. A call step is a call to a sub-routine, another script of the same format, specifying the sub-routine number in the success action field. Lastly, the check table action specifies a table id and row, with the number of bits to handle from the message representing the row and the value representing the table id. This table id and row are lookup values for the other resource file, namely the tables file.

The tables file is a text file containing several tables (See Figure 14), each having an id and multiple rows, each of them also having an id. The rows themselves are actual text messages giving simplified information about 3GPP messages. Each 3GPP message keyword script makes two table lookups, thereby giving two simplified information messages.

[TABLE ID] 2	
0	CELL UPDATE
1	RRC CONNECTION REQUEST
2	URA UPDATE
[TABLE END]	

Figure 14. An example of a table

4.2.6 Conclusions

Based on the problems with existing tools and the need for a tailored solution, it was decided that a new application was to be developed, supporting all the features wanted and with high usability and performance. C# and the .NET framework were decided to be used again

To reach high usability, many principles for good UI design were found in literature from Sommerville and Tidwell (See section 4.2.3). Regarding the UI design process, user analysis in the form of workshops and informal discussions will be used in order to get a clear picture of how the employees work with current tools. System prototyping will also be used in order to get quick feedback on the UI. It is much easier for users to express what they want regarding UIs when they see something tangible [3]. Prototypes with limited functionality will therefore be developed and shown to the employees, as often as possible. Beta versions with satisfying amount of functionality will also be distributed. A formal interface evaluation will not be performed, since the prototyping is seen as adequate and the process is too time-consuming for a project of this size [3]. Prototyping will instead be done until a certain level of satisfaction amongst the employees is reached.

Regarding performance, some interesting details regarding string operations in C# were found (See section 4.2.4), which will improve performance if used thoughtfully.

The application will also use the 3GPP resource files found and studied for translation of 3GPP messages (See section 4.2.5). It will also have support for interacting with the tool giving detailed information about 3GPP messages (See section 4.2.2).

4.3 Solution

Given the conclusions from the pre-study, described in 4.2.6, the new application will now be described. The application is called Analyzer and was decided to be part of the new EMBLA suite, described in section 3.3.1. This section will describe the system design of Analyzer, together with some relevant design and implementation information.

4.3.1 Functionality

The application was designed to provide functionality meeting the requirements described and facilitate the analysis of logs for users as much as possible. This section describes the main features provided.

One of the application's main functions is to translate 3GPP messages by showing simplified information about them. The application uses external files as specifications for how 3GPP messages should be translated. Settings for these external files' file paths can be set by the user in the application. When translating, users can choose whether both 2G⁴³ and 3G⁴⁴

⁴³ Second generation of telecommunication hardware standards

messages should be translated, or just one of them. 3GPP messages are also possible to open with an external application, launched by Analyzer with the currently selected message loaded, to get more detailed information about them.

Another main function is the filtering of logs. The application gives users the possibility of creating, importing, editing and exporting filters that can be used for this purpose. When importing a filter, in the form of a text file, each line in the file will correspond to one text item to search for when filtering. Filters can also be edited to add or remove individual text items to search for and there is also the option of choosing case sensitivity for each text item. When multiple filters are loaded, i.e. have been created or imported, users can choose which of them that should currently be active.

The original log contents, the filtered parts of the log, the translated 3GPP messages, and the marked lines, which are explained later, are shown in separate views. There is functionality provided for synchronizing the views, which will cause them to mark their lines closest to the line chosen by the user. This allows users to easily investigate lines from the same point in time in all views.

Two ways of opening a log file are provided, with or without tailing. It is also possible to start and stop tailing a file which has been opened without tailing. With tailing activated, the application is updated when data is written to the loaded log file by another application. The new data is then presented, translated and filtered so that all views are fully updated with the log contents. To make the newly added data visible, users have the option of enabling and disabling automatic scrolling of all views. With automatic scrolling enabled all views automatically scroll to their last line when new lines are added, allowing users to easily monitor the activity in tailed logs.

If the contents of one of the views are of certain interest, a function is provided for users to save the entire views of choice to text files. This could be useful when for example sharing information between employees when interesting results are found using the application. If only certain lines are of interest in the different views, users can mark these lines, adding them to a view which displays these marked lines. This prevents users from having to go back and forth between positions of interest, which would be cumbersome, especially when working with big log files. Lines from the view of marked lines can also be removed when desired. Analyzer also provides a search function which can be used to either find the next/previous matching line or to find all matching lines and showing them in a different view.

There is also functionality provided to copy, cut and paste lines using the Microsoft Windows clipboard. Using the clipboard allows users to copy and paste selections, single or multiple lines, as text into any text editor.

4.3.2 Design and Implementation

Analyzer is a Windows Forms application since such a GUI was seen as essential when working with data visualization of this kind. The application is designed to focus on the GUI because of its usability requirements, but is also designed to be fast at analyzing logs. Some focus was also put on maintainability, since the 3GPP information describing how to translate 3GPP messages

⁴⁴ Third generation of telecommunication hardware standards

will change over time. For simplicity, an installer was developed for installing Analyzer and its resource files.

The architecture of Analyzer consists of many different classes with specialized functionality, grouped together into appropriate namespaces (See Figure 15). Since much focus was put on speed, some aspects of the architecture are neither generic nor very maintainable. Instead, maintainability was favored in the parts of the application where further development or possible changes were likely to occur in the future, such as the 3GPP resource files. Despite this, the design still aims at reaching high modularity by minimizing the coupling and maximizing the cohesion of the modules, as suggested by Blundell et al. [55].

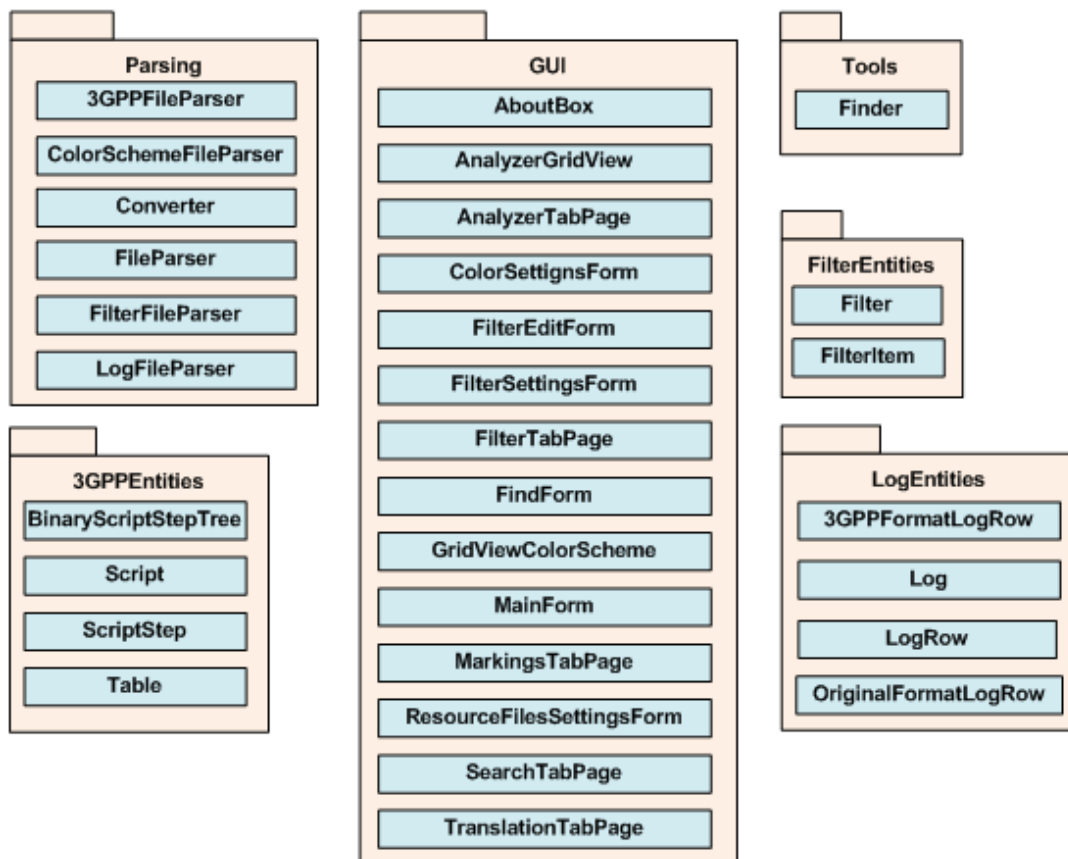


Figure 15. Simplified namespace diagram

The design and implementation fulfilling the usability requirements, the performance requirements and the 3GPP message translation will now be described.

4.3.2.1 Usability

During the user analysis it was discovered that the employees have a wide range of different opinions and ways of working, which inevitably creates a challenge when designing the UI. Because of the implicated need for configurability and versatility, the GUI features are important. When configurability was seen as inappropriate or impossible, what is thought to be generally good has been implemented, as supported by Tidwell [57].

The design chosen for the main window is a tiled pane design [57]. The main window is divided into two parts where the log file is shown in its original format to the right and a configurable workspace to the left (See Figure 16). This kind of design is appropriate for users who need to

see a lot of information at the same time [57], which is the case here. Having so called floating views could have improved this even more, but most users preferred the implemented configurable workspace with its tabbing system. The two panes are also color-coded differently for users to be able to quickly distinguish them from each other, as supported by Tidwell [57]. Colors are also used to enhance usability in the displayed logs via row stripping, which makes it easier to distinguish rows from each other [57].

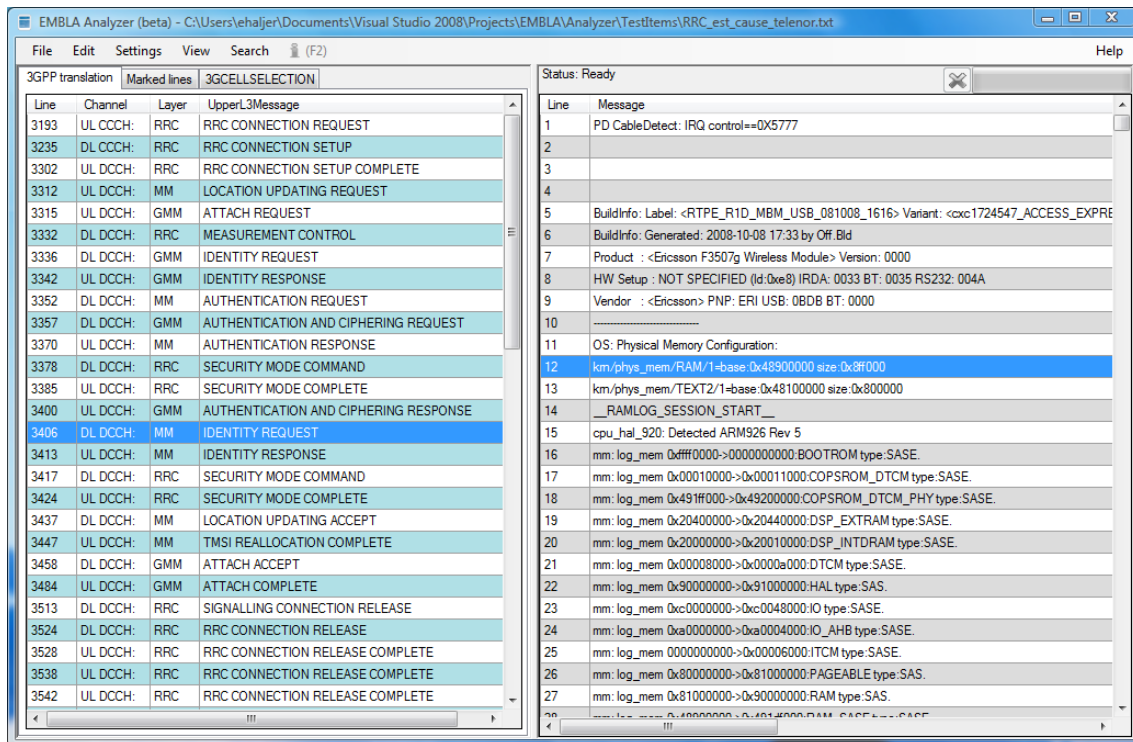


Figure 16. Screenshot of Analyzer's GUI

The configurable workspace area consists, as already mentioned, of tabs where the currently accessible tabs can be configured to support user diversity (See section 4.2.3) and that different tasks might require different configurations. Tabs containing the 3GPP message translation, the output from filters, the marked lines, and possible search results can be chosen. Each filter output, filtered log lines from the original log, is shown in a separate tab, but multiple filters can also be combined to one using a feature in the application's filter settings. This design allows users to choose only the features currently needed to make the interface easy to use. It is also beneficial for the performance of the application since the filtering and the translation of logs are very time consuming and only the work needed to fulfill the user's current wish can instead be done.

Regarding the very time consuming filtering and 3GPP message translation, users have the possibility to track the progress via a progress bar and also to cancel the work via a cancel button. Progress indicators make users more patient and cancelability is positive since users change their minds [57]. If the progress bar indicates that very long time will be spent on that action, users might want to abort for now and continue with other tasks. If users choose to cancel the current action the data loaded so far is displayed in the views. This makes cancelability also useful if only the beginning of a log file is of interest and the full analysis too time-consuming.

When the application is closed all the current settings, regarding everything from window positions and sizes to the workspace settings, are stored and loaded on the next startup. This also includes the currently open tabs and active filters, allowing users to easily continue working from the same configuration as when the application was closed last time.

Tidwell [57] also claims that it is beneficial to support both mouse and keyboard actions. Therefore, keyboard shortcuts have been added for all common tasks, using the same shortcut keys as the existing tools used for the sake of familiarity (See section 4.2.3).

If a 3GPP translation fails, because of incomplete resource files, an error message is displayed giving information on what needs to be updated in the resource files if possible. This kind of user guidance is supported by Sommerville [3] (See section 4.2.3).

Some design elements used in the GUI that are also supported by Tidwell [57] are: Prominent “done” buttons, smart menu items, auto-completion and new-item rows and many more. Using these design elements further improve the usability of the GUI.

4.3.2.2 Performance

As mentioned in the requirements section, 4.2.1, having an efficient implementation of the translation and filtering of the logs was crucial since the application should be able to handle log files of very large file sizes. Since a lot of the operations performed when translating and filtering are string operations, optimizing the handling of strings was important. The results from the performance pre-study (See section 4.2.4) were used when implementing to make sure that as high performance as possible could be gained.

The filtering of logs is done by checking for each row if it contains any of the active filter words. This results in a huge number of checks, making the efficiency of their implementation crucial. This was implemented by using the approach described in the pre-study, i.e. to only use ordinal string comparisons and only use case-insensitive comparisons when needed. By using ordinal comparisons culture-specific comparison rules are not handled, for the benefit of performance, but these kinds of comparisons are not relevant for this context.

The runtime’s internal string pool, also described in the performance pre-study, is used when translating logs, since keywords in the scripts are used many times for equality comparisons. This large number of times makes the approach very beneficial even though the constructing of the strings takes slightly more time than standard string declarations. This difference in construction time is insignificant compared to the decrease in loading time gained, and is also done at application startup only once.

4.3.2.3 3GPP Message Translation

As decided in the 3GPP Message Translation pre-study (See section 4.2.6), the scripts text file and the tables text file are used in Analyzer as resource files for 3GPP specifications. To keep the performance to a high level, the parsing of hexadecimal messages, via scripts, and the table lookups needed to be fast. To achieve this, both resource files are read and parsed by the application at startup and corresponding internal data structures are created. This requires a slightly longer loading time of the application, but the startup is still fast and the performance gains high.

Since the resource files are loaded automatically on startup, the replacement of them is very easy. Users can simply replace the text files in the application folders. A settings window for replacing, or relocating, any of the files is also present, which triggers a new parsing if needed. This allows users to have multiple files and switching back and forth between them in case different versions of the 3GPP specifications are needed. The 3GPP standards are also updated regularly, so replacing the files will be necessary. If the loaded files are corrupt or non-existing, the application can still be run, but a warning is given and the translation of 3GPP messages will not be done until the issue is solved.

The internal data structures created are a list of [Table](#) objects and a list of [Script](#) objects, both present in the [Analyzer._3GPPEntities](#) namespace (See Figure 17). The [Table](#) class contains an id and an array of rows, located at the array index corresponding with their row id for instant lookups. The [Script](#) class contains the script keyword it is identified with, a counter called `numberOfHits` and an instance of [BinaryScriptStepTree](#). [BinaryScriptStepTree](#) is a class representing a binary tree structure, having the class [BinaryScriptStepTreeNode](#) as node type. These nodes hold instances of the class [ScriptStep](#) as values, a class representing the step text rows in the script file. The tree structure is binary because each step has one action to perform if the current action succeeds and one action if it fails, as explained in section 4.2.5. The `numberOfHits` counter is a hit counter which is incremented every time a particular script's keyword is found in a log. The list of scripts is then sorted based on the scripts' counters so that the log lines are compared against the most common keywords first for performance.

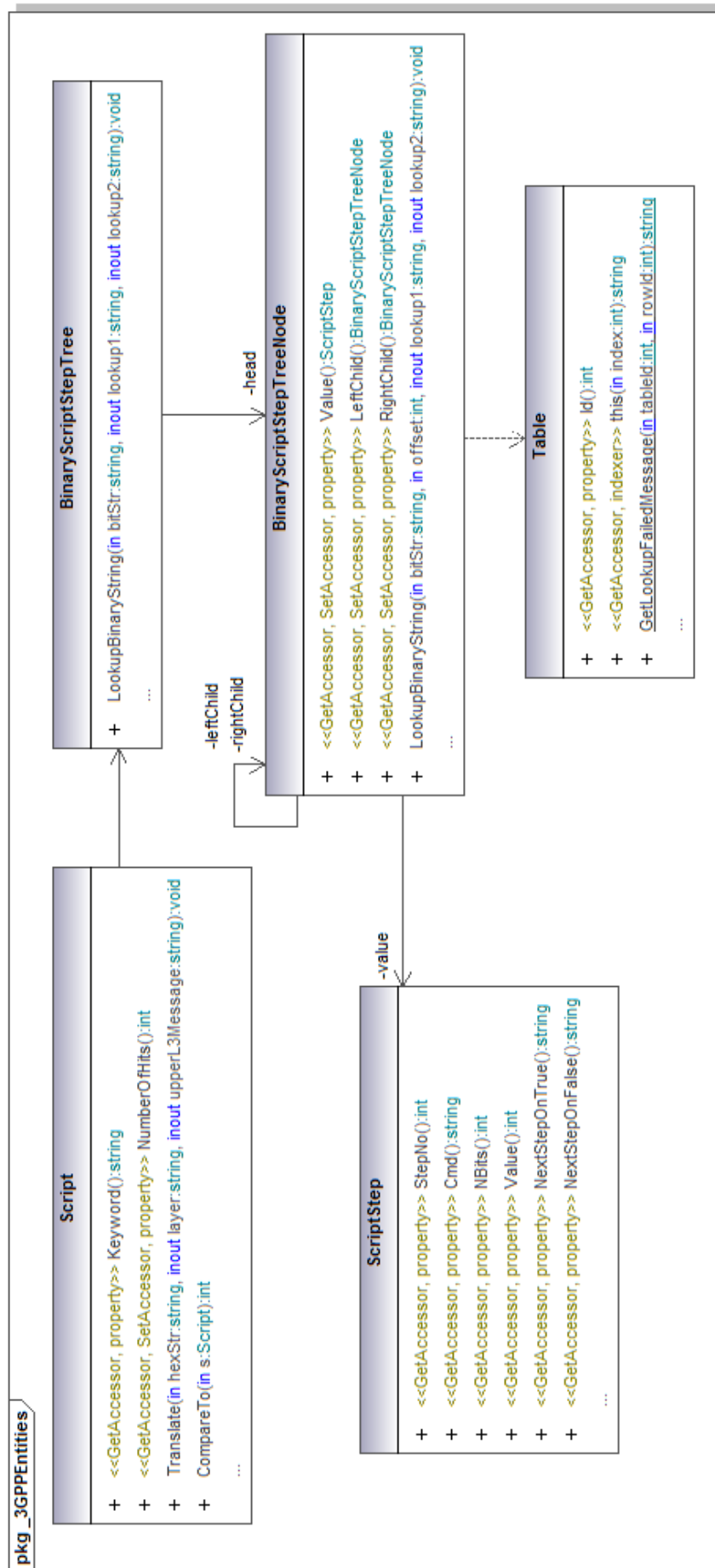


Figure 17. Simplified class diagram of the 3GPPEntities namespace

Having created the proper internal data structures, it is now possible to translate the 3GPP messages to a more readable format. Each line in the loaded logs is scanned for any of the keywords in the [Script](#) list, scanning in order of previous number of occurrences. If a keyword is found, that script's counter is increased and the following hexadecimal message is extracted. This hexadecimal message is then sent to its corresponding [Script](#) object, which translates it into a binary string, since the 3GPP script steps handle the hexadecimal messages bit-by-bit. The [Script](#) then starts the translation by sending the binary string to its [BinaryScriptStepTree](#) object. [BinaryScriptStepTree](#) then lets its nodes themselves handle the message and pass the message down the tree structure, by sending it to their left or right child depending on the result of their actions. When a leaf is finally reached, two table lookups will have been made on the way and two simplified messages corresponding to the initial hexadecimal message can be displayed. This workflow is illustrated in a sequence diagram for clarification (See Figure 18 and Figure 19).

This choice of data structures was seen as both intuitive, since the mapping is closely related to the resource files, and high performing in combination with the string optimization used (See section 4.3.2.2).

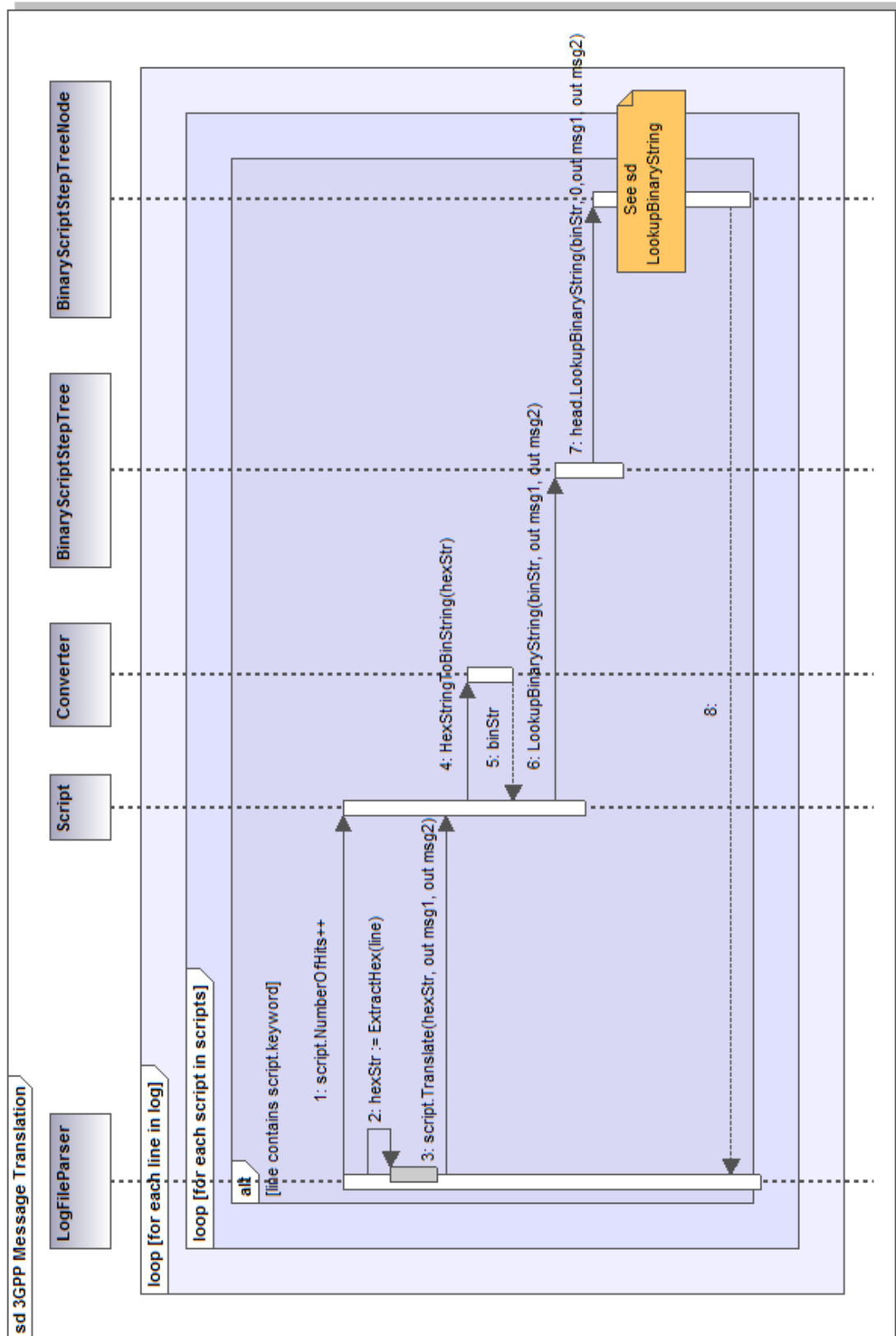


Figure 18. 3GPP Message Translation ("sd LookupBinaryString" is found in Figure 19)

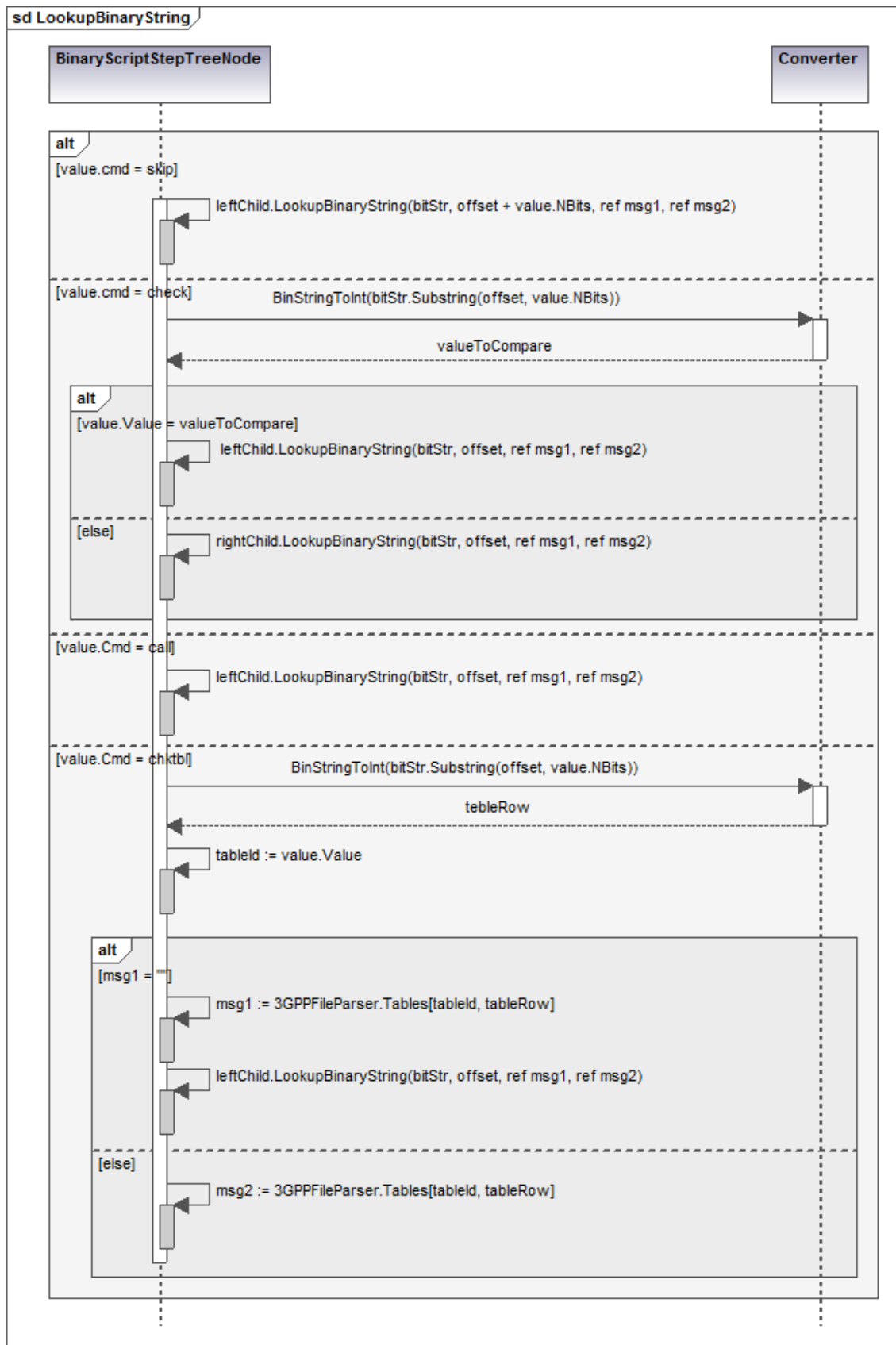


Figure 19. LookupBinaryString (Continuation of “sd 3GPP Message Translation”)
 (Note that the LookupBinaryString(...) calls are calls on the node’s children, re-iterating the flow illustrated in this diagram)

5 Results

The resulting systems of both parts of this thesis work were encapsulated into a new application suite, named EMBLA (Ericsson Mobile Broadband Logging Applications), consequently consisting of four applications. The possible interaction between the two new systems can be seen in Figure 20. Both systems fulfilled all of their individual requirements and some late additional features were even implemented. The results for both systems will now be presented.

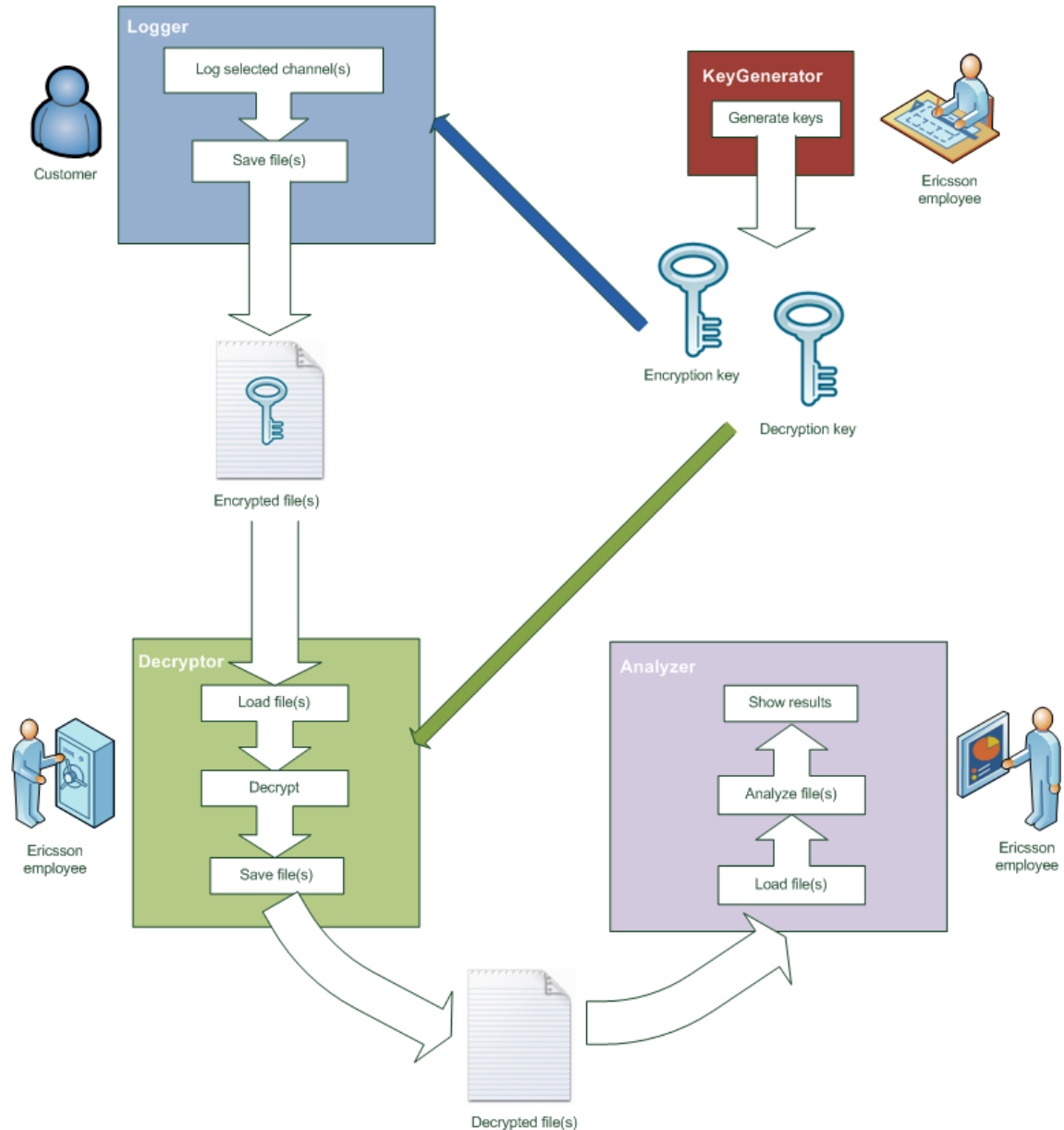


Figure 20. The possible interaction between the two systems

5.1 Log Collection

As already mentioned, all the requirements on the log collection system were fulfilled and implemented. The new system consists of three applications, called Logger, Decryptor and KeyGenerator. Logger is used to confidentially log data, Decryptor to decrypt the logs encrypted by Logger, and KeyGenerator to generate new sets of RSA keys.

The most important requirement, the required confidentiality, was fulfilled by using cryptography. Log data outputted from the modules, or collected from other data sources, is encrypted continuously, block-by-block, with AES, using a new key and IV for each logging session. The AES keys and IVs are then encrypted using MBM's public RSA key and appended to the encrypted log files. Employees can then decrypt the files, after receiving them from customers, using Decryptor. Decryptor decrypts the AES keys and IVs using MBM's private RSA key, and then uses these to decrypt the AES-encrypted log data into plaintext.

Both AES and RSA are known as unbreakable and by combining these, a system with high performance, high security, and with the benefit of only having the encryption key located in the application distributed to customers, Logger, and the decryption key in the application used at MBM, Decryptor, is realized.

Regarding the software quality, Logger is believed to have high modularity, which will facilitate the possible introduction of new data sources to log in the future. Decryptor and KeyGenerator both provide very little functionality, thus have very simple software architectures. All applications also have high error tolerance, which is important in this context.

No future work is planned on this system at the time of publication, but new data sources to log will probably be implemented in the future. Logger has been sent to a couple of customers, and so far, everything seems to work as desired.

5.2 Log Analysis

As with the log collection, all the requirements were fulfilled and implemented. The result is a new application called Analyzer. Analyzer translates all 3GPP messages, according to resource files, and even displays information when translations fail, making it easier for the employees to know what to update in the resource files. The application also supports filtering of logs and many other features.

Regarding the speed, Analyzer is a lot faster than all of the other tools used at MBM for the same purpose (See Table 3 for a speed comparison with Tool B). Analyzer is also believed to have reached high user-friendliness and flexibility. The system prototyping and the feedback received during the prototype workshops, in combination with theory on usability, helped in achieving this.

The software design of Analyzer is also believed to be of high quality, even though some focus was put on speed. The parts that are likely to be updated in the future provide the modularity and maintainability needed to do so.

Some future work regarding another type of filters, being more context sensitive, is planned, which should be easy to incorporate into the current software architecture. The application has also started to be used by the employees, with positive responses so far.

Test	Tool B	EMBLA Analyzer	Percentages faster
<ul style="list-style-type: none"> Log file of 174 MB (20090216_1449_Saga_R1B04_UL_HSUPA.txt) 	8:51min	29sec	95%
<ul style="list-style-type: none"> Log file of 174 MB (20090216_1449_Saga_R1B04_UL_HSUPA.txt) All of Tool B's 11 filters 	23:51min	8:09min	66%
<ul style="list-style-type: none"> Log file of 8 MB (20090203_1123_Saga_R1A25_PDP_active_2Gto3G.txt) 	46sec	2sec	96%
<ul style="list-style-type: none"> Log file of 8 MB (20090203_1123_Saga_R1A25_PDP_active_2Gto3G.txt) All of Tool B's 11 filters 	1:29min	20sec	78%
-	-	-	Avg: 84%

Table 3. Speed comparison between Analyzer and Tool B

6 Discussion

With the results in hand, we are very satisfied with our work and the outcomes of it. A lot has been learned and the work done has given us many interesting things to reflect upon. Some of these lessons learned and reflections, regarding the methodology used and the two systems, will now be described.

6.1 Methodology

Regarding the split of the thesis work into two distinct phases, we think that the right decision was made. While studying the log collection process, a lot was learnt about how people at the department work which helped us in the second phase which was highly dependent on the employees' ways of working. It was also positive to have a closure on the first phase, since software development of products can go on forever without final milestones or other kinds of closures.

Requirements engineering was something we learnt a lot from in general. Even though the rationales for the two systems were clear, specifying the requirements turned out to be quite difficult. It was not possible to freeze the requirements specification at the start of the project and end up with good systems, especially when high user-friendliness was required. To continuously refine and update the requirements seemed optimal, but required a lot of additional work. Even if software architectures themselves are flexible, the processes around them require flexibility. Updating release notes, help files, and re-testing the applications took a lot of resources. Not only did the applications need to be re-tested, late changes, when the systems were already in use, sometimes required verification using a formal internal process (See Appendix D for an example of a test specification). Despite this, we think that our flexible requirements process was the right choice, despite the work effort it required.

The testing of the applications turned out to be quite difficult, especially regarding Logger which has a large amount of parallel threads, creating a huge amount of possible timing scenarios. Using formal verification⁴⁵ for proving the correctness of Logger would have been an option, but creating such a verification model was unfortunately seen as far too time-consuming. Furthermore, the amount of testing, on a system level, possible to perform amongst us two developers was really small compared to the amount of testing done when the systems were to be in real use by employees. A log outputting framework, log4net [58], was therefore incorporated into all the applications to be able to trace the execution logs if employees would discover erroneous behavior. These logs were also of great use when testing our application on mobile broadband computers, not debugging from the Integrated Development Environment (IDE) otherwise used. This worked out very well and will hopefully help the department solve possible future errors found in the systems.

6.2 Log Collection

We are really satisfied with the results of the log collection system and its three applications. The cryptographic solution decided in the pre-study worked out very well in practice with help of the .NET framework's cryptographic libraries. We experienced some issues with the importing and exporting of cryptographic parameters in .NET and had trouble finding proper documentation, but the framework worked nicely in general and performed well.

⁴⁵ A method for proving correctness using mathematics

Despite the cryptographically safe solution used, one must not forget that 100% security cannot be reached and that logs are still possible to retrieve in some ways. When log data is sent from the broadband module, unencrypted, it must at some point in time be stored in the computer's volatile memory before it can be encrypted and made impossible to read by malicious users. We have tried hard to minimize the amount of log data stored in memory during logging, by encrypting with a block cipher in real-time, making these kinds of attack harder to perform. There are features provided by the .NET framework for protecting sensitive data in the runtime memory, but those features use a similar approach to the one implemented in the system. Using these redundant features would hurt performance and not contribute to the security since the data is still being received unencrypted from the module whether or not the data is protected when managed by the application. Having the physical module, it is also possible to retrieve log data directly via hardware connectors.

Another security issue is that the server that communicates with the firmware of the module is installed when Logger is installed. The API for communicating with the server is also part of the installation. Using the Tool A server for communicating with the module's firmware is not optimal for security, but the level of security achieved when using it was considered enough for this context.

Regarding the log collection, the decision to ourselves implement it for the other data sources than firmware is also a decision we are satisfied with. The server used to communicate with the firmware to be able to collect those logs also functioned very well. The API to communicate with it was well-documented and well-structured, providing the expected functionality. Despite this, there were some uncertainties regarding the underlying behavior of it. Timing issues when sending scripts to data sources' interactive channels where for example found. These kinds of issues had to be solved by work-around solutions such as using timers. Even though the server's functionality was very thoroughly documented, some of its behavior was still unclear, which gave us useful insight in how difficult and important it is to write and document APIs, which should not be disregarded when developing software. How to collect log data from WinAPI sources was actually not very well documented by Microsoft, but good online examples helped us understand the API and collect this data. The network data was quite easy to collect via the .NET framework, but the parsing of the network data turned out to be more cumbersome than expected. The network log data collected, the TCP and UDP packets, is formatted using the packet headers and corresponding values to a non-standardized format. The formatting of these logs could potentially be improved by using a well-known format, which could have allowed employees to analyze the files with existing network traffic analysis tools. The collection of the network log data is an area in the system where further development is possible, but this feature was seen as very low prioritized.

When designing the real-time encryption of log data, performance was important to ensure that large chunks of log data would not be temporarily stored in plaintext in the runtime memory before being encrypted. The implemented solution was heavily tested and was found to have the required performance, dependent on the output speed of log data from the data sources. It is important to remember though that the required performance might be higher in future versions of the broadband module due to larger amounts of log data being outputted at higher speeds. Also, adding more data sources, thereby increasing the possible amount of data sources users can log simultaneously, will increase the performance requirements. No testing

was unfortunately done to test the maximum performance capacity, but the application can handle the logging of all implemented data sources simultaneously without any problems.

6.3 Log Analysis

The log analysis system also has satisfying results. The performance of the application really exceeded our expectations, especially comparing to the currently used Tool B (See Table 3). What is noticeable is that implementation details, such as the ones regarding string optimization (See section 4.2.4), made the biggest difference in performance. Another huge performance boost, not documented in section 4, was achieved by simply turning off the automatic resize of columns for the .NET framework class `DataGridView` [47], the graphical component used to display logs. Letting the component automatically resize its columns was a huge performance bottleneck. This bottleneck could only be identified after testing, since no implementation details about it were available in the .NET framework's documentation. Instead, a custom column resize was implemented.

During the pre-study, tests were conducted to estimate how much performance could be gained by optimizing the string handling. It was at first difficult to identify any major differences in performance of the different approaches tested. It was then found that we needed to force runtime construction of the strings, by using for example console input, to avoid compiler optimizations. This showed the approaches' true performance, since most strings handled in the application are constructed at runtime from log files and resource files. This experience shows that trying to optimize performance of code can be difficult, especially when using a programming language where the code is compiled and then executed under a controlled environment, since the programmer's control over what is actually executed is limited. It is therefore important to always evaluate if any performance optimization really is beneficial.

Evaluating the performance of the system is difficult since it is difficult to judge what should be considered fast and what should be considered slow. To assess the performance of the system Tool B was used as a benchmark, since it is the most commonly used tool at MBM for log analysis. The developed system is a lot faster than Tool B, but an important consideration is that being faster than a poorly implemented system does not necessarily imply high performance. Since the source code of Tool B was not accessible, the quality of its implementation could not be evaluated. The log analysis system developed provides satisfying performance for its users, but some parts likely still exist where more optimizations can be made.

Using prototyping for designing the GUI was clearly beneficial when considering the feedback obtained and the effects it had on the final design. Without prototyping it would have been much harder to capture the users' opinions and preferences, most probably lowering the usability of the final product. Since usability was highly prioritized the extra effort required for prototyping was definitely worthwhile.

The problems with the requirements engineering mentioned in section 6.1 were predominant in this phase. It was hard, if not impossible, to please everyone, but we think that a good balance between the different preferences was found. Despite this, new tools always require, more or less, new ways of working and it will therefore probably take some time before the new system is fully adopted amongst the employees.

7 Acronyms and Abbreviations

2G	Second-generation
3G	Third-generation
3GPP	3rd Generation Partnership Project
AES	Advanced Encryption Standard
API	Application Programming Interface
CAPI	Cryptographic API
CBC	Cipher-Block Chaining
CE	Conformité Européenne
CLR	Common Language Runtime
CSD	Circuit Switched Data
D/L	Downlink
DES	Data Encryption Standard
DLL	Dynamic-Link Library
ECB	Electronic Codebook
EDGE	Enhanced Data rates for GSM Evolution
EMBLA	Ericsson Mobile Broadband Logging Applications
FCC	Federal Communications Commission
FIPS	Federal Information Processing Standard
FTP	File Transfer Protocol
GCF	Global Certification Forum
GnuPG	GNU Privacy Guard
GPG	GNU Privacy Guard
GPRS	General Packet Radio Service
GPS	Global Positioning System
GSM	Global System for Mobile communications
GUI	Graphical User Interface
HSPA	High Speed Packet Access
IDE	Integrated Development Environment
IOT	Interoperability Testing
IP	Internet Protocol
IrDA	Infrared Data Association
IV	Initialization Vector
Java SE	Java Standard Edition
JFC	Java Foundation Classes

JVM	Java Virtual Machine
MBM	Mobile Broadband Modules
Mbps	Megabit per second
NBS	National Bureau of Standards
NDA	Non-Disclosure Agreement
NIST	National Institute of Standards and Technology
OAEP	Optimal Asymmetric Encryption Padding
OpenPGP	Open Pretty Good Privacy
PC	Personal Computer
PCI	Peripheral Component Interconnect
PCMCIA	Personal Computer Memory Card International Association
PCS	Personal Communications Service
PRNG	Pseudorandom Number Generator
PTCRB	PCS Type Certification Review Board
R&TTE	The Radio and Telecommunications Terminal Equipment Directive
RAII	Resource Acquisition Is Initialization
RC4	Rivest Cipher 4
RoHS	Restriction of Hazardous Substances Directive
RSA	Rivest (Ron), Shamir (Adi), and Adleman (Leonard)
SIM card	Subscriber Identity Module card
SMS	Short Message Service
TCP	Transmission Control Protocol
U/L	Uplink
UDP	User Datagram Protocol
UI	User Interface
UMTS	Universal Mobile Telecommunications System
USB	Universal Serial Bus
VoIP	Voice Over IP
WCDMA	Wideband Code Division Multiple Access
WinAPI	Windows API
WinForms	Windows Forms
WPF	Windows Presentation Foundation
XML	Extensible Markup Language

8 References

- [1] Ericsson. (n.d.). Retrieved Mars 26, 2009, from Ericsson - Mobile Broadband Modules: <http://www.ericsson.com/solutions/page.asp?ArticleId=2EFE4BB4-E0FF-426E-8221-74CE8926A030>
- [2] Ericsson. (n.d.). Retrieved May 21, 2009, from Ericsson - Corporate Information: <http://www.ericsson.com/ericsson/corpinfo/index.shtml>
- [3] Sommerville, I. (2004). *Software Engineering (7th ed)*. Pearson Education Limited.
- [4] Schneier, B. (1996). *Applied Cryptography (2nd ed)*. John Wiley & Sons, Inc.
- [5] Partida, A., & Andina, D. (1999). Applied Cryptography in Java. *Proceedings of IEEE 33rd Annual International Carnahan Conference on Security Technology*, (pp. 345-348).
- [6] Shen, G., & Zheng, X. (2008). Java Cryptography Architecture and Security of Electronic Commerce. *Proceedings of the 4th International Conference on Wireless Communications, Networking and Mobile Computing*, (pp. 1-4).
- [7] Andrews, J. H., & Zhang, Y. (2000). Broad-spectrum Studies of Log File Analysis. *Proceedings of the 22nd International Conference on Software Engineering*, (pp. 105-114).
- [8] Vaarandi, R. (2003). A Data Clustering Algorithm For Mining Patterns From Event Logs. *Proceedings of the 3rd IEEE Workshop on IP Operations and Management*, (pp. 119-126).
- [9] Razavi, A., & Kontogiannis, K. (2008). Pattern and Policy Driven Log Analysis for Software Monitoring. *Proceedings of the 32nd Annual IEEE International Computer Software and Applications Conference*, (pp. 108-111).
- [10] Ohtaki, Y. (2005). Constructing a Searchable Encrypted Log using Encrypted Inverted Indexes. *Proceedings of the International Conference on Cyberworlds*, (pp. 130-138).
- [11] Ahlberg, C., & Shneiderman, B. (1994). Visual Information Seeking: Tight Coupling For Dynamic Query Filters with Starfield Displays. *Proceedings of the ACM Conference on Human Factors*, (pp. 313-317).
- [12] Roberts, J. C. (2000). Multiple-view and Multiform Visualization: Visual Data Exploration and Analysis VII. *Proceedings of SPIE*, (pp. 176-185).
- [13] Takada, T., & Koike, H. (2002). MieLog: A Highly Interactive Visual Log Browser Using Information Visualization and Statistical Analysis. *Proceedings of the 16th USENIX conference on System administration*, (pp. 133-144).
- [14] Ericsson. (2007). *F3507g Technical Description (1424_LZT1089714_01 - Rev PC1)* [confidential].
- [15] Ericsson. (2009, June 4). Retrieved June 5, 2009, from Mobile Broadband Modules Press Material: http://www.ericsson.com/solutions/mobile_broadband_modules/press.shtml

- [16] Engadget. (2009, March 31). Retrieved May 05, 2009, from Ericsson's F3607gw wake-on wireless HSPA module offers remote kill switch and recovery for laptops:
<http://www.engadget.com/2009/03/31/ericsson-outs-f3607gw-wake-on-wireless-hspa-gps-module-with-remo/>
- [17] Ericsson. (2007). Retrieved April 15, 2009, from Mobile Broadband Module F3507g:
http://www.ericsson.com/solutions/mobile_broadband_modules/docs/mobile_broadband_module_datasheet_print.pdf
- [18] Suiche, M., & Ruff, N. (2008, October 8). Retrieved April 10, 2009, from SandMan Project:
<http://sandman.msuiche.net/index.php>
- [19] Trappe, W., & Washington, L. (2006). *Introduction to Cryptography with Coding Theory (2nd ed.)*. Pearson Education International.
- [20] Smid, M. E., & Branstad, D. K. (1988). The Data Encryption Standard: Past and Future. *Proceedings of the IEEE Vol.76 (5)* , 550-559.
- [21] National Institute of Standards and Technology. (2006, May). Retrieved June 06, 2009, from Recommendation for Key Management - Part 1: General (Revised):
<http://csrc.nist.gov/publications/nistpubs/800-57/SP800-57-Part1.pdf>
- [22] National Institute of Standards and Technology. (2001, November 26). Retrieved May 20, 2009, from Announcing the Advanced Encryption Standard (AES):
<http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf>
- [23] Jamil, T. (2004). The Rijndael Algorithm. *IEEE Potentials Vol.23 (2)* , 36-38.
- [24] Liu, J., Chen, G., & Li, J. (2008). Revision of Security Proof on f-OAEP. *Proceedings of the International Conference on Information Security and Assurance*, (pp. 280-284).
- [25] RSA Laboratories. (2009). Retrieved June 09, 2009, from RSA Laboratories - 4.1.2.1 What key size should be used?: <http://www.rsa.com/rsalabs/node.asp?id=2264>
- [26] Ericsson Mobile Platforms. (2007). *Tool A User Guide (LZN 901 0345 R4C) [confidential]*.
- [27] Ericsson Mobile Platforms. (2007). *Tool A User Guide (LZN 901 0345 R4C) [confidential]*.
- [28] Russinovich, M. (2008, October 16). *DebugView for Windows*. Retrieved April 1, 2009, from Microsoft Technet: <http://technet.microsoft.com/en-us/sysinternals/bb896647.aspx>
- [29] Microsoft Corporation. (2009, June 9). Retrieved June 26, 2009, from Windows Sysinternals: Documentation, downloads and additional resources:
<http://technet.microsoft.com/en-us/sysinternals/default.aspx>
- [30] Wireshark Foundation. (n.d.). Retrieved April 1, 2009, from Wireshark: About:
<http://www.wireshark.org/about.html>
- [31] Krueger, C. W. (1992). Software Reuse. *ACM Computing Surveys Vol.24 (2)* , 131-183.
- [32] Dzung, D., Naedele, M., von Hoff, T. P., & Crevatin, M. (2005). Security for Industrial Communication Systems. *Proceedings of the IEEE Vol.93 (6)* , 1152-1177.

- [33] Ven, K., Verelst, J., & Mannaert, H. (2008, May-June). Should You Adopt Open Source Software? *Software, IEEE Vol.25 (3)* , 54-59.
- [34] Nguyen, P. Q. (2004). Can We Trust Cryptographic Software? Cryptographic Flaws in GNU Privacy Guard v1.2.3. *Proceedings of Eurocrypt '04*, (pp. 555-570).
- [35] Heng, C. (2008, December 20). *Free Encryption / Cryptographic Software*. Retrieved May 28, 2009, from <http://www.thefreecountry.com/security/encryption.shtml>
- [36] Bate, I. (2005). Dealing with Emergent Properties in Embedded Systems. *Proceedings of the 11th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, (pp. 63-66).
- [37] Sun Microsystems. (2009). Retrieved April 20, 2009, from Java SE Overview - at a glance: <http://java.sun.com/javase/>
- [38] Microsoft Corporation. (2008). Retrieved April 15, 2009, from Microsoft .NET Framework: <http://www.microsoft.com/.NET/>
- [39] Sun Microsystems. (2009). Retrieved April 14, 2009, from Java SE Security: <http://java.sun.com/javase/technologies/security/>
- [40] Sun Microsystems. (n.d.). Retrieved April 15, 2009, from Overview of Java SE Security: <http://java.sun.com/javase/6/docs/technotes/guides/security/overview/jsoverview.html>
- [41] Sun Microsystems. (2008, February 14). Retrieved June 03, 2009, from About the Java Technology: <http://java.sun.com/docs/books/tutorial/getStarted/intro/definition.html>
- [42] Sun Microsystems. (n.d.). Retrieved April 14, 2009, from Java Cryptography Architecture (JCA): <http://java.sun.com/javase/6/docs/technotes/guides/security/crypto/CryptoSpec.html>
- [43] Sun Microsystems. (2008). Retrieved June 03, 2009, from Java Platform, Standard Edition 6 API Specification: <http://java.sun.com/javase/6/docs/api/>
- [44] National Institute of Standards and Technology. (2001, May 25). Retrieved February 26, 2009, from FIPS PUB 140-2 - Security Requirements For Cryptographic Modules: <http://csrc.nist.gov/publications/fips/fips140-2/fips1402.pdf>
- [45] Microsoft Corporation. (2009). Retrieved June 01, 2009, from Microsoft Developer Network: .NET Framework Conceptual Overview: <http://msdn.microsoft.com/en-us/library/zw4w595w.aspx>
- [46] Microsoft Corporation. (2009). Retrieved June 02, 2009, from Microsoft Developer Network: Cryptography Overview: <http://msdn.microsoft.com/en-us/library/92f9ye3s.aspx>
- [47] Microsoft Corporation. (2009). Retrieved June 11, 2009, from Microsoft Developer Network: .NET Framework Class Library: <http://msdn.microsoft.com/en-us/library/ms229335.aspx>
- [48] Barker, E., & Kelsey, J. (2007, March). Retrieved June 2, 2009, from NIST Special Publication 800-90: Recommendation for Random Number Generation Using Deterministic

Random Bit Generators (Revised): http://csrc.nist.gov/publications/nistpubs/800-90/SP800-90revised_March2007.pdf

[49] Microsoft Corporation. (2009). Retrieved May 28, 2009, from Microsoft Developer Network: CryptGenRandom Function: [http://msdn.microsoft.com/en-us/library/aa379942\(VS.85\).aspx](http://msdn.microsoft.com/en-us/library/aa379942(VS.85).aspx)

[50] Microsoft Corporation. (2009). Retrieved June 2, 2009, from Microsoft WindowsClient.NET: Windows Forms, WPF - Get Started: <http://windowsclient.net/getstarted/>

[51] Pilipchouk, D. (2003, December 10). Retrieved June 4, 2009, from Java vs. .NET Security, Part 2 - Cryptography and Communication: <http://www.onjava.com/pub/a/onjava/2003/12/10/javavsdotnet.html>

[52] Mel, H. X., & Baker, D. M. (2000). *Cryptography Decrypted*. Addison-Wesley Professional.

[53] Francia, G. I., & Francia, R. R. (2007). An Empirical Study on the Performance of Java/.Net Cryptographic APIs. *Information Security Journal: A Global Perspective Vol.16 (6)* , 344-354.

[54] Microsoft Corporation. (2009). Retrieved Mars 15, 2009, from Microsoft Developer Network: Cross-Language Interoperability: <http://msdn.microsoft.com/en-us/library/730f1wy3.aspx>

[55] Blundell, J. K., Hines, M. L., & Stach, J. (1997). The Measurement of Software Design Quality. *Annals of Software Engineering Vol.4 (1)* , 235-255.

[56] Boehm, H.-J. (2003). Destructors, Finalizers, and Synchronization. *Proceedings of the 30th ACM SIGPLAN-SIGACT symposium on Principles of Programming Languages*, (pp. 262-272).

[57] Tidwell, J. (2006). *Designing Interfaces*. O'Reilly Media, Inc.

[58] Apache Software Foundation. (2007, August 30). Retrieved June 09, 2009, from Apache log4net: <http://logging.apache.org/log4net/index.html>

[59] Microsoft Corporation. (2009, May 7). Retrieved May 21, 2009, from Microsoft Developer Network: Windows API Reference (Windows): [http://msdn.microsoft.com/en-us/library/aa383749\(VS.85\).aspx](http://msdn.microsoft.com/en-us/library/aa383749(VS.85).aspx)

[60] Microsoft Corporation. (2009, May 20). Retrieved May 21, 2009, from Microsoft Developer Network: DbgPrint: <http://msdn.microsoft.com/en-us/library/ms792790.aspx>

[61] Various authors. (2009, June 22). Retrieved June 23, 2009, from Data logging - Wikipedia, the free encyclopedia: http://en.wikipedia.org/wiki/Data_logging

[62] Various authors. (2009, May 27). Retrieved May 29, 2009, from Data visualization - Wikipedia, the free encyclopedia: http://en.wikipedia.org/wiki/Data_visualization

[63] Various authors. (2009, May 27). Retrieved May 29, 2009, from Data visualization #Data analysis - Wikipedia, the free encyclopedia: http://en.wikipedia.org/wiki/Data_visualization#Data_analysis

Prepared (also subject responsible if other) EHALJER/EERISVE		No.		
Approved	Checked	Date 2009-05-18	Rev PA4	Reference

Appendix A. EMBLA Software Requirements Specification

Abstract: This document contains the software requirements for the implementation of EMBLA.

Prepared (also subject responsible if other) EHALJER/EERISVE		No.		
Approved	Checked	Date 2009-05-18	Rev PA4	Reference

Table of Contents

1 INTRODUCTION..... 3

1.1 Purpose..... 3

1.2 Scope 3

1.3 Abbreviations 3

1.4 References..... 3

1.5 Revision History 3

2 SYSTEM OVERVIEW 4

3 REQUIREMENTS 6

3.1 Operating environment requirements..... 6

3.1.1 EMBLA Logger 6

3.1.2 EMBLA Decryptor..... 6

3.1.3 EMBLA KeyGenerator 6

3.1.4 EMBLA Analyzer 6

3.2 Functional requirements..... 7

3.2.1 EMBLA Logger 7

3.2.2 EMBLA Decryptor..... 10

3.2.3 EMBLA KeyGenerator 12

3.2.4 EMBLA Analyzer 13

Prepared (also subject responsible if other) EHALJER/EERISVE		No.		
Approved	Checked	Date 2009-05-18	Rev PA4	Reference

1 Introduction

EMBLA (Ericsson Mobile Broadband Logging Applications) is a set of tools to be developed as a part of a Master Thesis project, with the goal of facilitating customer support. This will be done by introducing a system which allows customers to safely log confidential information and provides the modularity needed to gather and analyze the different kinds of data needed.

1.1 Purpose

This document shall form a base for the design and implementation of the product. It is not intended for the customer or the user of the product.

1.2 Scope

This document will cover the system requirements at a functional level.

1.3 Abbreviations

EMBLA Ericsson Mobile Broadband Logging Applications

1.4 References

-

1.5 Revision History

Revision	Date	Prepared	Changes made
PA1	2009-02-25	EHALJER/EERISVE	Created
PA2	2009-03-26	EHALJER/EERISVE	Updated
PA3	2009-04-24	EHALJER/EERISVE	Updated
PA4	2009-05-18	EHALJER/EERISVE	Updated

Prepared (also subject responsible if other) EHALJER/EERISVE		No.		
Approved	Checked	Date 2009-05-18	Rev PA4	Reference

2 System overview

This section contains an overview of the system. The applications included in EMBLA are:

- **EMBLA Logger**
Logs data confidentially and saves the output as encrypted files. (used by customers/operators)
- **EMBLA Decryptor**
Decrypts logs created by EMBLA Logger and saves the outputs as plaintext files. (used by MBM Support)
- **EMBLA KeyGenerator**
Generates the files necessary for EMBLA Logger & Decryptor to encrypt/decrypt. (used by MBM Support)
- **EMBLA Analyzer**
Opens log files, for example decrypted by EMBLA Decryptor, and displays them and allows the user to easier analyze the logs. (used by MBM Support)

The following figure (See Figure 1) illustrates the interaction between the applications in EMBLA:

Prepared (also subject responsible if other) EHALJER/EERISVE		No.		
Approved	Checked	Date 2009-05-18	Rev PA4	Reference

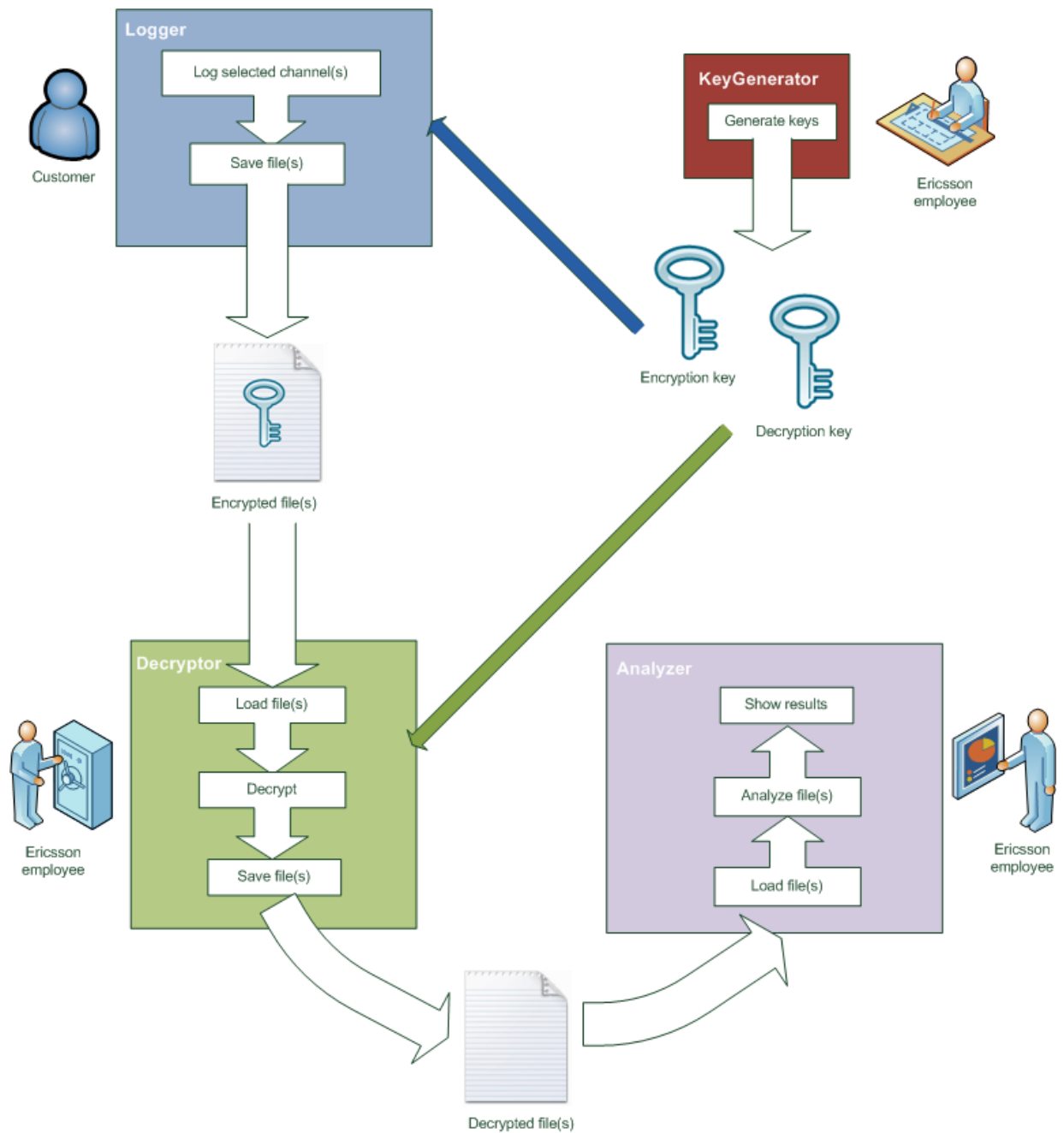


Figure 1. System Overview

Prepared (also subject responsible if other) EHALJER/EERISVE		No.		
Approved	Checked	Date 2009-05-18	Rev PA4	Reference

3 Requirements

This section describes the operating environment requirements for the EMBLA applications and the functional requirements for each of them. The functional requirements are also classified as mandatory, desirable or optional.

3.1 Operating environment requirements

The section contains the operating environments required to run the applications. Existing performance attributes are omitted.

3.1.1 EMBLA Logger

- 3.1.1.1 Microsoft Windows XP, Microsoft Windows Vista or Microsoft Windows 7
- 3.1.1.2 Microsoft .Net version 2.0 or above
- 3.1.1.3 Double the log files' size as free space on the device from which the application is run

3.1.2 EMBLA Decryptor

- 3.1.2.1 Microsoft Windows XP, Microsoft Windows Vista or Microsoft Windows 7
- 3.1.2.2 Microsoft .Net version 2.0 or above
- 3.1.2.3 Double the log files' size as free space on the device from which the application is run

3.1.3 EMBLA KeyGenerator

- 3.1.3.1 Microsoft Windows XP, Microsoft Windows Vista or Microsoft Windows 7
- 3.1.3.2 Microsoft .Net version 2.0 or above

3.1.4 EMBLA Analyzer

- 3.1.4.1 Microsoft Windows XP, Microsoft Windows Vista or Microsoft Windows 7
- 3.1.4.2 Microsoft .Net version 3.5 or above
- 3.1.4.3 At least the log files' size as free RAM memory

Prepared (also subject responsible if other) EHALJER/EERISVE		No.		
Approved	Checked	Date 2009-05-18	Rev PA4	Reference

3.2 Functional requirements

This section states the functional requirements for the EMBLA applications together with a description and a prioritization classification.

3.2.1 EMBLA Logger

3.2.1.1 Select what to log

The user should be able to select what type of logging that will be performed.

CLASSIFICATION: Mandatory

3.2.1.2 Start logging

The user should be able to trigger the logging to start.

CLASSIFICATION: Mandatory

3.2.1.3 Stop Logging

The user should be able to trigger the logging to stop.

CLASSIFICATION: Mandatory

3.2.1.4 Log firmware activity

The user should be able to log activity in firmware.

CLASSIFICATION: Mandatory

3.2.1.5 Log WinAPI-debug messages

The user should be able to log WinAPI-debug messages.

CLASSIFICATION: Mandatory

3.2.1.6 The logs should not be readable by anyone else than Ericsson

The log files should not be readable without decrypting them with EMBLA Decryptor.

CLASSIFICATION: Mandatory

3.2.1.7 The logs should be unreadable even during logging

The log data should be made unreadable in real-time to avoid leakage of data.

CLASSIFICATION: Mandatory

Prepared (also subject responsible if other) EHALJER/EERISVE		No.		
Approved	Checked	Date 2009-05-18	Rev PA4	Reference

3.2.1.8 **Save logs to files**

The user should be able to save logs to files.

CLASSIFICATION: Mandatory

3.2.1.9 **Log time stamps**

The user should be able to log time stamps.

CLASSIFICATION: Mandatory

3.2.1.10 **COM-port selectable**

The user should be able to select which COM-port to use when communicating with the module.

CLASSIFICATION: Mandatory

3.2.1.11 **Load scripts for interactive debug of firmware**

The user should be able to load scripts used for interactive debug of firmware

CLASSIFICATION: Desirable

3.2.1.12 **Interactively add markers to a log**

The user should be able to place markers in logs during logging.

CLASSIFICATION: Desirable

3.2.1.13 **Show feedback during logging**

The user should be given feedback during logging to make visible that data is really being logged.

CLASSIFICATION: Desirable

3.2.1.14 **Simultaneous logging of multiple data sources**

The user should be able to log multiple data sources simultaneously.

CLASSIFICATION: Desirable

3.2.1.15 **Logging is not interrupted by module crash**

The logging session should not be aborted if a module crashes or is disconnected

CLASSIFICATION: Desirable

Prepared (also subject responsible if other) EHALJER/EERISVE		No.		
Approved	Checked	Date 2009-05-18	Rev PA4	Reference

3.2.1.16 Log module startup

The user should be able to log the startup of a module

CLASSIFICATION: Desirable

3.2.1.17 Change the input key filename

The user should be able to specify the filename of the input key to use for encryption.

CLASSIFICATION: Optional

Prepared (also subject responsible if other) EHALJER/EERISVE		No.		
Approved	Checked	Date 2009-05-18	Rev PA4	Reference

3.2.2 EMBLA Decryptor

3.2.2.1 Load encrypted file

The user should be able to load encrypted files created by EMBLA Logger.

CLASSIFICATION: Mandatory

3.2.2.2 Load multiple encrypted files

The user should be able to load multiple encrypted files created by EMBLA Logger.

CLASSIFICATION: Desirable

3.2.2.3 Load encrypted files by drag-n-drop

The user should be able to load multiple encrypted files created by EMBLA Logger by drag-n-dropping them onto the GUI.

CLASSIFICATION: Optional

3.2.2.4 Decrypt logs

The user should be able to decrypt loaded files.

CLASSIFICATION: Mandatory

3.2.2.5 Decrypt multiple logs

The user should be able to decrypts multiple loaded files at once.

CLASSIFICATION: Desirable

3.2.2.6 Save decrypted logs

The user should be able to save a decrypted log as a plaintext file.

CLASSIFICATION: Mandatory

3.2.2.7 Save multiple decrypted logs

The user should be able to save multiple decrypted logs as plaintext files.

CLASSIFICATION: Desirable

3.2.2.8 Change the input key filename

The user should be able to specify the filename of the input key to use for decryption.

Prepared (also subject responsible if other) EHALJER/EERISVE		No.		
Approved	Checked	Date 2009-05-18	Rev PA4	Reference

CLASSIFICATION: Optional

Prepared (also subject responsible if other) EHALJER/EERISVE		No.		
Approved	Checked	Date 2009-05-18	Rev PA4	Reference

3.2.3 EMBLA KeyGenerator

3.2.3.1 Generate files necessary to encrypt/decrypt

The user should be able to generate files that can be used by EMBLA Logger and EMBLA Decryptor for for encryption/decryption.

CLASSIFICATION: Mandatory

3.2.3.2 Change the output filenames

The user should be able to specify the filenames of the generated keys.

CLASSIFICATION: Optional

3.2.3.3 Change the output directory

The user should be able to specify the directory to where the generated keys are saved.

CLASSIFICATION: Optional

Prepared (also subject responsible if other) EHALJER/EERISVE		No.		
Approved	Checked	Date 2009-05-18	Rev PA4	Reference

3.2.4 EMBLA Analyzer

3.2.4.1 Open log files

The user should be able to open log files.

CLASSIFICATION: Mandatory

3.2.4.2 Translate 2G and 3G messages

The application should translate 2G and 3G messages in logs according to 3GPP standards.

CLASSIFICATION: Mandatory

3.2.4.3 Filter logs

The user should be able to filter desired messages when displaying logs.

CLASSIFICATION: Mandatory

3.2.4.4 Choose active filters

The user should be able to choose which filters that should be active.

CLASSIFICATION: Mandatory

3.2.4.5 Edit filters

The user should be able to edit filters.

CLASSIFICATION: Desirable

3.2.4.6 Combine filters

The user should be able to combine different filters in one view.

CLASSIFICATION: Optional

3.2.4.7 Intelligent filtering

The application should be able to filter logs by using analyzing sequences of messages for patterns.

CLASSIFICATION: Optional

3.2.4.8 Customizable user interface

The user should be able to customize the color scheme used by the application.

Prepared (also subject responsible if other) EHALJER/EERISVE		No.		
Approved	Checked	Date 2009-05-18	Rev PA4	Reference

CLASSIFICATION: Desirable

3.2.4.9 **Mark lines in logs**

The user should be able to mark lines in logs and show them in a separate view.

CLASSIFICATION: Desirable

3.2.4.10 **Search**

The user should be able to search for lines in logs.

CLASSIFICATION: Mandatory

3.2.4.11 **Set external files' paths**

The user should be able to set the path to external files used by the application.

CLASSIFICATION: Desirable

3.2.4.12 **Save external files' paths**

The paths to external files set by the user should be saved to file.

CLASSIFICATION: Desirable

3.2.4.13 **Save color scheme settings**

The color scheme settings should be saved to file.

CLASSIFICATION: Desirable

3.2.4.14 **Import filers**

The user should be able to import filters from text files.

CLASSIFICATION: Mandatory

3.2.4.15 **Create new filters**

The user should be able to create new filters.

CLASSIFICATION: Desirable

3.2.4.16 **Delete filters**

The user should be able to delete filters.

CLASSIFICATION: Desirable

Prepared (also subject responsible if other) EHALJER/EERISVE		No.		
Approved	Checked	Date 2009-05-18	Rev PA4	Reference

3.2.4.17 **Export filters**

The user should be able to export filters to text files.

CLASSIFICATION: Optional

3.2.4.18 **Save filters**

The filters should be saved to file by the application.

CLASSIFICATION: Desirable

3.2.4.19 **Open files with tailing**

The application should be able to display, translate and filter new contents of a log file when it is changed.

CLASSIFICATION: Desirable

3.2.4.20 **Choose type of messages to translate**

The user should be able to choose if 2G messages, 3G messages or both should be translated by the application.

CLASSIFICATION: Optional

3.2.4.21 **Open 2G and 3G messages with external message application**

The user should be able to open 2G and 3G messages with an external message viewer.

CLASSIFICATION: Desirable

3.2.4.22 **Copy**

The user should be able to copy lines from the application to the Microsoft Windows clipboard.

CLASSIFICATION: Desirable

Prepared (also subject responsible if other) EHALJER/EERISVE		No.		
Approved	Checked	Date 2009-05-18	Rev PA4	Reference

3.2.4.23 **Paste**

The user should be able to paste log lines from the Microsoft Windows clipboard.

CLASSIFICATION: Desirable

3.2.4.24 **Cut**

The user should be able to cut log lines from the application adding them to the Microsoft Windows clipboard.

CLASSIFICATION: Optional

3.2.4.25 **Go to corresponding lines in other active views**

The user should be able to go from a line in one log view to corresponding line in other open log views.

CLASSIFICATION: Mandatory

3.2.4.26 **Auto scroll**

The user should be able choose auto scroll to automatically show lines added to the log when tailing is active.

CLASSIFICATION: Desirable

3.2.4.27 **Go to a specified line**

The user should be able to go to a specified line number in all logs opened log views.

CLASSIFICATION: Mandatory

3.2.4.28 **Choose visible views**

The user should be able to choose which different views that should be displayed.

CLASSIFICATION: Desirable

Prepared (also subject responsible if other) EHALJER/EERISVE		No.		
Approved	Checked	Date 2009-05-18	Rev PA2	Reference

Appendix B. EMBLA Administrator Manual

Abstract: This document contains the administrator manual for the management of EMBLA.

Prepared (also subject responsible if other) EHALJER/EERISVE		No.		
Approved	Checked	Date 2009-05-18	Rev PA2	Reference

1	INTRODUCTION.....	3
1.1	Purpose.....	3
1.2	Scope	3
1.3	Abbreviations	3
1.4	References.....	3
1.5	Revision History	3
2	SYSTEM OVERVIEW	4
3	DISTRIBUTION.....	6
3.1	EMBLA Logger	6
3.2	EMBLA Decryptor.....	6
3.3	EMBLA KeyGenerator	6
3.4	EMBLA Analyzer	6
4	KEY MANAGEMENT	6
5	3GPP RESOURCE FILES MANAGEMENT	7

Prepared (also subject responsible if other) EHALJER/EERISVE		No.		
Approved	Checked	Date 2009-05-18	Rev PA2	Reference

1 Introduction

EMBLA (Ericsson Mobile Broadband Logging Applications) is a set of tools developed, as a part of a Master Thesis project, with the goal of facilitating customer support. This is done by introducing a system which allows operators to safely log confidential information and provides the modularity needed to gather and analyze the different kinds of data needed.

1.1 Purpose

This document shall form a base for the management of the system when in use. It is not intended for the customers or the users of the product, or for maintenance of software source code.

1.2 Scope

This document will give an overview of the system and explain how the system is intended to be managed.

1.3 Abbreviations

EMBLA	Ericsson Mobile Broadband Logging Applications
-------	--

1.4 References

-

1.5 Revision History

Revision	Date	Prepared	Changes made
PA1	2009-03-26	EHALJER/EERISVE	Created
PA2	2009-05-18	EHALJER/EERISVE	Updated
PA3	2009-08-06	EHALJER/EERISVE	Updated

Prepared (also subject responsible if other) EHALJER/EERISVE		No.		
Approved	Checked	Date 2009-05-18	Rev PA2	Reference

2 System overview

This section contains an overview of the system. The applications included in EMBLA are:

- **EMBLA Logger**
Logs data confidentially and saves the output as encrypted files. (used by customers/operators)
- **EMBLA Decryptor**
Decrypts logs created by EMBLA Logger and saves the outputs as plaintext files. (used by MBM Support)
- **EMBLA KeyGenerator**
Generates the files necessary for EMBLA Logger & Decryptor to encrypt/decrypt. (used by MBM Support)
- **EMBLA Analyzer**
Opens log files, for example decrypted by EMBLA Decryptor, and displays them and allows the user to easier analyze the logs. (used by MBM Support)

The following figure (See Figure 1) illustrates the interaction between the applications in EMBLA:

Prepared (also subject responsible if other) EHALJER/EERISVE		No.		
Approved	Checked	Date 2009-05-18	Rev PA2	Reference

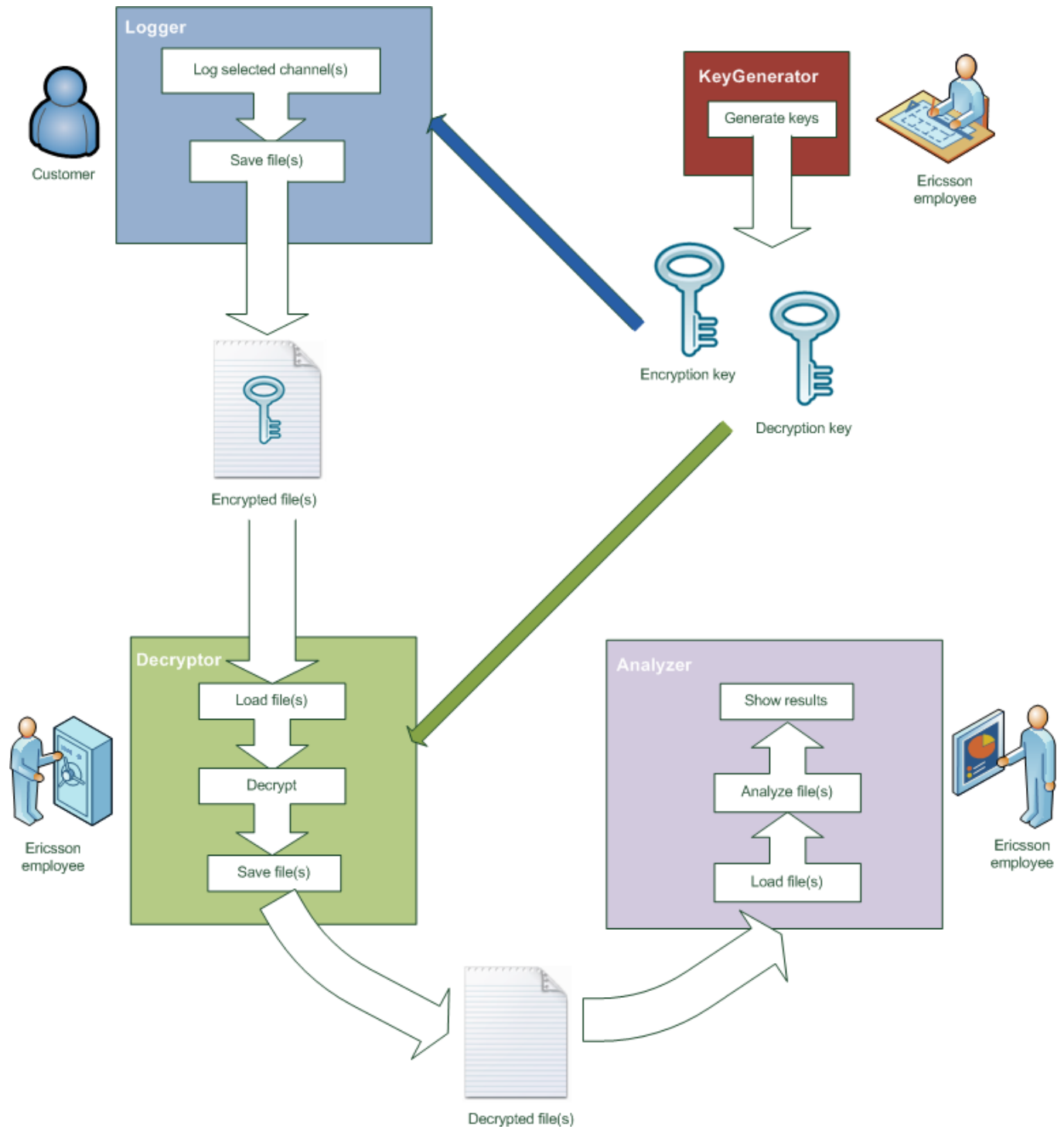


Figure 1: System Overview

Prepared (also subject responsible if other) EHALJER/EERISVE		No.		
Approved	Checked	Date 2009-05-18	Rev PA2	Reference

3 Distribution

This section describes how the different applications should be distributed.

3.1 EMBLA Logger

EMBLA Logger should be used by customers and operators and is installed by running the installation file 'EMBLA Logger Setup.exe'. The application is then run by starting 'Logger.exe' or any of the shortcuts placed on the desktop and in the start menu.

3.2 EMBLA Decryptor

EMBLA Decryptor should be used by MBM and is deployed by extracting the file 'EMBLA Decryptor.zip'. The application is then run by starting 'Decryptor.exe'.

3.3 EMBLA KeyGenerator

EMBLA KeyGenerator should be used by MBM and is deployed by extracting the file 'EMBLA KeyGenerator.zip'. The application is then run by starting 'KeyGenerator.exe'.

3.4 EMBLA Analyzer

EMBLA Analyzer should be used by MBM and is installed by running the installation file 'EMBLA Logger Analyzer.exe'. The application is then run by starting 'Analyzer.exe' or any of the shortcuts placed on the desktop and in the start menu.

4 Key management

The manageable cryptography mechanism in the EMBLA system uses so called asymmetric keys, which means that the key used for decryption is not the same as the one used for encryption. These two keys must be generated as a pair though to work together mathematically. The encryption keys in asymmetric key cryptography are called public keys, and the decryption keys are called private keys. EMBLA KeyGenerator generates such a key pair.

An encryption key (public key) is included in the EMBLA Logger executable when the application is built and its corresponding decryption key (private key) is included in the EMBLA Decryptor folder. Therefore, EMBLA KeyGenerator is not needed unless the private key has been leaked or lost. In this case, it is just a matter of generating a new pair and replacing the key files in the build projects' Config-folders (EMBLA Logger, used by customers/operators, must only have the public key).

Prepared (also subject responsible if other) EHALJER/EERISVE		No.		
Approved	Checked	Date 2009-05-18	Rev PA2	Reference

5 3GPP resource files management

Two kinds of resource files are used by EMBLA Analyzer to translate network log messages into a more readable format. One file contains so called scripts (See Figure 2) that specify how to get translation table lookup values from the hexadecimal part of the log messages.

```
[KEY WORD] <UL CCCH:>
0   CHECK   1   1   1   2
1   SKIP    36  0   NEXT  INVALID
2   SKIP    1   0   NEXT  INVALID
3   CHKTBL  1  13  NEXT  INVALID
4   CHKTBL  2   2   END   INVALID
[SCRIPT END]
```

Figure 2. A script

The found table lookup values are then used to lookup the translation in the other resource file containing the translation tables (See Figure 3).

```
[TABLE ID] 2
0   CELL UPDATE
1   RRC CONNECTION REQUEST
2   URA UPDATE
[TABLE END]
```

Figure 3. A table

These files need to be updated in order to correctly translate all network messages found in logs. EMBLA Analyzer handles if no lookup values are found and if table lookup values are found, but the actual table lookup fails, the displayed translation message indicates which table id/row id is missing.

Which resource files to use is changeable graphically in the EMBLA Analyzer settings window.

Prepared (also subject responsible if other) EHALJER/EERISVE		No.		
Approved	Checked	Date 2009-05-18	Rev PA2	Reference

Appendix C. EMBLA Logger Customer Manual

Abstract: This document contains the customer manual for EMBLA
Logger.

Prepared (also subject responsible if other) EHALJER/EERISVE		No.		
Approved	Checked	Date 2009-05-18	Rev PA2	Reference

1	INTRODUCTION.....	3
1.1	Purpose.....	3
1.2	Scope	3
1.3	Abbreviations	3
1.4	References.....	3
1.5	Revision History	3
2	EMBLA	4
2.1	System overview	4
3	EMBLA LOGGER	5
3.1	Product overview.....	5
3.1.1	Features	5
3.1.2	System Requirements	5
3.1.3	Quick guides.....	6
3.1.4	GUI Symbols	6
3.1.5	FAQ.....	7
3.1.6	Error reporting	8

Prepared (also subject responsible if other) EHALJER/EERISVE		No.		
Approved	Checked	Date 2009-05-18	Rev PA2	Reference

1 Introduction

EMBLA (Ericsson Mobile Broadband Logging Applications) is a set of tools developed at Ericsson Mobile Broadband Modules, with the goal of facilitating customer support. EMBLA Logger is the application used by customers and operators to log data confidentially.

1.1 Purpose

This document shall form a base for the customer usage of EMBLA Logger. It is intended for the customers and the users of the product.

1.2 Scope

This document will give an overview of the system and the application and explain how the application is intended to be used.

1.3 Abbreviations

EMBLA Ericsson Mobile Broadband Logging Applications

1.4 References

-

1.5 Revision History

Revision	Date	Prepared	Changes made
PA1	2009-04-24	EHALJER/EERISVE	Created
PA2	2009-05-18	EHALJER/EERISVE	Updated

Prepared (also subject responsible if other) EHALJER/EERISVE		No.		
Approved	Checked	Date 2009-05-18	Rev PA2	Reference

2 EMBLA

2.1 System overview

The following figure (See Figure 1) illustrates the interaction between EMBLA Logger and the other applications in EMBLA.

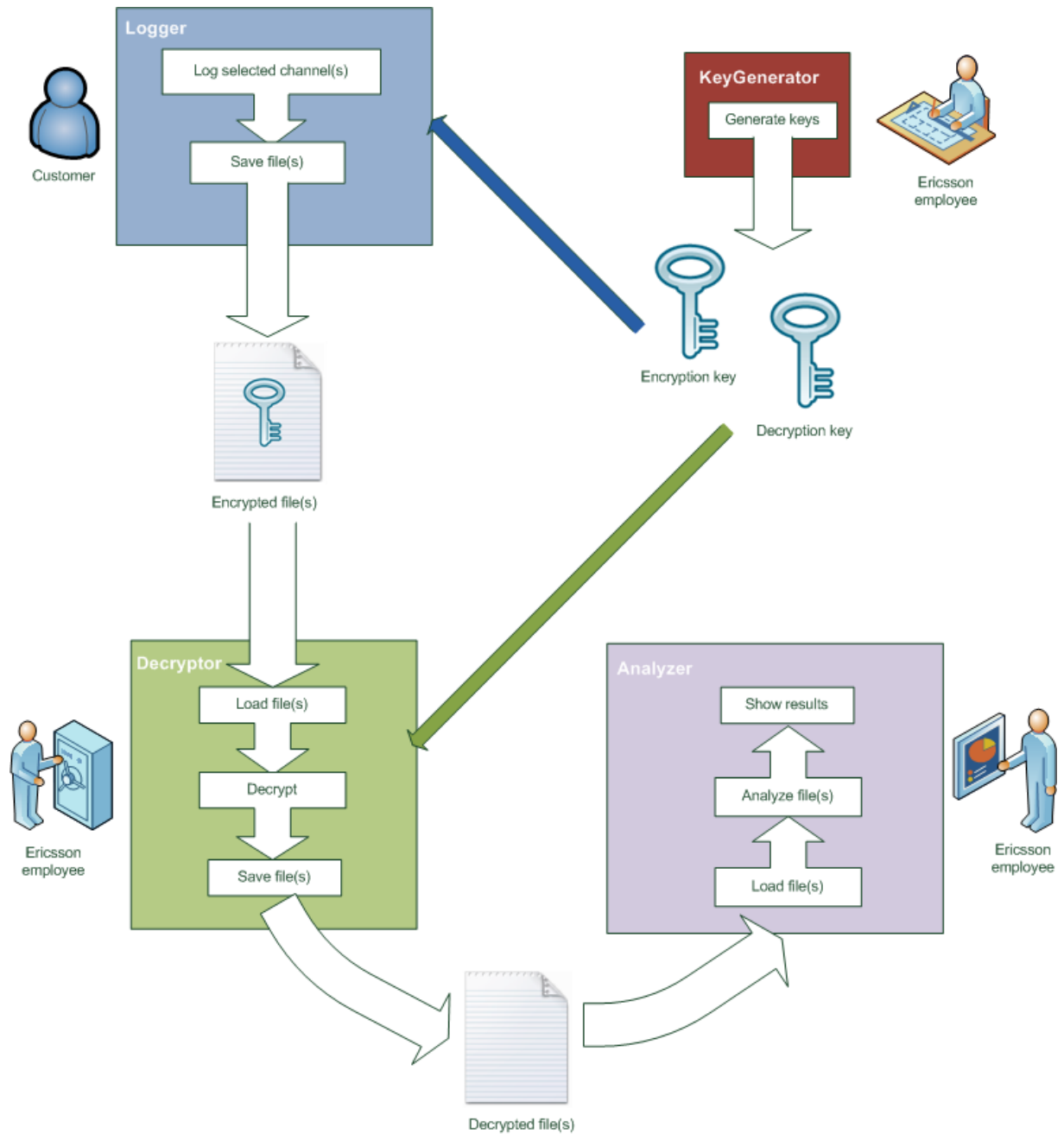


Figure 1: System Overview

Prepared (also subject responsible if other) EHALJER/EERISVE		No.		
Approved	Checked	Date 2009-05-18	Rev PA2	Reference

3 EMBLA Logger

3.1 Product overview

EMBLA Logger logs data confidentially and saves the output as encrypted files. The outputted encrypted files are then sent to Ericsson for analysis and support.

3.1.1 Features

- Logs firmware activity
- Logs Windows debug messages from OutputDebugString() in WinAPI
- Logs network traffic (TCP and UDP packets)
- Logging is not stopped even in the case of a module crash. Instead, a warning is given
- Encrypts log data "in real-time"
- Is not restricted by large log file sizes since only small chunks are saved in memory
- Decrypted log data is never exposed in large chunks even in memory
- Can log different sources simultaneously
- Which COM-port to connect to is selectable
- Can reset specific module
- Can automatically start logging selected channels after reset
- Can also send reset command while not connected to a COM-port, which sometimes solves the 'no carrier', 'searching for device' and 'configuration conflict'-problems
- Can add markers while logging
- Possible to load scripts to 'interactive channels' to enable extra output
- Shows statistics while logging - data logged, elapsed time, number of markers added, number of connections

3.1.2 System Requirements

- Windows XP, Windows Vista, Windows 7
- .NET version 2.0.50727 or above
- At least double the log files' size as free space on the device from which the application is run

Prepared (also subject responsible if other) EHALJER/EERISVE		No.		
Approved	Checked	Date 2009-05-18	Rev PA2	Reference

3.1.3 Quick guides

Standard logging:











Choose which channels to log by either double clicking on them or by right clicking on them and choosing 'Select'. Click the play button to start logging. At least one channel must be selected to be able to start.

Automatically start logging after module reset:

Choose which channels that should automatically be logged after a module reset. This is done by right clicking on loggable nodes and choosing 'Auto-log after module reset'. The actual reset is then done by right clicking on the node representing the broadband module and choosing 'Reset'. The application will then automatically start logging the chosen channels as soon as they are available.

3.1.4 GUI Symbols

Icons:

-  The default icon for a node
-  Indicates that the node is loggable but not currently selected for logging
-  Indicates that the node is chosen for logging
-  Indicates that the node is chosen for automatic logging after module reset
-  Indicates that the node is chosen for logging and also chosen for automatic logging after module reset
-  Indicates that the node is configurable. This option is accessed by right clicking on the node and choosing 'Configure'
-  Indicates that scripts can be sent to the node. Scripts are sent by right clicking on the node and choosing 'Load script'
-  Indicates that scripts can be sent to the node and that some script(s) are already loaded
-  Represents a module and indicates that the node allows reset
-  Indication for the force reset function in the treeview context menu

Prepared (also subject responsible if other) EHALJER/EERISVE		No.		
Approved	Checked	Date 2009-05-18	Rev PA2	Reference

Buttons:



Adds markers to the logs during logging



Adding markers to the logs is currently not possible



Starts logging the selected nodes



Start logging is currently not possible



Stops logging



Stop logging is currently not possible

3.1.5

FAQ

Symptom

Suggested action

No nodes are shown under 'Firmware'

Make sure that the application is connected to the broadband module by right clicking 'Firmware' and choosing 'Configure'

A connection cannot be established to the module in the firmware configuration window, i.e. only 'configuration conflict', 'no carrier' or 'searching for device' is shown as status

Right clicking 'Firmware' and choosing 'Force Reset' sometimes solves this issue

Prepared (also subject responsible if other) EHALJER/EERISVE		No.		
Approved	Checked	Date 2009-05-18	Rev PA2	Reference

3.1.6 Error reporting

If an error is found, please send the file 'ApplicationLog.txt' found in the application folder, together with an error description to mbmsupport@ericsson.com. Note that this file is overwritten at each run, so please copy this file before re-starting the application.

Known issues:

- Application sometimes malfunctions after sleep (S3) and hibernate (S4)
- Unexpected behavior sometimes occur when using this application at the same time as another application also using the Tool A server, such as the Tool A logger
- Uninstalling the Tool A logger causes the Tool A server to be unregistered from the system. This prevents this application from starting and uninstalling properly. To be able to uninstall, first reinstall the Tool A logger
- The name of the module can sometimes be different after a reset
- It is not possible to send reset commands through the Data Modem port

Appendix D. EMBLA LOGGER TEST SPECIFICATION

Product	EMBLA Logger
Date	
Name of tester	
Release	
Computer brand	
Operating system	
Location	IOV/MBM LINDHOLMEN

If an error is found, please attach the file 'ApplicationLog.txt' found in the application folder.

Note that this file is overwritten at each run, so please copy this file before re-starting the application.

Test no	Priority class	Area	Category	Test	Action	Expected result	Tools, etc.	Pass / Fail / NT	Notes, Comments, and Log File Names
---------	----------------	------	----------	------	--------	-----------------	-------------	------------------	-------------------------------------

0 - Preparation

0.001	Full	Pre	All	Ghost Computer.	Insert USB-Stick, reboot and follow the instructions on the screen.	Clean system	Bootable usb stick		
0.002	Full	Pre	All	Complete driver installation.	<p>After the system has booted, the OS should automatically detect the new hardware and complete the driver installation.</p> <p>Vista: Let the installation complete automatically.</p> <p>XP: Go through the guide to complete the installation.</p>	<p>1. The driver installation completes without any errors or warnings.</p> <p>2. Device names presented during installation are according to branding sheet.</p> <p>Vista: Drivers should install automatically. A tray icon should give feedback on the process.</p> <p>XP: User will be prompted that a new hardware has been found. After the guide for all driver components completes a tray icon should notify the user that the hardware has been installed and is working properly.</p>	Branding sheet		

1 - Installation

1.001	Full	Pre	All	(PRE) Preparation	Do steps in 0-Preparation.				
1.002	Full	Inst	All	Run EMBLA Logger Setup.exe.	Complete the installation wizard.	<p>Verify that the following files exist in the chosen directory:</p> <p>log4net.dll Logger.exe Logger.exe.config \\Config\\ComPortNames.xml \\Config\\EricssonPubKey.xml \\Core\\DebugMuxClientAPI.dll \\Core\\DebugMuxSrv.exe (hidden) \\Help\\help.chm</p> <p>Verify that shortcuts are placed on the desktop and in the start menu.</p>			

2 - Startup									
2.001	Full	Pre	All	(PRE) Preparation	Do steps in 0-Preparation.				
2.002	Full	Pre	All	(PRE) Installation	Do steps in 1-Installation.				
2.003	Full	Strt	All	Start EMBLA Logger.	Run Logger.exe.	Verify that the program starts without error messages.			
2.004	Full	Strt	All	Check firmware connectivity.	Right-click on the 'Firmware' node and choose 'Configure'. Make sure that a port belonging to the module is checked and is showing 'connected' (If impossible to get status connected for any of the ports, first make sure that the ports are not being used. If not, try closing the configuration box, right-click the 'firmware' node and choose 'force reset').	Verify that sub nodes are shown under 'firmware'.			

3 - Standard logging									
3.001	Full	Pre	All	(PRE) Preparation	Do steps in 0-Preparation.				
3.002	Full	Pre	All	(PRE) Installation	Do steps in 1-Installation.				
3.003	Full	Strt	All	(PRE) Startup	Do steps in 2-Startup.				
3.004	Full	Std	All	Choose a firmware data source to log.	Double-click on a treeview node that has a blue circle next to the name and is a subnode to 'Firmware'.	Verify that the icon on the chosen node is changed to a green arrow and that the start button is enabled.			
3.005	Full	Std	All	Choose a network data source to log.	Double-click on a treeview node that has a blue circle next to the name and is a subnode to 'Network' (If no nodes with a blue circle is shown under network, make sure the computer has an external IP-address).	Verify that the icon on the chosen node is changed to a green arrow.			
3.006	Full	Std	All	Choose a WinAPI data source to log.	Double-click on a treeview node that has a blue circle next to the name and is a sub node to 'WinAPI'.	Verify that the icon on the chosen node is changed to a green arrow.			
3.007	Full	Std	All	Start logging.	Click the start button (the green button with a white triangle).	Verify that the stop button (the red button with a white square) is enabled, the start button is disabled, the 'Add a marker' section is enabled and the 'Statistics' section is started.			

3.008	Full	Std	All	Add marker.	Write something in the text box in the 'Add a marker' section. Press enter or click the button next to the text box.	Verify that the 'Markers added' counter in the 'Statistics' section is increased and the text box is cleared.			
3.009	Full	Std	All	Stop logging.	Press the stop button.	Verify that a box for choosing where to save the log(s) is shown.			
3.010	Full	Std	All	Save the log(s).	Choose where to save the log(s) and press ok.	Verify that a message box is shown containing "File(s) were successfully saved" and file name(s) corresponding to the chosen folder.			
3.011	Full	Std	All	Check logging is stopped.	Press ok to close the message box.	Verify that the 'Statistics' section is reset and stopped, the 'Add a marker' section is disabled, the start button is enabled and the stop button is disabled.			
3.012	Full	Std	All	Check saved log file(s).	Browse to the folder selected in step 3.010.	Verify that file(s) exist in the folder with names containing the name of the chosen data source(s) and the date and time corresponding to when the logging was started.			
3.013	Full	Std	All	Check contents of log file(s).	Open the log file(s) in the folder with a text editor.	Verify that the file(s) do not contain any readable log data.			
3.014	Full	Std	All	Check temporary file(s).	Browse to the application folder.	Verify that no file(s) with a name containing the name of the chosen data source(s), the date and time of when logging was started and "_tmp" exist.			
3.015	Full	Std	All	Check log file(s) contents.	Use the EMBLA Decryptor to decrypt the log file(s).	Verify that the output files contains the added marker and possible log data.	EMBLA Decryptor		

4 - Automatic logging after reset									
4.001	Full	Pre	All	(PRE) Preparation	Do steps in 0-Preparation.				
4.002	Full	Pre	All	(PRE) Installation	Do steps in 1-Installation.				
4.003	Full	Strt	All	(PRE) Startup	Do steps in 2-Startup.				
4.004	Full	Auto	All	Choose a firmware data source to auto-log.	Right-click on a treeview node that has a blue circle next to the name and is a subnode to 'Firmware' and choose 'Auto-log after module reset'.	Verify that the chosen node has a blue arrow shown next to it.			

4.004	Full	Auto	All	Reset the module.	Right-click on the sub node to 'Firmware' with the name of the module and choose Reset.	Verify that all the sections in the GUI are disabled for 30 seconds, except for the 'Statistics' section that should start as soon as the chosen data source is available again. After 30 seconds the stop button and the 'Add a marker' section should be enabled.			
4.005	Full	Auto	All	Stop logging.	Press the stop button.	Verify that a box for choosing where to save the log(s) is shown.			
4.006	Full	Auto	All	Discard the log file(s).	Press cancel in the message box for choosing folder. Press yes for discarding.				

5 - Load script									
5.001	Full	Pre	All	(PRE) Preparation	Do steps in 0-Preparation.				
5.002	Full	Pre	All	(PRE) Installation	Do steps in 1-Installation.				
5.003	Full	Strt	All	(PRE) Startup	Do steps in 2-Startup.				
5.004	Full	Script	All	Load script.	Right-click on 'APP - Interactive Debug' and choose load script. Load the text file 'Ok script.txt' from appendix.	Verify that a message box containing "Script successfully loaded" is shown and that the icon next to the 'APP - Interactive Debug' node is changed.			
5.005	Full	Script	All	Choose a firmware data source to log.	Double-click on a treeview node that has a blue circle next to the name and is a subnode to Firmware.	Verify that the icon on the chosen node is changed to a green arrow and that the start button is enabled.			
5.006	Full	Script	All	Start logging.	Click the start button (the green button with a white triangle).	Verify that the stop button (the red button with a white square) is enabled, the start button is disabled, the 'Add a marker' section is enabled and the 'Statistics' section is started.			
5.007	Full	Script	All	Load script during logging.	Right-click on 'ACC - Interactive Debug' and choose load script. Load the file 'Erroneous script.txt' from appendix.	Verify that a message box containing "Error when sending script." + "XXX" (the erroneous command(s) of the script) is shown and that the icon next to 'ACC - Interactive Debug' is not changed.			
5.008	Full	Script	All	Stop logging.	Press the stop button.	Verify that a box for choosing where to save the log(s) is shown.			

5.009	Full	Script	All	Discard the log file(s).	Press cancel in the message box for choosing folder. Press yes for discarding.				
-------	------	--------	-----	--------------------------	--	--	--	--	--

