Web Service and Client API for server side node locked licensing

Master's Thesis

submitted by

Gloria Janjic

Supervisor: Ao.Univ.-Prof. Dipl.-Ing. Dr.techn. Nikolai Scerbakov

Graz University of Technology

Institute of Interactive Systems and Data Science

Graz University of Technology

A-8010 Graz, Austria

In Cooperation with:

Graz, May 2017

Webservice und Client API für serverseitige node-locked Lizenzierung

Masterarbeit

vorgelegt von

Gloria Janjic

Betreuer: Ao.Univ.-Prof. Dipl.-Ing. Dr.techn. Nikolai Scerbakov

Technische Universität Graz

Institute of Interactive Systems and Data Science

Technische Universität Graz

A-8010 Graz, Austria

In Kooperation mit:

Graz, Mai 2017

EIDESSTATTLICHE ERKLÄRUNG

STATUTORY DECLARATION

Kurzfassung

Diese These ist eine Studie welche sich damit beschäftigt, wie ein Lizenzierungssystem für eine unabhängige Software Firma implementiert werden könnte. Die Lösung dieses Projektes basiert auf den Anforderungen und Beschränkungen seitens der Firma, welche durch die bereits existierende Umgebung gegeben sind. Der Zweck dieser Arbeit ist es, eine Implementierung zu erstellen welche diesen Anforderungen entspricht und sich optimal in das Framework der Firma einbauen lässt. Ziel dieser Arbeit ist es nicht, das resultierende Projekt für Verkaufs- Zwecke zu nutzen. Trotzdem sind die Grundvoraussetzungen da, um mit einigen Modifikationen das System in Zukunft auch für diese Zwecke zu benutzen. Dieses Lizenzierungs- System ermöglicht es demnach weitergehende Software Implementationen durchzuführen. Die wichtigste Aufgabe dieser Software Lizenzierung und User Management Systems, ist die Implementierung der serverseitigen Datenbank, eines HTTP Web Services und einer Client Access API Bibliothek. Alle drei Komponenten wurden von Grund auf selbst erstellt, nachdem sorgfältige Recherchen durchgeführt wurden.

Auf Seiten der serverseitigen Datenbank, wurden Informationen des Users implementiert, welcher in diesem Fall eine Software Firma ist. Zusätzlich werden Informationen über die Mitarbeiter einer Firma welche die Software Lizenz Instanzen verwenden aufbereitet, diese werden als End- User angesehen. Die Datenbank enthält auch Informationen der geschützten Software und Details zu Lizenzen welche sich nach dem „hardware-node locked“ Prinzip auf dem Computer des Benutzers befinden.

In der existierenden serverseitigen Implementierung wurde server side Scripting eingebaut welches in diesem System funktioniert. Die Client API Bibliothek wurde mit dem Software Produkt, welches in C/C++ erstellt wurde, verbunden und bietet ein Set an Methoden für „silent“- Anfragen an den Server und die Bearbeitung von Antworten. Der Web Server, welcher hinter einem Apache Server läuft, unterstützt Methoden für Client Anfragen, das Updaten vom Benutzungsprotokoll und das Zurücksende von Antworten an den Client.

Abstract

This thesis is a study on how a licensing system for an independent software company could be implemented. The solution of this project is based on some identified requirements and limitations connected to the company's already existing environment. The aim of this work was to create an implementation that fulfils those requirements and perfectly fits in the given framework the company is already working with. Using the resulting project as a selling software was not part of the goal. Nevertheless the system is created in a way, that with some modifications it could be a task of future work. The designed licensing system therefore provides a usable system which enables options to future software developing. The main task of this software licensing and user management system was implementing and designing a server side database, a HTTP web service and a client access API library. All three components were created from scratch, after some research was done in the first place.

The server side database implements information about the user, which in this case is a software company. Additionally it provides information about the company's staff that uses software licenses, which in this case are identified as the actual end users. Besides that, the database contains information about the protected software products and license details which are all hardware- node locked on the client computers.

In the existing server implementation a server side scripting was integrated which works with the existing system. The server side Client API library was linked to software products developed in C/C++ language, providing a set of methods for silently sending license requests to the server and getting responses back. A web service which is running behind an Apache server, supports the client license requests, updates usage histories and sends responses back to the client.

Acknowledgement

Firstly, I would like to express my deep gratitude to my mentor and advisor professor Nikolai Scerbakov for the continuous support during this whole process. Not only has he always been there for me when I needed advice, but he has also been a steady inspiration during my whole studies at the Technical University of Graz. He consistently allowed this thesis to be my own work and guided me in the right direction whenever I needed it.

Additionally I would like to thank my employer and co-workers for helping me during this period of time. They helped me come up with the thesis topic and guided me during the whole development. Without the help of these experts and their passionate participation, this project could not have been successfully completed.

In these lines I want to thank all the special persons of my personal life that contributed to making this work possible. I must express my very profound gratitude to my parents and my little brother who provided me with unconditional support and continuous encouragement throughout my years of studies as well as the process of researching and writing this thesis. Of course, I do not forget the rest of my family who supported me even though we are living in different countries. Despite the distance, they were always there for me giving me love, inspiration and motivation. Last, but certainly not least, I send big thanks to all my friends that always gave me good advice and made my life colorful!

THANK YOU.

Gloria Janjic

Contents

1 Introduction

Software piracy has become a big problem nowadays and it continues to grow as new technologies make it easier to share files. The goal of this master thesis is to develop a software licensing system that is more or less secure, using the current state of nowadays techniques. In order to keep track of the sold licenses the aim is to develop a new licensing system.

The company which is going to use the licensing system, provides Consulting Services for Engineering and Structural Analysis and is based in Graz, Austria. Out of security reasons, they chose to be anonymous and not be mentioned by name. For the usage of a new software an online licensing system has to be developed. The installation package will be downloaded from the software provider’s website. After the software is purchased, the user gets an activation key which gives access a software license instance.

The work is grounded on specific requirements which are made by the company. Therefore, it is the solution to a specific problem rather than a general one and developed to fit perfectly into the environment of the owner’s software. The focus lies on designing and developing a functioning licensing solution in a suitable way.

This chapter first explains the goal, objectives, methodology and the structure of the thesis and topic. Besides that, some existing techniques and theories regarding software licensing, client- server application, server- side databases and cryptographic issues are presented. Additionally information about the technical requirements from the developer’s side of view is provided followed by tables describing the solution requirements. The basic idea, a solution and method overview, but also some predicted error cases and workflows are shown.

1.1 Goals and Objectives

The main goal of this project is to develop a software licensing system that can later on be used in the above mentioned company. Rather than creating a perfect system that can compete with other professional, outstanding products on the market, the goal is to learn about the general functionalities of software licensing. This knowledge and technologies can later on be used in plenty of different applications. This whole project is built from scratch with the help of several libraries which are going to be mentioned later on.

To address the problem of tracking sold licenses a license management system needs to be developed. The task is to ease the usage and documentation for both the company and its clients. In the end it should save time and money by keeping an overview of the sold software and reducing working hours. Besides that, it is a good way to protect the company’s knowledge and code. This is achieved by making the illegal use of the software more difficult, protecting the software by using cryptographic techniques. It is truly believed, that this will immensely increase the market for the software.

The implementation includes the following:

- Design and implementation of the Server Side Database with information about the company’s licensing system
- Design and implementation of the Client Side API interface
- Design interface with request- response for the Web Service

The goal of this implementation was not to make a conventional licensing customer management system which can later on be sold as a software service. Nevertheless the architecture allows it in terms of further and prospective work.

This work’s purpose is, besides presenting and analyzing the practical part, also the investigation of protection methods and comparisons between these.

1.2 Structure of the thesis

The following work concentrates on the company’s needs and its already existing environment. Taking that into consideration, the focus of this thesis lies on implementing a working licensing system in this particular, already existing environment. As this is a rather big topic, it is assumed that there is going to be further, future work on this thesis after the successful implementation in order to cover all thinkable use cases.

This master thesis is structured in 3 main parts which are described as follows.

The first part of the thesis gives an introduction to the topic describing why this work was done. Besides that it discusses and explains theoretical background on which the thesis is based on. In order to understand the requirements of the implemented project, it was needed to outline some cryptographic techniques and how they are used. After summarizing those shortly, a solution- and method overview is given followed by an expected workflow of the whole project.

The second part focuses on the solution and implementation of the previously stated requirements. The preparation of the chosen tools is presented along with the concrete implementation details of the Server Side Database, Web Service and Client Access API Library. This part also contains the functionality, architecture and interface from the user’s point of view.

In the third part of this thesis, the project is analyzed and evaluated. It is looked into some problematic scenarios and improvements that could be used for future related work.

1.3 Technical Background

The thesis works on the knowledge which is presented and looked into in the following chapter. It summarizes the necessary information needed to understand the implemented solution for the given problem. The first part discusses some definitions about software licensing, so it is clear when used later on in the document. Later on the theory and definitions behind the 3 main parts of the implementation are discussed. Those are the Client- Server Application, a Server Side Database and the HTTP/HTTPS Web Services.

1.3.1 Software Licensing

A software license is a way of providing guidelines about the usage of the software to a customer. By downloading a license an end user gets the right to one or more copies of the software product. It defines responsibilities and restrictions on how the software can be used. Usually one can distinguish between *proprietary*, *free* and *open source* types of licensing which differ by future development redistribution and copying of the software.

Proprietary software means, that the publisher of the software holds the intellectual property for himself. Usually this is the copyright of the source code or a patent right. *Free* and *open source* software is made available with less restriction on licensing. That is, anyone is freely licensed to use and change it any way and the source code is openly shared so people get involved and improve the design of the software.

Software licensing allows programmers to protect the software and also make profit out of every copy installed. In this thesis the term will be used referring to the authenticating access and process to software usage. Usually this is achieved by the usage of a key string which is provided by the owner/vendor and used as an input for the client. This technique can also be used for upgrades or updates of the vendor’s software. Therefore the implemented system has a licensing authority and an application. The license authority gets several inputs some of them could be: a unique ID for a license, a list of what is featured in the license and an expiration time which is checked to be up-to-date.[1]

Software license manager

With a License Manager Software Tool companies and organization can manage their software licenses, with the goal to determine whether the amount of software licenses matches the number of used software. One way to do so could be, to create a database which contains information about all sold licenses. Checking all listed licenses and their usages, losses due to software piracy should be discovered or prevented rather easily.

A software asset management tool is a business practice that intends to be a part of the business strategy of an organization. Its goal is to limit business and legal risk and reduce costs for information technology.

Code based protection

This kind of protection usually relies on a unique code which later on is used to allow the usage of something protected. This code is verified by the protected software and could be anything desired. Generally one can distinguish between local installation based and online activation based schemes.

[1] (Rouse, 2014), (Wiki1, 2016)

Local installation based schemes:

This is a strategy, which allows the end user to carry out the validation of the license code. In this case no communications to other servers or systems are necessary. The end user is provided with a unique code which contains some sensitive information about the needed license. The protected program is able to verify this data during the validation process. When the software program gets executed, it checks if the code has been installed or not. If the code had not been validated before, the user is asked to do so with a new code. If the code validation is successful, the program saves the code and allows the usage of the application.

Even though this way of protecting software is rather simple, it does show some major disadvantages. The end user does not need to compromise on usability or privacy in this case. On the other hand speaking in terms of security for the system, this scheme has some fundamental problems. It is difficult to put all necessary information into this one string of code and a user is therefore able to reuse it on different computers and install the software multiple times.

Online activation based schemes:

Usually, all activation schemes in practice use an online based activation. To do so, the software distributor generates a unique code (activation key) for each license. This happens the same way as it is in done in local activation based schemes. The difference is that this unique code is only used to look up the real license data rather than saving information about the license data in the key itself.

The protected software communicates with an activation service and checks if activation key has been already used. The activation service checks the license key and possibly other additional data. Depending on the requirements of the software developer this additional data differs. The request of the user is validated and the appropriate result is sent back to the client.

From the client’s sight of view, this scheme has some possible privacy restrictions. As the online version needs a connection over the internet in order to communicate with a server, it could be inconvenient in some circumstances. Besides that, an activation service could keep track of the activation of a user. However, from the owner’s side of view this system opens ways of stronger protection and setting limits to the amount of allowed hosts per license and besides controlling the usage. This might also be the reason why most systems apply this method rather than the previous one. [2]

Licensing Using Cryptography

The best way of securing software with a licensing system is to use cryptography. Through cryptography you can secure a licensing system in two different ways using a symmetric or asymmetric key. This part will be looked into deeper in chapter 1.4. Using this, some of the software misuses can be avoided. Controlled usages are possible to realize and an appropriate pricing model for the producing company could be performed. Protections that

[2] (Carlsson, 2014)

are based using cryptography can stop illegal usage of software but is often combined also with other technologies.

Cryptography provides a system with many objectives. One of which is Data Integrity, that works in the field of unauthorized modification of critical data. Another objective is Data Origin Authentication which deals with the identification of an entity that created some specific data. For all code based protection systems, algorithms that generate data for critical data are necessary.[3]

Types of Software licensing

Software owners have to decide which way of protecting their property is the best for them, depending on the program but also the license strategy. Licenses are also keen to be helping to make companies a profit in a way that is desirable.

There are differences between standalone/single user licenses and network/multi user licenses. The first instance allows only a single machine or user to use a license of a software, whereas the second instance lets more than one user use the software at the same time.

“Product key only” licensing is not recommended, because the owner does not know about the concrete number of users working with the software. It works by providing the user or company with a product key which is to be entered before using the software. Unfortunately, in case of only using this key, the software can be used over and over again over multiple computers. There are several types of systems like this (serial- only systems), but the fact that the license can be shared endlessly applies to all of them. Even though implementation costs seem to be lower for it, the owner of the software loses much more in real life usage on the copies of license.

In the implementation of this particular software, a standalone/single user licenses was used. Moreover it is a “node locked” licensing system, which will be discussed in the next paragraph about hardware- locked licensing. It has the benefit that the owner has complete control over user management and by that it is possible to prevent major kinds of piracy but also increase the revenue.[4]

Usually a software that is protected by this type of licensing, is very expensive which makes its owner fight piracy even harder. As the company’s intellectual property of the software is rather expensive and copies are limited, this kind of implementation was chosen.

Hardware- locked licensing

The usage of well designed node- locked licensing systems with the online activation might look like a serial- only licensing in the eyes of a user. In both cases the user gets a product key with a specific length which has to be entered in the program. After being activated the application will either let the user work with the program or encourage buying another license.

[3] (Shamir)

[4] (Bastion, 2015), (LimeLM, 2017)

Fingerprints

In order to understand what happens on the server side of this procedure, it is necessary to be familiar with the term of *fingerprints in computing*. Fingerprints can be seen as the human fingerprint which uniquely identifies people for a given purpose. It is a way of taking an amount of data and map it to a shorter bit string. They can be seen as a sort of hash function (see Chapter 1.4.1), offering a high performance. There are some properties a fingerprint should contain, so it serves the purpose. One of them is that it has to be unique and the probability of a collision has to be minimal. A collision describes when two inputs result in the same output. This probability of this scenario should be mineralized by choosing the right method of creating a fingerprint. [5]

The procedure is as follows:

1) The customer of software enters a product key (e.g "A1B2-C3D4-E5F6-G7H8…)
2) A fingerprint, which uniquely identifies the host computer, is created
3) Information is send to the corresponding server of the program
4) Server decides whether to allow an activation and sends back an encrypted version of the fingerprint and the product key
5) Based on this information the owner will know if a user is allowed to use the application or not

Metrics which are used for fingerprints should not change, unless the hardware ofr the machine directly is changed. Therefore some metrics that could be used for fingerprints, are the following ones:

- Type of processors and processor ID
- Details from graphic card
- RAM number and size
- Brand, screen resolution and number of monitors
- MAC address
- Hard- drive serial number
- Serial number of installed HDs
- Amount of other devices (e.g CD, DVD, readers…)

Advantages of node- locked licensing:

The owner knows the amount of users and also who they are. Besides that, he can have an insight in the activity log of each user telling him when and how software is used by a client. It is unnecessary to mention what an impact this could have for the profit of an organization, helping to avoid piracy.

[5] (Encyclopedia, 2016)

Disadvantages of node- locked licensing:

It is a complex process to build this licensing process, including IT security issues along with user management on the server side. Besides that, it might need some further maintenance keeping all cryptography methods up to date. The most important and difficult part of a node-locked licensing system is to secure it.[6]

1.3.2 Client- Server Application

In order to understand the up- following chapters with Server Side Databases and the implementation techniques, it is important to know about the term of Client- Server relationships. It is a structure for distributed applications where a client requests a service and a server on the other hand provides a service. Service in this case could be any resource from files to objects or other types of data. Often Client and Server operate on different hardware distributed over a computer network.

Servers could be for instance file servers that provide computer files or web servers that provide web pages. The shared information could be anything like computer's software or programs, data of processors or storage devices. Computers are determined to be client, server or even both.[7]

Request- Response

Messages between Client and Servers are exchanged by methods called Request-Response (or Request- Reply). The communication between those two happens in the way that the first computer sends a request for some data after which the second computer responds to this request. Until a message is completely sent, a series of interaction has to happen.[8]

1.3.3 Server Side Database

Website scripts are either run on the client or on the server side. When it comes to websites, the client is usually a web browser that is showing the content. The client is usually referred to as the front- end side which request pages from the server. The server side is also called back- end side and is there to serve pages.

Server- side programming is generally used as a term to talk about the programs that are run on the server itself. Processing users input, interacting with SQL files, displaying pages and structuring web applications are just some of the cases in which server- side programming is used for. Example languages for that are PHP, Python, C#, C++ and Visual Basic but nearly any language can be used as an application- level service.

[6] (LimeLM, 2017)

[7] (Chung)

[8] (Wiki2, 2016)

Server- side scripting is mostly used to interact with databases which are located on the server. By working with these, it is possible to update content which is displayed on a webpage without changing the HTML. This is how one template can be used for many different pages if the information is stored in a database.[9]

There are many SQL database systems which can serve for implementation of licensing system. As an important criterion for choosing database systems, one has to take the complexity/simplicity of the database installation and management into consideration.

As potential candidate one can refer to simple SQLITE database system and MYSQL database. The following RDBMS (Relational Database Management System) are important open source applications that are popular and important in the world of application development. SQLite is an embedded and powerful relational database management system whereas MYSQL counts as the RDBMS that is the most popular and commonly used application.

SQLite

SQLite is contained in a C programming library and is used for managing relational database. Compared to other managing systems SQLite is directly integrated in the end application process rather than in a client- server database. It manages different kind of data with much less constraints. Instead of making communication through an interface (ports, sockets) SQLite, when integrated in an application, makes calls to a SQLite database holding this data. Thanks to that, SQLite is extremely fast and efficient.

Advantages of SQLite:

The entire database is very portable because it is file based and it only consists of a single file on a disk. SQLite is standards- aware by using SQL. It is great for development and testing as it works with a linked C based library.

Disadvantages of SQLite:

Compared to advanced databases, SQLite has no user management. There is no support for users and managed connections. Since the library is not complicated, it is technically not possible to make it more efficient in terms of performance than it already is.

In conclusion one can tell that SQLite should be used for embedded applications, that do not need expansions and the ones where the main purpose is the functionality. When working with applications where multiple clients need to access database it might be better to use MySQL over SQLite.[10]

MySQL

MySQL is an open- source management system used for relational databases and is written in C and C++. It is also the most popular of all the large- scale database servers. Unlike in SQLite applications communicate with MySQL daemon to make an access to the database.

[9] ({cc}codeconquest)

[10] (Tezer, 2014)

Advantages of MySQL:

It is a system that is easily installed and not complicated to work with, besides that it offers a lot of SQL functionalities that are needed by the users. MySQL is secure, scalable and powerful which allows it to handle a lot of data.

Disadvantages of MySQL:

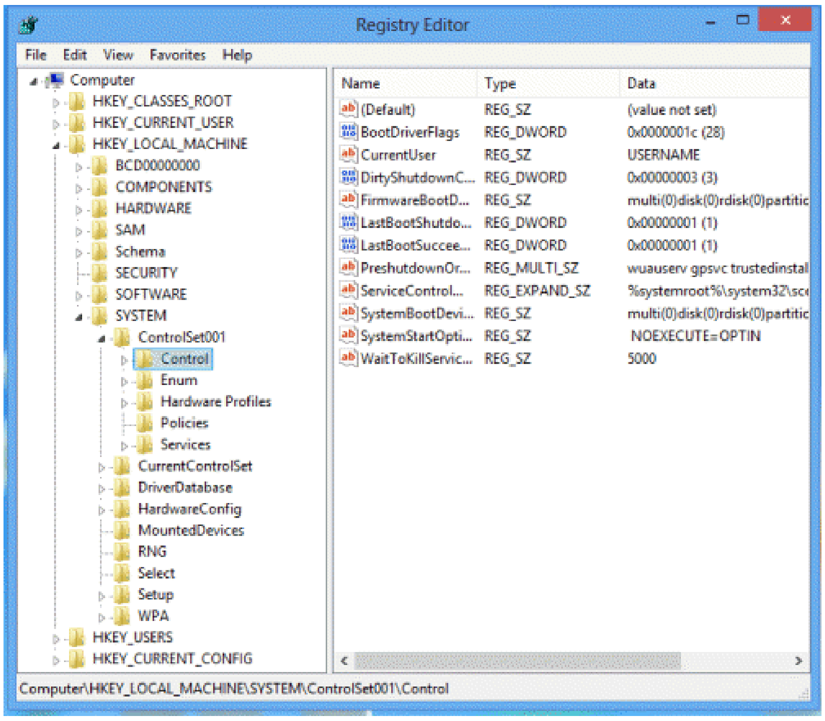
Nevertheless, some known functional limitations and reliability issues compared to other RDBMS do exist.

In conclusion one should use MySQL when you need operational freedom and high security as there are features to provide protection while data- access. As this tool is not completely SQL standard switching from MySQL might not be easy.[11]

Windows registry [12]

This registry is a hierarchical database which stores low level setting for a Microsoft Windows operating system. It contains information about hardware and programs installed on a host computer and their system settings. Besides that, it contains information about users of the computer. One of the most important parts of this work is to create a unique fingerprint of a computer, so this information is crucial to collect.

The windows registry structure consists out of keys, sub- keys and values for the registry. It can be found on windows computer by searching for “RegEdit” tool. An example of a Registry Windows editor is shown in Figure 1-1.



Name	Type	Data
(Default)	REG_SZ	(value not set)
BootDriverFlags	REG_DWORD	0x0000001c (28)
CurrentUser	REG_SZ	USERNAME
DirtyShutdownC...	REG_DWORD	0x00000003 (3)
FirmwareBootD...	REG_SZ	multi(0)disk(0)rdisk(0)partitic
LastBootShutdo...	REG_DWORD	0x00000001 (1)
LastBootSuccee...	REG_DWORD	0x00000001 (1)
PreshutdownOr...	REG_MULTI_SZ	wuauserv gpsvc trustedinstal
ServiceControl...	REG_EXPAND_SZ	%systemroot%\system32\sc
SystemBootDevi...	REG_SZ	multi(0)disk(0)rdisk(0)partitic
SystemStartOpti...	REG_SZ	NOEXECUTE=OPTIN
WaitToKillServic...	REG_SZ	5000

Computer\HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\Control

Figure 1-1: Windows registry editor (tech support alert, 2015)

[11] (Tezer, 2014), (Wiki3, 2017)

[12] (wiki4, 2017)

1.3.4 HTTP/HTTPS Web Service

Web Services are software components that can interact and communicate between electronic devices over a network, using open protocols via the World Wide Web. Many organizations often use different software solutions among their systems and a web service provides a way of communication among them. This communication allows all systems to be in contact with each other. It can be distinguished between the service requester, a system that requests data and the service provider, which is a system that provides data. As different systems use different programming languages, the data exchanged has to be independent of those languages. XML tags can be interpreted by most kind of software, which is the reason why it is often used for web services.

They can be discovered using Universal Description, Discovery and Integration (UDDI) which lists what services are available. Each web service has a string of character called Uniform Resource Identifier (URI) that is used to identify the source. HTTP and XML are the basis for web services. The HTTP protocol is used for communication from one machine to another, using files that are readable for machines. XML and JSON are such formats that contain those files. The end user is usually provided with a web- based interface to a database server. Using Simple Object Access Protocols (SOAP) over HTTP allows less costly interactions. It is a protocol with some specification used in web services of computer networks.[13]

HTTPS is developed by Netscape and stands for Hyper Text Transfer Protocol. It is a TCP/IP protocol used by Web Servers to display Web content. It is used in the way that data which is about to be transported is encrypted first, so it cannot be read by anyone except the recipient. It can protect against man- in- the middle attacks and eavesdropping.[14]

1.4 Cryptographic techniques

Nowadays modern cryptography is based on mathematical theories making the algorithms hard to break. Most of the following described techniques are used in the thesis and therefore looked into further detail. The following subchapters describe the general purpose of cryptographic hash functions, as well as symmetric and asymmetric key algorithms. Furthermore the specific techniques used in the thesis are described in detail so it makes it easier to understand their usage in the practical part (see Chapter 3). Not all of the algorithms and modes are used in practice. Nevertheless it is in the author's intension to present all of them in order to better understand the decision made in the application.

The fictional characters called Alice, Bob and Eve are going to be used in order to explain some underlying cryptographic issues. Generally speaking, Alice and Bob are usually trying to exchange some secrets in a secure way, whereas Eve is an eavesdropper that is trying to attack that and find out the secret.

[13] (ITWissen, 2013), (w3schools), (Wiki4, 2017)

[14] (Kyrnin, 2016)

1.4.1 Cryptographic hash functions

The basis of cryptographic hash functions is of course the *hash function* itself. Hash functions use an arbitrary big input and develop an output with a fixed size. Besides using them for encryption, they are used when looking for items in databases. This is due to their immense efficiency, as it is much faster to look for a short hash than a long string. One of the most important properties of hash functions is that it is almost impossible to guess the input, when knowing only the output. This property also makes hash algorithms being so called *one way functions*. The definition of a one way functions claims, that it is significantly easier to compute in one direction than in the opposite direction. For the aim of giving a feeling to the reader: In some examples this means that computation in the forward direction takes a few seconds, whereas the inverse computation can take up to a month or year or maybe it is not computable at all.

There are different kinds of hash functions that are used nowadays of which some are MD5 or SHA1. Those are also the two functions that are mostly used. MD5 stands for "Message Digest 5" and has an input length of 128- bit. Whereas SHA1 stands for "Secure Hash Algorithm 1" and supports message inputs of any length less than 264. SHA1 is also considered the more secure algorithm of those mentioned two.

The reason why those algorithms count as secure is the fact that there are no known techniques for finding collisions. Collisions in this case mean, that two different messages can result in the same output. [15]

SHA-1, SHA-2, SHA-3[16]

Special cryptographic methods needed to be developed to ensure security of SSL/TLS (Secure Sockets Layer/ Transport Layer Security) when sending and receiving important data over the internet (HTTPS).

The cryptographic function SHA was created by the NSA and improved security issues due to the increased number of operations before a collision. It takes a maximum of 2^64 bits as an input and calculates a 160 bit output hash. The predecessor of it was SHA-0 which was not as technically developed as the improved version SHA-1. There are already two successors with improvements over SHA- 1 which are SHA-2 and SHA-3. Even though some of the attacks on the encrypting systems are just in theory, the history of cryptography shows that the theoretical attacks lead to a practical attack sooner or later.

SHA-1 and SHA-2 are two versions, which are showing differences not only in the bit- length of the signature but also in the way that the input data is hashed into string of encrypted bits. Rather than having a fixed length of 160- bit SHA-2 comes with a family of hashes in different kind of bit lengths. Websites and authors call the SHA-2 encryption with different names, only because of the variation in bit length. Some of them are referring to the alternate bit length

[15] (techopedia), (Silva, 2003)

[16] (Lynch, 2016), (MD5 decrypt, 2015)

(SHA224, SHA384, SHA512), whereas others are being more explicit about the exact bit length and the name of the algorithm itself (SHA2 384).

SHA-1 was the primary used algorithm for hashing digital signature, before some researchers showed the weaknesses. Since 2016 SHA-2 is the new standard, which also needs to be used when getting a certificate. It is most likely that it will keep being a standard for several years, even though some unexpected attacks can always be discovered.

Here is how the string "Master thesis by Gloria Janjic" looks like when encrypted with SHA-1 and SHA-256 (SHA-2). In order to calculate this hash a website called MD5 decrypt is used.

Encryption	***"Master thesis by Gloria Janjic"***
SHA-1	*d9099d67fe9a11451a84347a8c6fa10ef0d1ad6e*
SHA-256	*00b6da3ff40fd9773ad6653473fff95e7c3a0ce4ec539bda15ada92c76257c42*

In general, one can say, that a larger hash provides more security as it means there is more different combination that can be the solution (no collisions).

1.4.2 Symmetric key algorithm

This technique is also called secret key encryption and is one of the oldest and best- known encryption systems. The algorithm is based on a secret key which can be a number, a word or a string of random letters which is applied to the message that needs to be encrypted. This key is used for encryption and decryption of the message. Sender and recipient know the secret key and are able to encrypt and decrypt the message. Some of the most famous symmetric key encryptions are: AES, DES, Blowfish and Skipjack. The main disadvantage of this type of encryption is that the parties involved have to exchange the used key before using it to encrypt or decrypt.[17]

Block ciphers vs. Stream cipher [18]

Talking about symmetric algorithms, they can be further divided between stream ciphers and block ciphers. They differ in the amount of bits encrypted from a plaintext into a ciphertext at a time.

Stream cipher: [19]

Using stream ciphers, each bit of the plaintext is encrypted individually. This encryption algorithm encrypts one bit at a time adding a bit from the key stream. That key stream consists of an infinite stream of pseudorandom bits. In synchronous stream ciphers the key stream depends on the key only, whereas asynchrony stream ciphers additionally depend on the ciphertext.

[17] (Higashi, 2013)

[18] (Pelzl, 2010)

[19] (Villanueva, 2015)

The central issue for the stream cipher is the generation of the values for the key stream. In fact, its whole security depends on exactly that. Therefore it is pretty much the key to all stream ciphers. The most important thing is that for an attacker these bits appear random. As they should be unpredictable and never reused this leads us to the One- Time Pad (OTP).

The OTP is supposed to give perfect security and a technique that cannot be attacked. In case the key is truly random, kept secret and never reused it would indeed provide perfect secrecy. Nevertheless it requires a key which is the at least the same size as the message itself, which is mainly the reason why this algorithm was never widely used. Keys of stream ciphers are no longer as long as the original messages, which means they cannot guarantee perfect security for the secret message encrypted. Anyways, it can achieve a high level of security. Examples for stream ciphers are FISH, RC4, SEAL, SNOW, ISAAC etc.

Block cipher:

Whereas stream ciphers encrypt plaintexts bit after bit, block ciphers encrypt entire blocks of plaintext bits at a time, using the same key. Therefore the encryption of one single plaintext bit in a block depends on all the other plaintext bits in that particular block. The majority of block cipher algorithms typically uses a length of 128 bits like AES or 64 bits like DES and 3DES or 256. In some cases Padding needs to be added, because the plaintext might be shorter than the length of the block size. These cases will be mentioned later on, when talking about the different modes of operation.

Nowadays mostly block ciphers are used for symmetric cipher encryption, which is why later on in practice this mechanism was chosen. Some of the commonly used encryption algorithms based on block ciphers are DES, 3DES, AES, IDEA and Blowfish just to mention some.

Comparing stream cipher and block cipher:

- In practice one can say, that mostly block ciphers are used for encrypting communication over the internet but in the case of RC4 stream ciphers are used
- Stream ciphers are used in cases where there is lower computational resources: old mobile phones or other small devices
- An advantage of stream ciphers is, that they are usually small and fast
- Stream ciphers are usually also very simple and much faster in operation, whereas block ciphers tend to be complex and slower

Importance of Initialization vector

In the following examples of symmetric key block mode operations, the initialization vector will be used and referred to as IV. In order to understand why they were used in the first place it will be discussed why this vector is that important.

Basically what happens when using an IV is that it adds an additional randomness to the start of the encryption process. Generally speaking, if there was no such thing as an initialization vector on the beginning of the encryption, two ciphertexts that begin with the same characters would produce the same block ciphers for that part. Therefore an attacker that notices that similarity could take advantage of that fact. He or she could deduce parts of the plaintext as well as the key. Obviously, once the attacker has the key he could decrypt the whole ciphertext. On the other hand, when an IV is used even if the plaintexts were the same in the beginning the block encryption mode would give different output for that part. Nevertheless this would only be the case if the IV always has a different value and not the same for each encrypted text. In case the IV is always the same, it would lead to the same problem as not using an initialization vector.

<u>XOR Cipher:</u> [20]

This encryption cipher is a type of additive ciphers and works with the following principles:

$$A \oplus 0 = A,$$
$$A \oplus A = 0,$$
$$(A \oplus B) \oplus C = A \oplus (B \oplus C)\ ,$$
$$(B \oplus A) \oplus A = B \oplus 0 = B$$

The $\oplus$ denoted the XOR (exclusive disjunction) and has the same logic of a modulus 2 operation. In order to explain this, the basic idea of this encryption has to be discussed. An additive cipher is the basic idea for the encryption, which adds a key stream (marked with the letter K) to a plaintext (Marked with a letter P).

$$C = K + P$$

The problem in this lies in exceeding the range of the bits that can be represented. In order to fix that, one can use modulus 2. [21]

$$C = K \oplus P$$

A truth table for this operation looks like this:

K_i	P_i	C_i
0	0	0
0	1	1
1	0	1
1	1	0

Possible attacks could be done in the following scenario assuming there are two encrypted messages C^1 and C^2:

$$C_1 = K \oplus P_1$$
$$C_2 = K \oplus P_2$$

By adding those two messages together, one could get rid of the keystream and obtain parts of the two messages and key stream:

$$C_2 \oplus C_2 = P_1 \oplus P_2$$

[20] (Wikipeida, 2017)

[21] (Roberts, 2009)

Block Cipher modes of operation: [22]

Block cipher modes are algorithms that work by using block modes to provide security in encryption of secret data. Nevertheless a block cipher mode by itself is only used for the secure transformation of one fixed length of bits. It only describes how a block operation is applied on big amounts of data in order to encrypt or decrypt messages. In addition to a block mode, in practice when encrypting anything one would need another algorithm like AES to complete encryption and decryption. Anyways it is important to understand the following block cipher modes.

ECB (Electronic Code Block): [23]

The ECB mode is the simplest block cipher mode operation, where the plaintext is divided into several blocks and each block is encrypted separately without depending on the following encrypted block. Figure 1-2 shows the encryption using exactly that block mode operation.

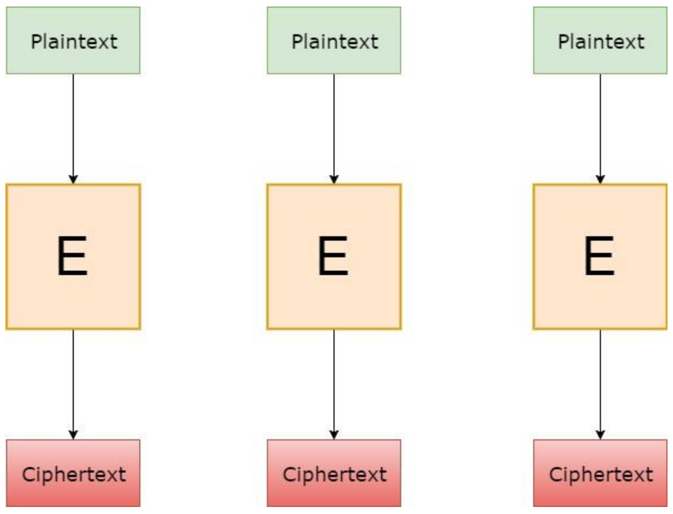


Figure 1-2: ECB block operation mode
(©Gloria Janjic, 2017)

Therefore the cipher text of two same messages is always the same which makes it vulnerable for attackers that want to decrypt the secret message. This method does not hide patterns and repetitions very well. An attacker can create and use a dictionary of known pairs of plaintext and ciphertext in order to guess the message. Besides that, Eve can modify messages without the receiver knowing. When implementing, one has to be aware that ECB works only with messages that are a multiple of the block length. This mode works only with a limited number of applications and is not widely used.

[22] (Mendel)

[23] (tutorialspoint, 2017)

CBC (Cipher Block Chaining):

In this block encrypting mode, each block of a plaintext is combined with the previous ciphertext block by using the operation XOR (exclusive OR). The XORed data result of those two inputs is afterwards encrypted with a secret key and results into a ciphertext block. In Figure 1-3 the CBC encryption is shown in a more graphical way.

Unlike ECB the Cipher Block Chaining mode has a way of hiding patterns to avoid attacks like in ECB mode. Repeating the encryption of the same message over and over again does not help the attacker here. That is, because an additional initialization vector is used to add the component of randomness to the encryption mode. As one can see in the graphic, this mode has dependencies between cipher text and plaintext blocks.

The last ciphertext block for example depends on all plaintext blocks.

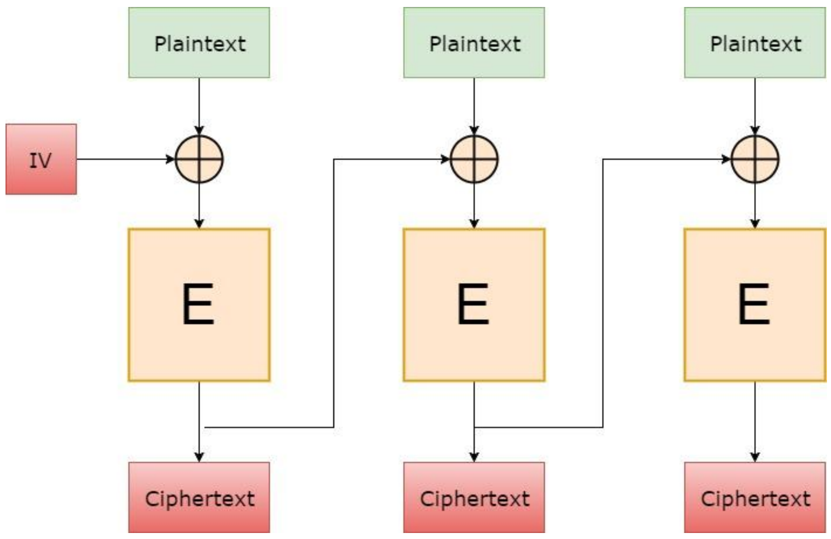


Figure 1-3: CBC block operation mode

(©Gloria Janjic, 2017)

Disadvantage of this mode are, that the encryption cannot get parallelized messages must be padded to a multiple of the used cipher block size. The padding oracle attack can be applied to this mode, where data about whether padding is used or not is leaked. Using this, attackers can reveal the plaintext message without using the encryption key.

CTR (Counter Mode): [24]

This mode uses block cipher to build a stream cipher and works with a so called counter function (CTR) which produces a sequence of numbers. This counter function is used instead of a traditional IV (Initialization Vector).

The National Institute of Standard and Technology basically defines two types of counters. The first type is created by taking a random nonce of a defined number of bytes

[24] (Cryptopp, 2016)

concatenated with counter bytes which get incremented. The second type is a counter block, where each byte can be incremented (generating carries).

CTR does not require padding of the plaintext in order to make it work with the block size of the cipher. Besides that, it does not have a dependency between the messages which results in independency of each cipher block on the previous plaintext block.

A serious disadvantage of this mode is that the counter at receiver and sender needs to be synchronous. If otherwise, loss synchronization can lead to wrong recovery of the plaintext.

The core of this mode is the requirement of a synchronous counter at sender and receiver and could also make problems in term of implementation. Figure 1-4 shows this CTR mode for encryption.

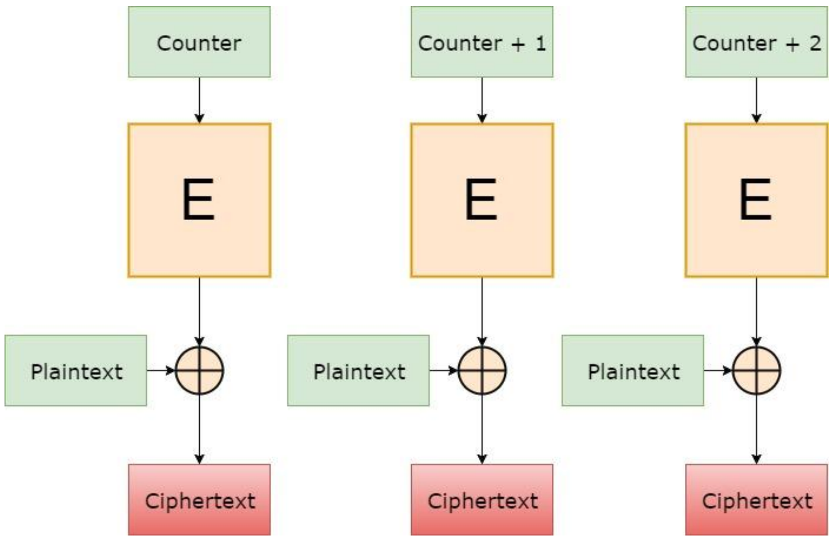


Figure 1-4: CTR block operation mode
(©Gloria Janjic, 2017)

CFB (Ciphertext Feedback):

CFB stands for Cipher Feedback Mode and works with a self- synchronizing stream cipher and does not require the plaintext to be padded to the block size of the cipher. Each ciphertext depends on all the previous plaintexts; still this mode hides patterns and repetitions very well. With this mode any number of bits can be encrypted.

The Figure 1-5 gives an overview of the encryption process when using the CFB mode.

To create a current ciphertext block, the previous ciphertext block is encrypted by a chosen algorithm and onwards XOR-ed with the current plaintext of this block. In this mode, plaintexts cannot be manipulated except removing blocks from the beginning or end of ciphertext.

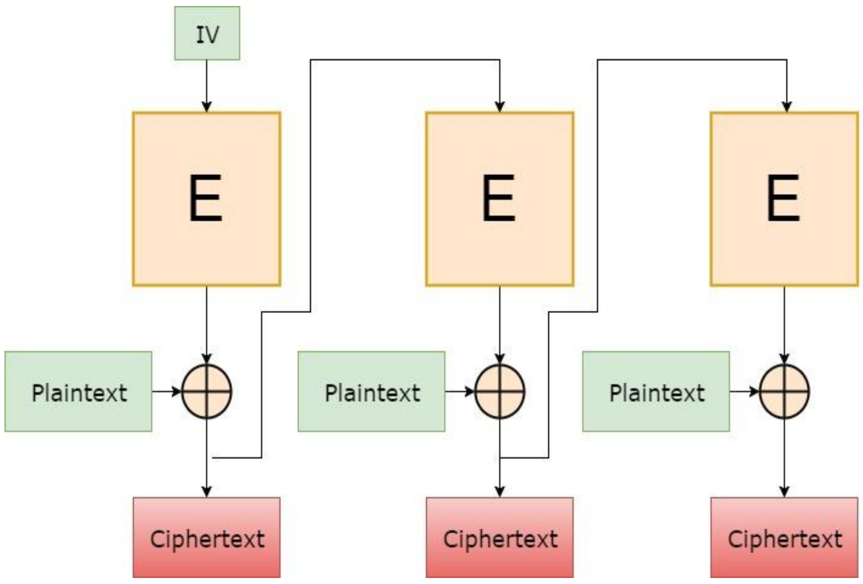


Figure 1-5: CFB block operation mode

(©Gloria Janjic, 2017)

The Figure 1-6 above shows how encryption and decryption look in further detail when thinking about the encrypted bits. As in the figure before, the mode starts with filling the initialization vector into the shift register. Afterwards the encryption algorithm is run with that input and it produces a 64 bit output. From that output 8 left most bits are chosen and XOR-ed with the plaintext byte. That output result is sent over the network and sent back to the shift register which shifts the 8 already encrypted and sent left- most bits out. This allows the next 8 left- most bits to be prepared and encrypted. With each round another character is encrypted in the same way.

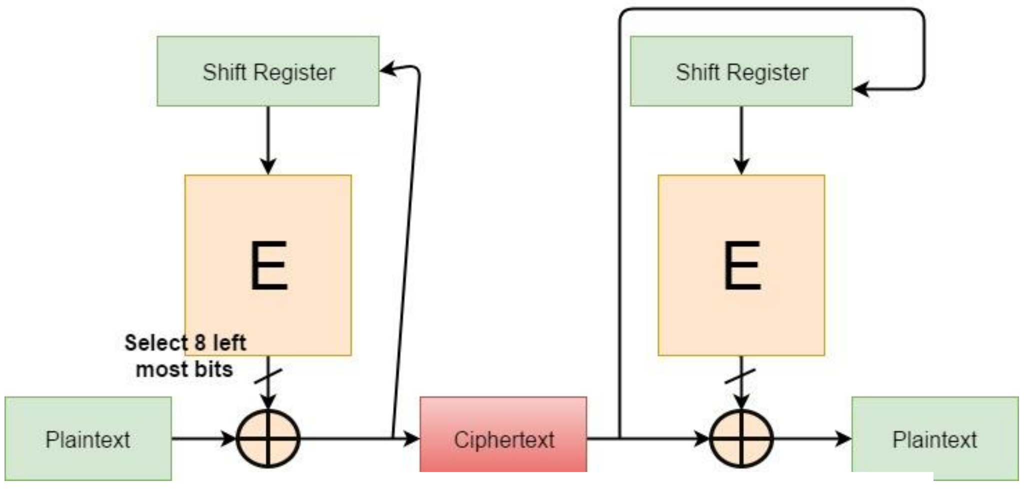


Figure 1-6: CFB encryption and decryption

(©Gloria Janjic, 2017)

<u>AES</u> [25]

AES stands for Advanced Encryption Standard and is a symmetric block cipher, which is developed by the National Institute of Standards and Technology (NIST) back in 1997. It is used in software but also hardware and is chosen by the U.S. government for protecting data. AES can be called the successor of the Data Encryption Standard also called DES, which is also a symmetric cipher but was later on vulnerable to brute- force attacks. AES is stronger in security than DES or 3DES, as it uses longer key lengths. Besides that it is ideal for software implementation as it is enabling fast encryption.

The selection process for this algorithm was transparent and open to the public, which allowed submissions of detailed and carefully chosen designs. In order to choose an AES algorithm 15 different symmetric key algorithms were competing. Out of which 5 got selected after preliminary analysis to get further tested. These analyses were made by the world cryptographic community, which includes the National Security Agency (NSA). Those implementations were tested in different languages like C and Java for reliability, speed and cryptographic attacks.

In 2000 the Rijndael cipher was chosen as the proposed algorithm for AES and AES became the government standard by 2002.

When looking into the functionality of AES, there are three different types AES-128, AES-192 and AES- 256. The number after the name of the algorithm stands for the size of the used key. Encryption and decryption works with block ciphers that are 128 bits long.

The algorithm is divided into four different functions that are repeated several times. The first step of the algorithm applies the function AddRoundKey, which works with XOR between the block and the roundkey. Later on this function is followed by SubBytes, which is looking for an equivalent in the S-Box, ShiftRow that shifts the rows and MixColumn that mixes the data in the columns. This is repeated in rounds and always in the same order. All these functions work with a matrix 4x4, which in the beginning ins filled with the plaintext and in the end shows the cipher text. [26]

1.4.3 Asymmetric key algorithm

This technique is also called public key encryption and addresses the problem of secret keys that are exchanged over the internet. Exchanging secret keys over the internet, means that an attacker that found out the secret key can decrypt a message. As it is a challenge to transport keys securely over the internet it represents a problem to symmetric key algorithms. Asymmetric encryptions have key pairs which consist out of two related keys. One is called the public key that is made freely available to anyone that wants to send and encrypt a message. The other key is the private key which is kept secret and ideally only the recipient has it. In order to decrypt a message, it is necessary to use the same algorithm as

[25] (Rouse, TechTarget, 2017)

[26] (Wikipedia-Autoren, 2017)

for encrypting by using a private key that is matching. Compared to symmetric encryption, the asymmetric algorithm is slower and requires more processing power for encrypting and decrypting. The first practical asymmetric encryption algorithm is created by Diffie and Hellman in 1976 after which RSA encryption became widely deployed. Secure transaction systems mostly rely on asymmetric encryption to establish secure channels. One famous example for that is SSL which is a protocol that provides communication security in the internet.[27]

Asymmetric cryptology makes use of one-way functions which are easy computable for all given input, but difficult to invert. Here difficult and easy relates to the computational complexity behind the algorithm. In practice this means, that is relatively easy to encrypt data with it, whereas the decryption process is made difficult.

Popular candidates for these functions are multiplication and modular exponentiation.

Computing $n = p \ x \ q$ where *p* an *q* are prime numbers is rather easy than the inverse operation which is factorizing the given number. This inverse operation leads to the known Integer Factorization Problem. The point is to find two primes, that are about the same size by knowing by *n*. Same goes for the modular exponentiation a^m mod n which can be computed efficiently, where in contrary computing the discrete logarithm given a, n and x and finding m such that $a^m = x \pmod{n}$ hold is rather inefficient.

Integer Factorization Problem:

This is known as one of the oldest problems in mathematics, knowing that many of the nowadays used techniques lead back to the ancient Greeks. It is known, that factoring large numbers is computably difficult. Nevertheless its real breakthrough was only with the public key cryptography and RSA encryption system. As mentioned before, many one- way hash functions make use of exactly that property.

Factoring an integer n means finding two positive integers *p* and *q* such that the product of those two equals the given number *n* where both integers are at least 1. In this scenario *p* and *q* are the factors of *n*. Composites are the positive integers that can be factored, primes are positive integers that are greater than 1 and cannot be factored.

To sum up, not every large number is difficult to factorize, which means that only specific numbers can be used to take advantage of the Integer Factorization Problem.

- Large prime numbers are factorized rather easy
- Large numbers are factorized rather easy
- Product of two large prime numbers are difficult to factorize

Some important, modern factoring methods would be Trial division, Pollards'rho factoring algorithm, Pollard's p-1 factoring algorithm and the random square method. At this point they will not be discussed.

[27] (Microsoft, 2007)

Diffie- Hellman key exchange: [28]

This algorithm for key exchange makes use of the second candidate of one- way functions explained above in this chapter which is *modular exponentiation*. It uses this method to raise security and make it overwhelming for an attacker to break.

- Alice and Bob agree on whole numbers *p* and *q*, such that *p* is a prime number and *q* a generator of *p*
- Both chose a random number as their personal/private key *a* and *b* that are less than the modulus p. Neither Alice nor Bob give away their private key to anyone
- Both A and B compute public keys a* and b* and share them over a communication medium

  $a^* = q^a \bmod p$

  $b^* = q^b \bmod p$

- *A and B compute the value x which for both of them has to be the same*

  $x = (b^*)^a \bmod p$

  $x = (a^*)^b \bmod p$

As the private keys *a* and *b* have not been transported over a communication medium and are important in order to calculate the value x correctly, it is very difficult for an attacker to guess the large number of x right. This approach gives high security in theory as the two users can use a public medium for communication and still communicate privately. Figure 1-7 shows the key exchange of Diffie- Hellman.

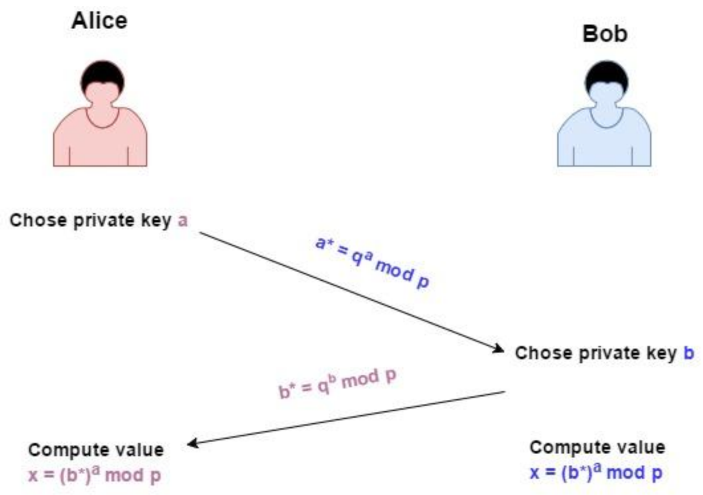


Figure 1-7: Diffie- Hellman key exchange

(©Gloria Janjic, 2017)

[28] (Rouse, TechTarget, 2017)

The biggest and main advantage of Diffie- Hellman is the Lack of authentication which is not provided in the algorithm. This enables an attacker Eve to proceed with a *Man- in- the-middle- attack* and makes Alice and Bob believe, that they are communicating with each other. The attacker has full control over the communication medium.

➔ Potential solutions for binding keys to identities [29]

1) *Authority method (Hierarchical PKI solution)*
   Some trustable authority like a certificate authority, confirms that this is indeed Bob. This can be done by recursively verifying signature until someone you trust is reached and confirms the authority.
2) *The network method (Web of trust PKI solution)*
   A friend confirms that the identity is indeed the user he claims to be. To make this possible, each participant needs to sign the keys of all their friends. Verify friends signatures until you are sure that the key indeed belongs to the new identy.
3) *White paper method (Self- Signing PKI solution)*
   Verify the signature of the other entity and consider him or her like someone you do not know anyways. It is just assumed its Bob, as one does not know anything about him anyways.

<u>*General usage of asymmetric encryption*</u>

The principle of asymmetric encryption uses trapdoor one way functions, which are one- way functions that can be inverted by using some additional information. So the encryption goes as follows:

1. Bob, the receiver, creates and distributes a trapdoor one- way function
2. The sender Alice encrypts the message by applying the public key F
3. Bob computes the plaintext by applying his private key F^{-1}:

$$\boxed{\begin{array}{c} \textit{ciphertext} = F(\textit{plaintext}) \\ F^{-1}(\textit{ciphertext}) = F^{-1}(F(\textit{plaintext})) = \textit{plaintext} \end{array}}$$

Knowing all that, it can be continued with the description of an algorithm that is working with these underlying techniques.

<u>RSA</u> [30]

This algorithm is invented by Ron Rivest, Adi Shamir and Len Adleman in 1977 and is therefore named after them. This cryptosystem is the public key algorithm which is used the most around the world. One of the reasons might be, because it is possible to encrypt messages without sharing and exchanging a secret key separately.

[29] (Rechberger)

[30] (Ireland, 2016)

Besides the power of public encryption RSA can also be used for digital signatures. The security of this algorithm is provided by the difficulty of factoring large integer numbers. This algorithm uses the Integer Factorizing problem (see above) to secure messages.

It works in the way that Alice can encrypt and send a message to Bob by using Bob's public key for encryption. In this way only Bob can decrypt the message using his private key and no other keys need to be exchanged or transported over insecure transportation channels. Besides that it can also be used in order to sign messages where Alice signs a message with her private key and Bob can verify the signature by using Alices's public key.

In order to understand the principals of encryption, decryption, digital signatures and signature verification with RSA that are going to be described next, these requirements are important:

- The number n stands for a modulus number
- The encryption exponent is referred to with the letter e
- The secret decryption and encryption exponent is marked with the letter d

The RSA algorithm generates its keys the following way:

1. Choose 2 distinct, random prime numbers p and q which are at least 20148 bits long
2. Computation of $n = p \times q$ where n is used as modulus
3. Compute Euler function $E(n) = (p\ 1)(q\ 1)$
4. Choose an integer e which is prime to E(n) and $e \neq \pm 1$
5. Afterwards compute $d = e\text{-}1 \ (mod\ E(n))$
    - Public key = (e, n)
    - Private key = (d, n)

The *encryption* using RSA works the following way, providing a step by step instruction what the sender Alice does:

1. Alice takes the public key of the recipient Bob (n, e)
2. Alice represents the plaintext as a positive integer m where $1 < m < n$ has to hold
3. Alice computes a ciphertext using the following equation $c = m^e \bmod n$
4. Alice sends the ciphertext to Bob

For the *decryption* the receiver Bob does the following steps:

1. Bob computes $m = c^d \bmod n$ by using his private key (n, d)
2. He extracts the plaintext from the computed message m

Digital signing is done by the sender Alice in this order:

1. Alice creates a message digest of the information that she is about to send. This means, that the message she is sending contains digits created by a one- way hash function.

2. This message digest is represented with an integer *m* that fulfills the requirement of *1 < n-1*
3. Alice then uses her own private key (*n, d*) in order to compute her signature which looks like this $s = m^d \bmod n$
4. The signature is sent to the recipient Bob

For the *signature verification* the recipient bob is doing the following:

1. Bob uses the public key of Alice (*n, e*) to compute an integer $v = s^e \bmod n$
2. Afterwards he extracts the message digest from this integer
3. Bob then computes the message digest of the information that has been signed in order to check if they are identical
4. Only if both message digests are identical, the signature has to be valid

1.5 Technical Requirements

The purpose of this chapter is to provide enough information about the project challenge so that the solution later on can be understood better.

1.5.1 Basic idea

The basic requirements for the license system are as follows:

1) After installing and starting the program for the first time, it has to recognize that the software is not licensed for the running host and automatically offer license activation. The user will be asked to enter the activation key, license server will be contacted and after getting the license, application will decrypt the license itself.

2) In case that licensing process has already been done, it should be checked if the license is still available on the software provider's website. Besides other information, each license has expiration date. In order to use the program's features, license should be valid and not expired.

3) It is not allowed or possible to start the software without internet connection.

4) License information (extracted from server database to local license file) should be stored in *xml format* and encrypted with the local host fingerprint. Among user, company and activation details, all product features must be stored in license file.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<License>
     <user>
          <Name>UserName</name>
          <Address>UserAdress</address>
          ...
     </user>
     <company>
         <Name>UserName</name>
         <Address>UserAdress</address>
         ...
     </company>
     <activation>
         <act-key>{CA444825-21171D5F-45044191-85E83CF8-4445EE09}</act-key>
         <act-time>2016-12-01 07::11::00</act-time>
         <due-date>2017-12-31</due-date>

     </activation>
     <features>
        ...
     </features>

</License>
```

5) Licenses should be “safe” for sending over the internet; therefore it is necessary to use a format like base64. Base64 is used to represent binary data in an ASCII string format where each base64 digit is representing exactly 6 bits of data. In order to ensure that the data remains the same without any modifications, even when sent over media that is designed to transport textual data, base64 is the encoding scheme that is used. The above license in XML format would look something like this in Base64 format:

```
WS09bbB/25A87XernWPlZTW7kLO2c9gEphIAq6iD4ju+B6FAqEcMVDGF2ho7tXB4QBeFzSZQx6w740Fb/aH8
Kuq+NE962cTu…
```

6) License should be hardware-locked

7) A resolve conflict feature should be implemented in case of hardware damages

8) Server side database should be simple for installation and management

9) Sever side database should cover all needed information for the licensing process:
    a. product information (name, list of features)
    b. company information (name, country, ZIP, address…)
    c. user information (name, company, address…)
    d. host information (disk ID, CPU ID, volume ID, host name…)
    e. license information (product, company, activation time, due date, host, password, activation key…)
    f. license usage (history of product activities)

10) Because all companies secure their network with Firewalls the only port which is (most likely) always open is HTTP and HTTPS. Therefore, web service on the server side should be developed which will respond to standard HTTP/HTTPS request for all methods (product start, product stop…)

11) Client API library, written in C/C++ which “hides” all licensing methods, should be developed. Main software should only communicate with Client API Library.

This particular software is working with a hardware- locked licensing system as mentioned above in chapter 2.1.5. To use that kind of licensing it is needed to use the hardware information of a running host. With this, a unique fingerprint key can be created for encrypting/decrypting the license. This should provide reasonable security such that the software cannot be illegally used. As hardware identification it is possible to use the disk hardware ID, Volume ID, CPU ID, hostname, MAC address of net interfaces and so on.

1.5.2 Licensing concept and domain model

In order to describe the licensing system and its functionalities, there are several entities to be discussed. The terms for the entities are going to be used through the whole thesis.

Licensing management system (LMS):

This term refers to the system implemented in this work and practical part of the master thesis. As it is not yet available on the market but used for internal purposes, it does not yet have a commercial name.

LMS owner:

The organization that implemented and developed the licensing management system (LMS) is the owner of that very same system. In this case this refers to the company for which the system was created.

Company:

When talking about the company, the user of the licensing system that is trying to protect its software is meant. In this thesis the company in most cases refers to the one for which the LMS was implemented. It is the owner of that system as well as the company that uses it.

License:

A license instance or more generally speaking a license is the right for one end- user to install a copy of the protected software. One license should be associated to exactly one end- user.

Set of licenses:

Usually a company or organization orders more than just one license which is defined with a set of licenses. In the best case, each of these licenses is later on used by one employee only.

Customer organization:

These organizations are the ones that purchase a set of licenses for their company. It is the purchaser of one license or a set of licenses for a particular product.

End- user:

These are the people that actually run the product on their computer. License instances are used by one end- user.

The entities described above are used as domain entities to create a model. This model contains a few “one to one” as well as “one to many” relations. A domain model is used to describe a concept which analysis data as well as behavior. This model is shown in Figure 1-8.

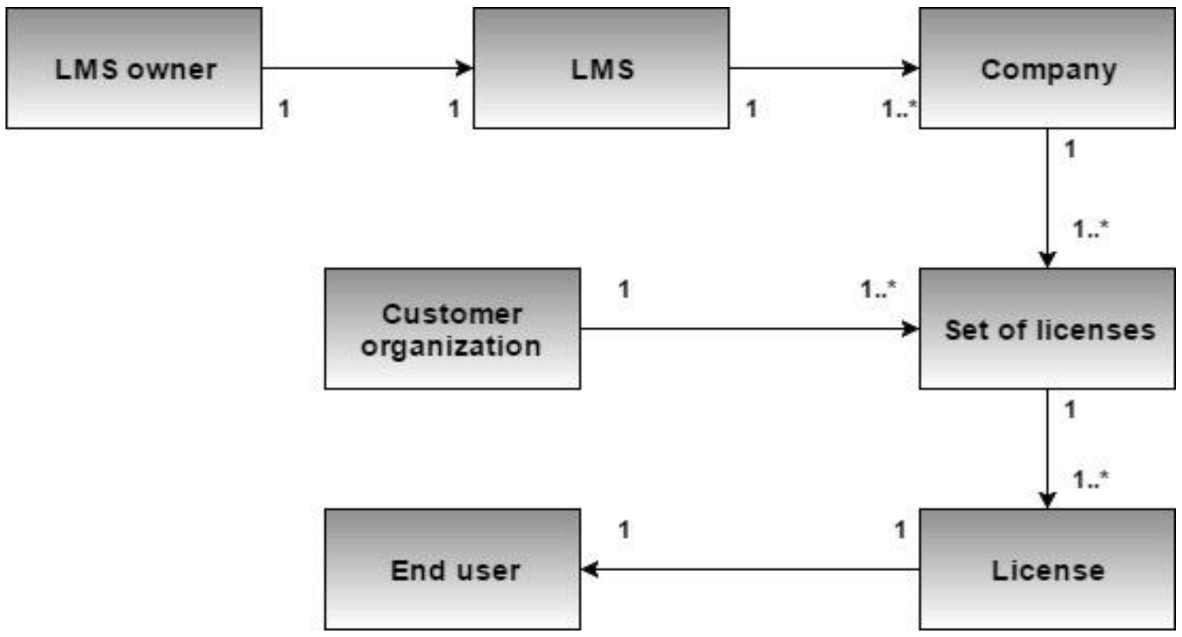


Figure 1-8: Domain model with relations

(©Gloria Janjic, 2017)

When knowing what relations there are between entities, it makes it easier to understand the whole concept for further work and elaboration. Figure 1-8 containing the domain model should be rather simple to understand; nevertheless it will be explained now.

As discussed before the LMS owner is the developer or owner of a licensing system (*one to one relation*). This licensing system could be used for commercial purposes or like in this case for internal usage. It protects software by managing licenses and storing data about usages.

This licensing system also referred to as licensing management system (LMS) can be used by one or many companies which want to protect their intellectual property (*one to many relation*). In the particular case of this work, the owner of the LSM and the company that uses it, are the same.

A customer organization is the company that wants to have access to the protected software and requires a set of licenses which can later on be used among employees (*one to many* *relation*). This organization can order a set of license at one point and order more sets afterwards if needed.

The set of licenses consists of multiple licenses which are sometimes also referred to as license instances (*one to many relation*). They allow the usage of the protected software of a company.

Each license instance should be used by only one end user (one to one relation). End users are persons that work with the software.

1.5.3 Solution overview

The thesis consists out of three components which are the HTTP web service, client access API library and a license manager. The Figure 1-9 shows the interaction between those parts in a brief overview that makes it easier to understand the upcoming, more technical explanations.

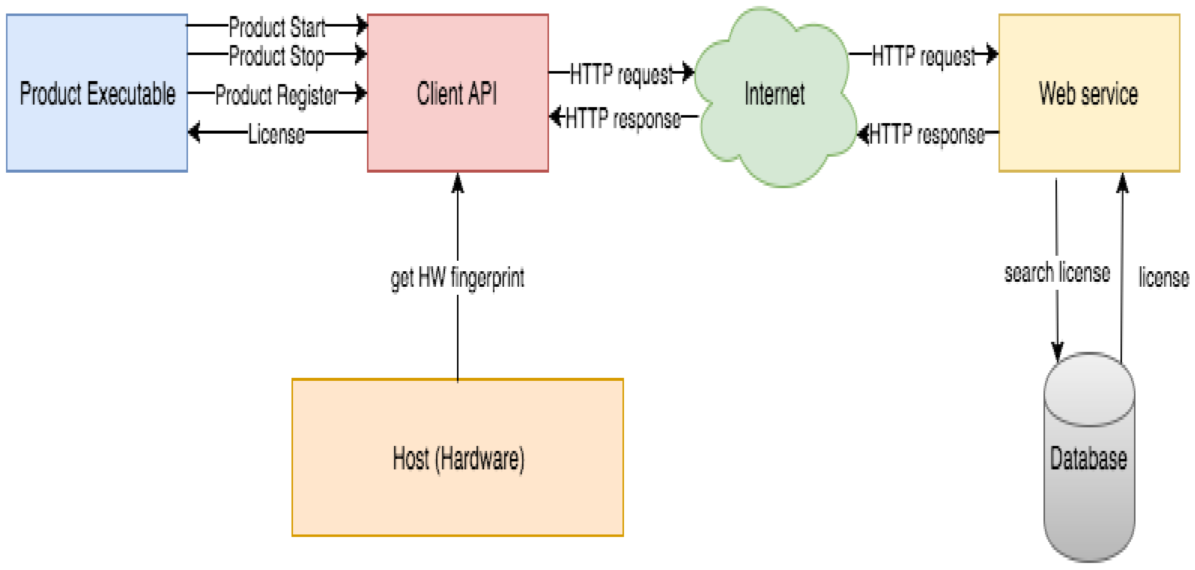


Figure 1-9: Interaction between Web service, Client API and database

(©Gloria Janjic, 2017)

As one can see the key to making the system work is having an overview of the general picture and the functionalities of the different components. In order to start the software product for a client, a fingerprint needs to be created. This could be done by taking hardware and software information about the host's computer. Before the product executable is ready to start, the license needs to be found on the database of the server side. This part can only happen after the state of Product Register is finished. The Client API is working on the customer's computer and is communicating with the web service by the Internet. In order to make this possible, HTTP requests and responses are used to get information from the server or database to the client's API or store information on it. The communication between the web service and the database is crucial, as the web service is searching for licenses in the local database and getting licenses back when they are found. When a license is successfully found, it is first sent to the client API and then the product executable can be used properly. An assumed workflow of the implementation is shown in Figure 1-10.

Application:

- Input: Product ID, Method = Register
- call client API
- Output: error code, encrypted license
- get hardware fingerprint()
- decrypt license using hardware fingerprint()

Client API:

- Input: Product ID, method = register (from application)
- ask user for license Key (from User),
- get hardware Info (disk ID, CPU ID ..) – make hash (hardware fingerprint)
- call server(HTTP) – prepare key for exchange …
- Output: error code, encrypted license

Server Web Service

- detect new request is coming (product ID, method, license key, hardware fingerprint)
- check in database for product ID
- check for license key
- assign license to host with hardware fingerprint
- encrypt license
- send back

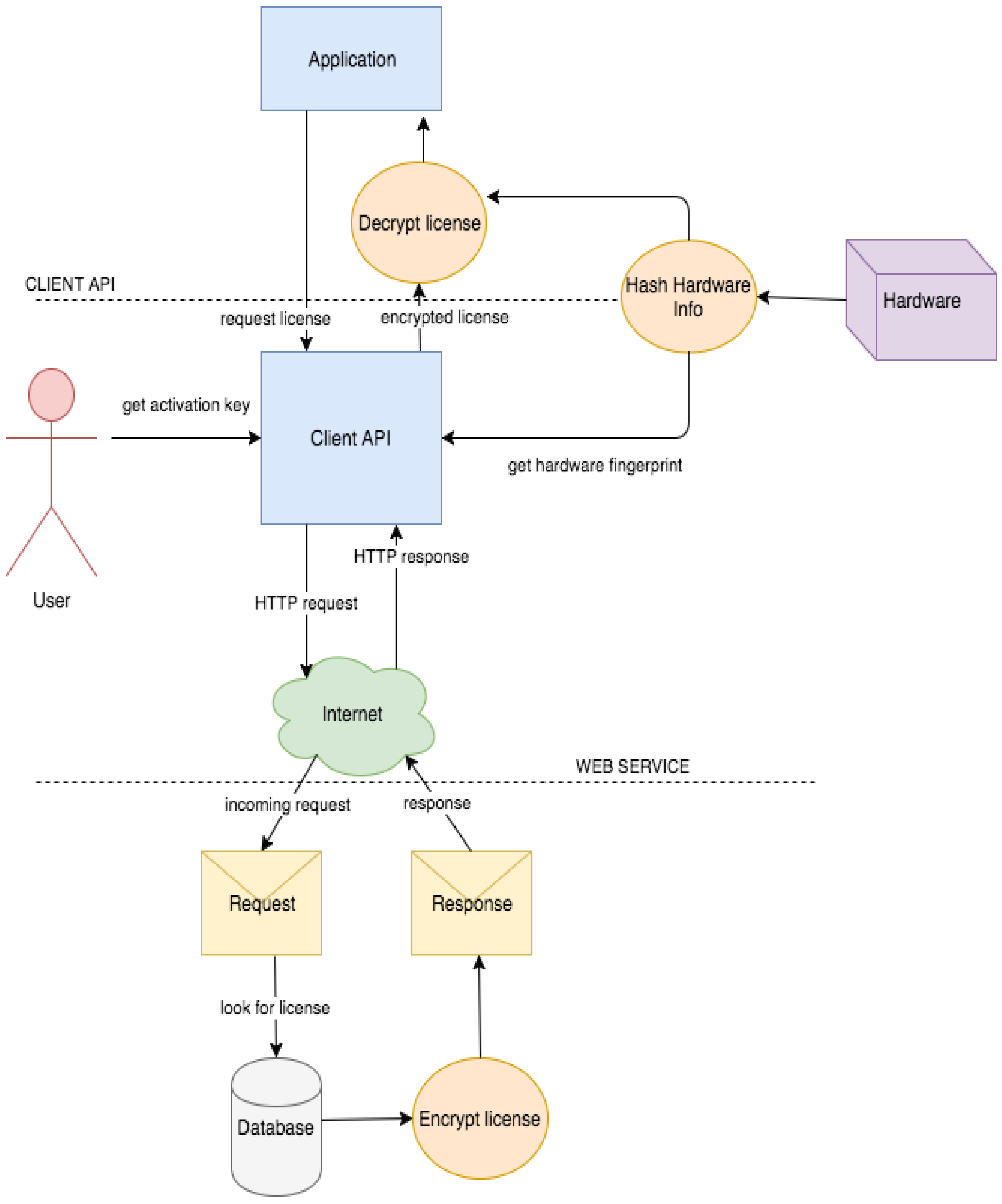


Figure 1-10: Assumed workflow of the project implementation

(©Gloria Janjic, 2017)

1.5.4 Database content

The sever side database should cover all needed information for the licensing process. This covers information about the product type and the list of features it comes with, detailed information about the company that is using it, license information including the dates of activation and expiration, additional information about the host and the user and a log of the history how and when the product is used. The Figure 1-11 shows the assumed database content which is going to be implemented and used for the LMS.

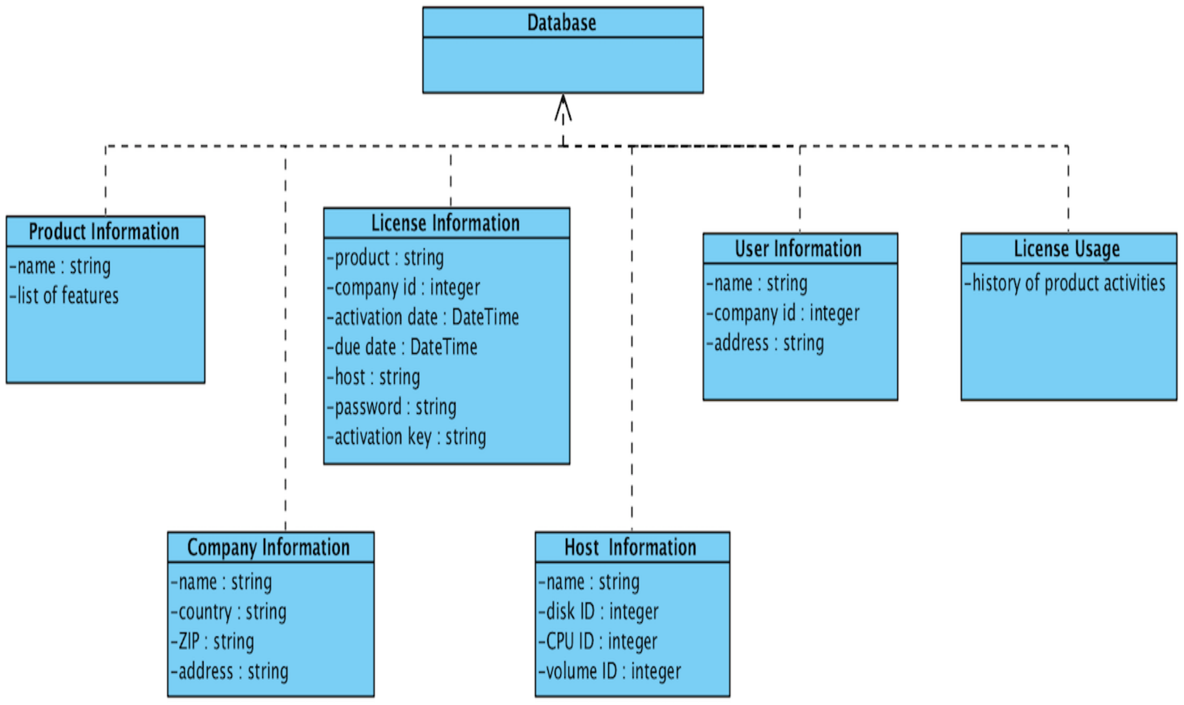


Figure 1-11: Server- side database content

(©Gloria Janjic, 2017)

1.5.5 Method overview

After careful research and consideration, it was decided that the following design of actions are the best way to cover all existing scenarios that need to be handled by the implementation. These are the following actions identified: Product Register, Product Resolve Conflict, Product Start and Product Stop.

Method	Input	Intermediate	Output	Event Description
product start	company ID, application ID	host fingerprint,	error code, encrypted	Software Product is started at user host

		activation key	license	computer
product stop	company ID, application ID, encrypted license	host fingerprint	error code	Software Product is stopped at user host computer
product register	company ID, application ID	host fingerprint, activation key	error code, encrypted license	Software Product requires registration
product resolve conflict	company ID, application ID password	host fingerprint, activation key	error code, encrypted license	Catastrophic case – resolving conflicts

Table 1-1: Method overview of project

The Figure 1-12 below shows a very simplified workflow for these four most important methods implemented. Green arrows point to the follow up task which is called after a correct execution. Red arrows point to tasks which are executed after a failed execution of the task.

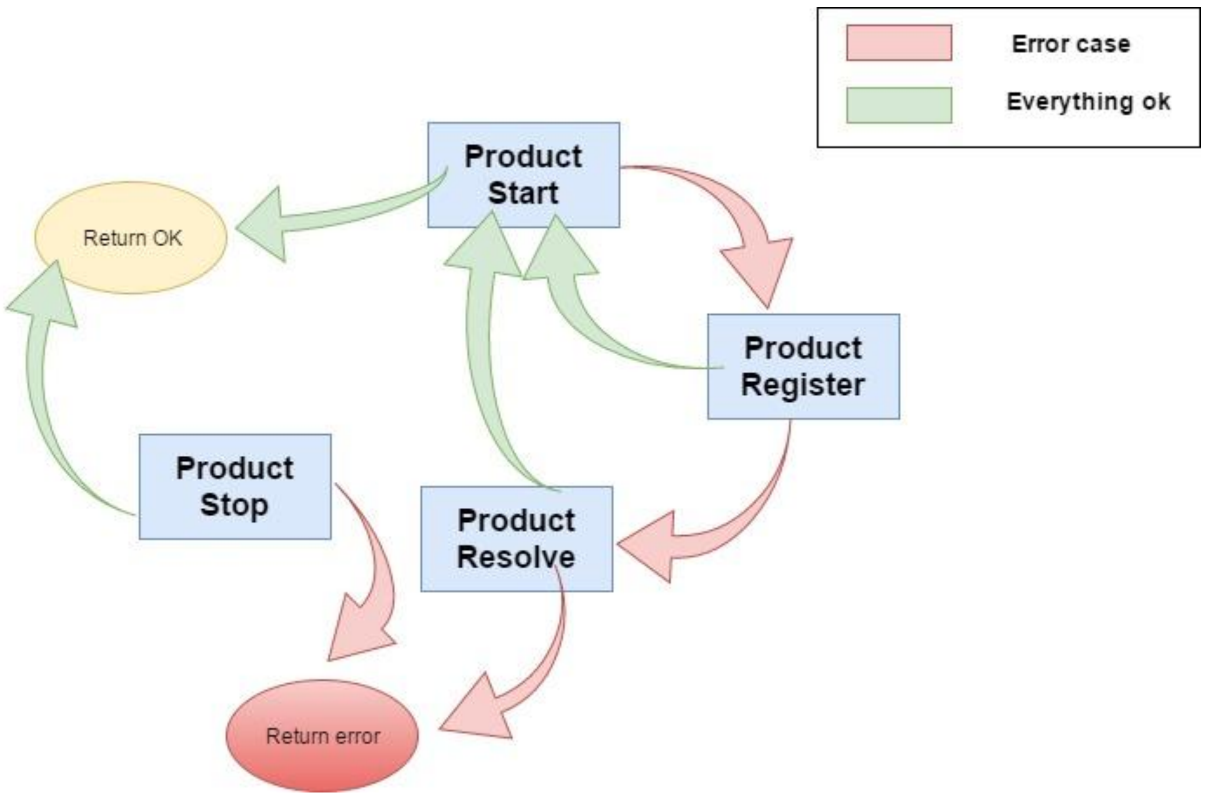


Figure 1-12: Simplified workflow of methods

(©Gloria Janjic, 2017)

If the method Product Start succeeds, a OK message is returned- if it fails the following method called is Product Register. In case the method Product Register is successfully executed Product Start is called, if not it goes on to the method Product Resolve. After successfully executing the method Product Resolve, Product Start is called. If not, an error message is returned. Product stop is a silent method which means that it does not throw any error messages and it simply stops the program. Otherwise it returns an OK message.

1.5.6 Expected workflow

As the following four methods will be mentioned rather often during this thesis, their workflow will be discussed in the following descriptions. The methods take both client and server side into consideration.

Method: Product register (License activation)

[client] Firstly on the client side it is asked for a license key which can also be referred to as the activation key later on.
[client] Afterwards a unique fingerprint of the local host is collected. This fingerprint is later on used as a secret key.
[client] Next an activation request is sent for an encrypted license from the software provider website using an activation key, application ID and host fingerprint.

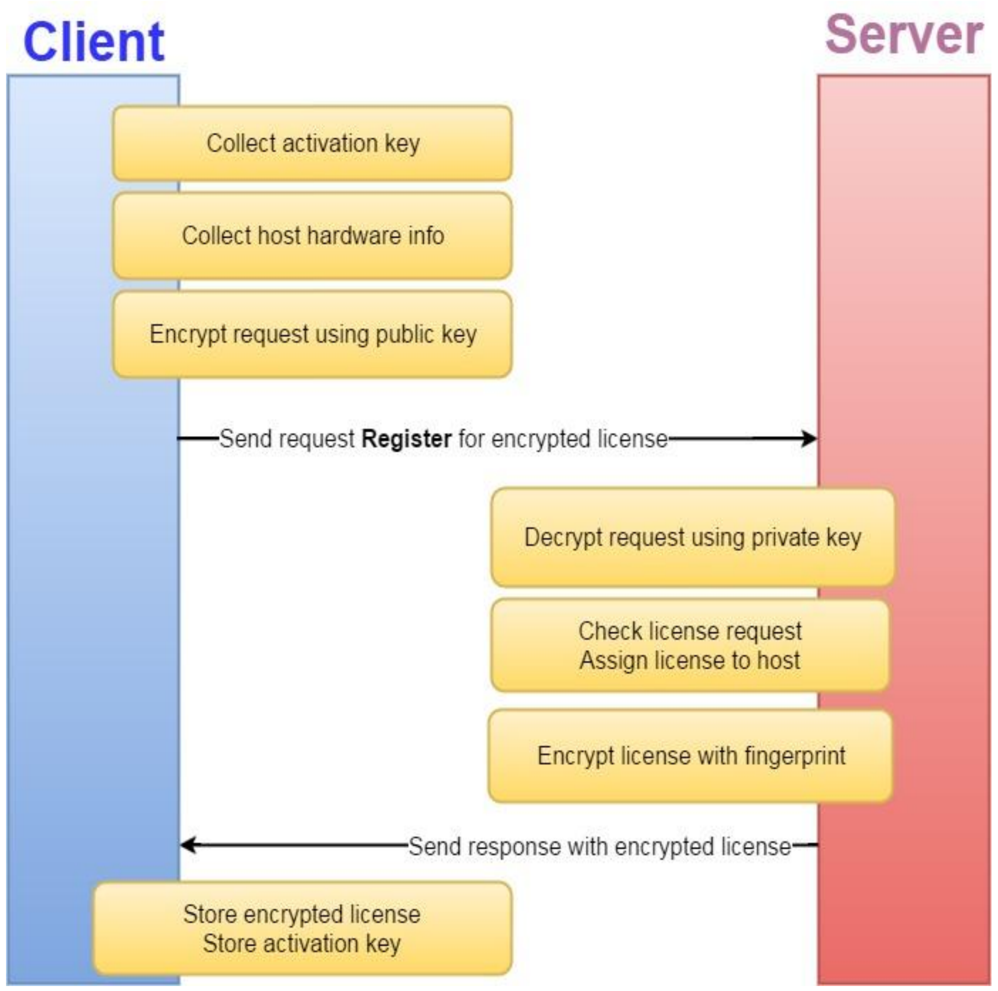

Client
Server
Collect activation key
Collect host hardware info
Encrypt request using public key
Send request **Register** for encrypted license
Decrypt request using private key
Check license request
Assign license to host
Encrypt license with fingerprint
Send response with encrypted license
Store encrypted license
Store activation key

Figure 1-13: Product register method on server and client side
(©Gloria Janjic, 2017)

[server] The first task of the server is to lookup the key and application ID in license database

[server] In case the license is already assigned a conflict error is returned

[server] Otherwise the license is assigned to the host

[server] The encrypted license is returned and the server is “done” for this step

[client] The client gets the response with encrypted license

[client] Afterwards the encrypted license and activation key are stored at the running host

Figure 1-13 describes the communication between Client and Server during the excecution of the method “Product Register”.

Method: Product resolve conflict (License conflict)

[client] Firstly on the client side it is asked for a license key and an additional password. [client] Like in Product Register collect unique fingerprint of the local host which will be used as a secret key

[client] Send a request to resolve conflict license to the server

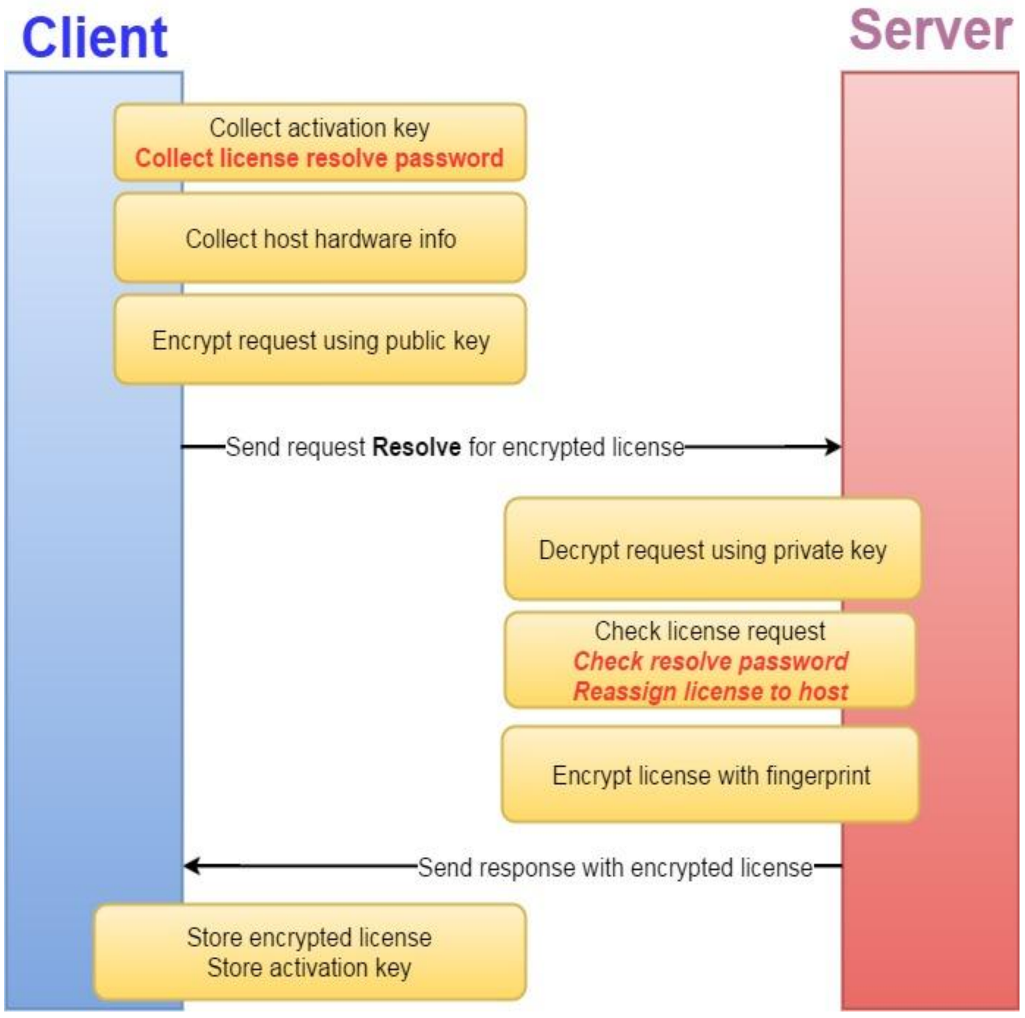

Client
Server
Collect activation key
Collect license resolve password
Collect host hardware info
Encrypt request using public key
Send request Resolve for encrypted license
Decrypt request using private key
Check license request
Check resolve password
Reassign license to host
Encrypt license with fingerprint
Send response with encrypted license
Store encrypted license
Store activation key

Figure 1-14: Product resolve conflict method for server and client

(©Gloria Janjic, 2017)

[server] Again the application ID and the key are firstly looked up in the license database.

[server] In case the password is incorrect an error is returned.

[server] Otherwise “move” (reassign) the license from the old host to the new host.

[server] The encrypted license is returned and the server is “done” for now.

[client] The clients gets the response with the encrypted license and extracts it.

[client] Afterward the encrypted license along with the activation key are stored on the the running host using “Windows Registry”

Figure 1-14 describes the communication between Client and Server during the execution of the method “Product Resolve Conflict”.

Method: Product start (standard usage):

[client] Get encrypted license and activation key at running host

- if fail go to license activation (license not found)

[client] Collect fingerprint of the local host and decrypt license using a fingerprint

- if fails got to license activation (license corrupted)

[client] Check if license is not expired

- if license is expired go to license activation (license expired)

[client] Afterwards send a request to the server that the product is started. Additionally the client sends the encrypted license along with the application ID and fingerprint.

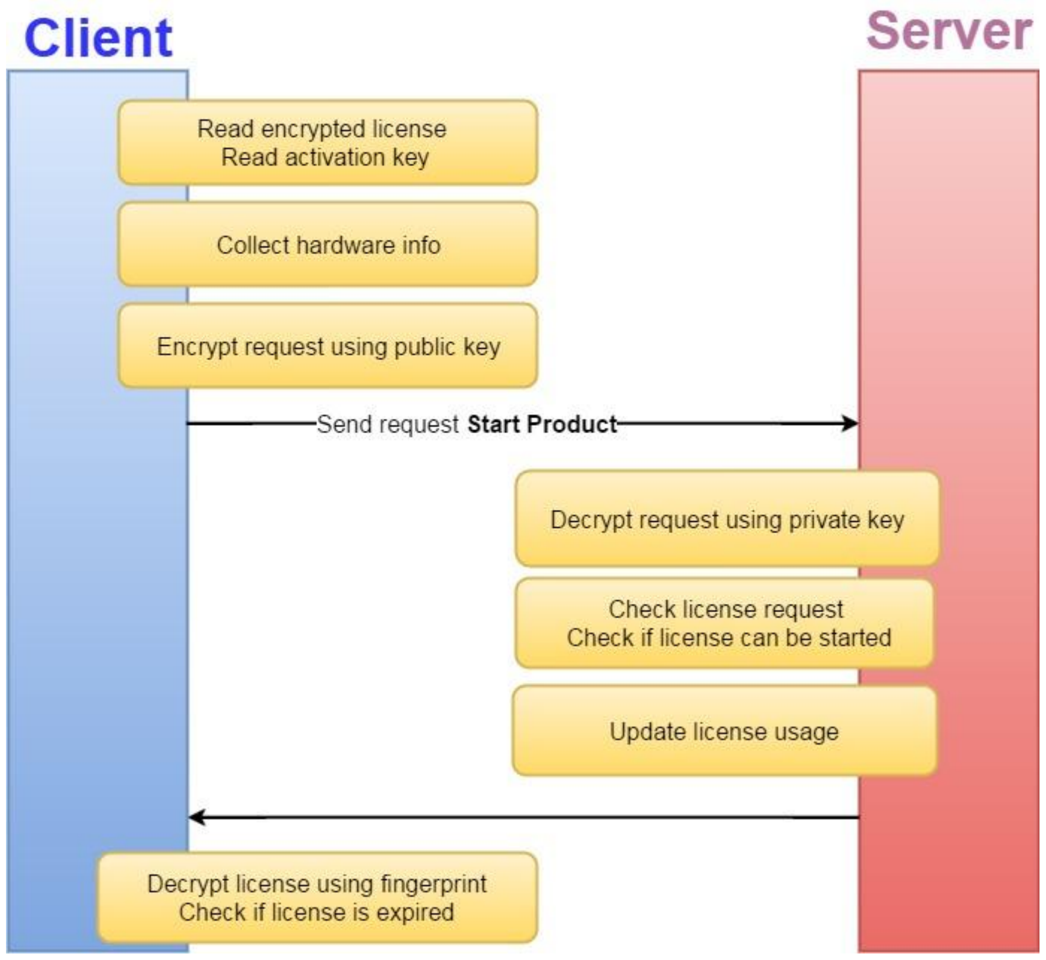

Client
Server
Read encrypted license
Read activation key
Collect hardware info
Encrypt request using public key
Send request **Start Product**
Decrypt request using private key
Check license request
Check if license can be started
Update license usage
Decrypt license using fingerprint
Check if license is expired

Figure 1-15: Product start method for server and client side

(©Gloria Janjic, 2017)

[server] The server looks up the key and application ID in license database.

[server] In case the license is not assigned to given host a error message is returned.

[server] Afterwards the information about the program start is stored (date, time…)

[server] Finally the server sends back an encrypted license to the client

[client] Once the response is received with the encrypted license it is checked

- if problem with internet- stop
- if license not found (maybe deleted on server because it is stolen or …) go to license activation;
- if license conflict (checkout to other host in the meantime) go to license activation

[client] The client decrypts the license using a fingerprint

[client] If everything is okay, the program should continue normally.

Figure 1-15 describes the communication between Client and Server during the execution of the “Product Start”.

Method: Product stop (standard usage)

[client] Send request to software provider website that product is stopped, send encrypted license, application ID and host fingerprint

- if problem with internet, ignore problem
- if license not found, ignore problem
- if license conflict, ignore problem

[server] The server looks up the key and application ID in license database.

[server] In case the license is not assigned to the given host a error message is returned.

[server] Afterwards information about the program is stored (date, time…)

[server] Return OK

[client] The client gets a response with encrypted license.

[client] The license is decrypted and checked.

Figure 1-16 describes the communication between Client and Server during the execution of the “Product Stop”.

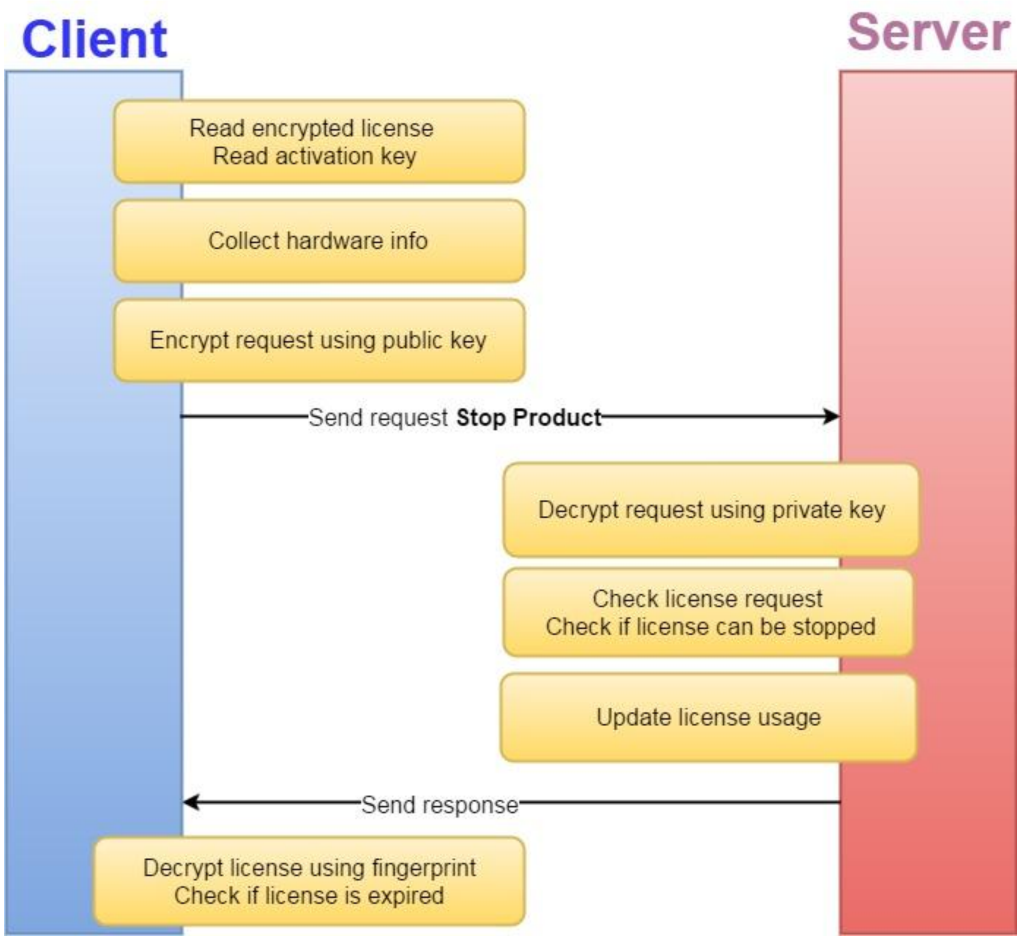

Client
Server
Read encrypted license
Read activation key
Collect hardware info
Encrypt request using public key
Send request **Stop Product**
Decrypt request using private key
Check license request
Check if license can be stopped
Update license usage
Send response
Decrypt license using fingerprint
Check if license is expired

Figure 1-16: Product stop method for client and server side

(©Gloria Janjic, 2017)

1.5.7 Predicted error cases

The following error cases can occur in the licensing system implementation:

Error Code	***Description***
ERROR_OK	Execution succeeded
ERROR_USER_CANCEL	Execution canceled by user
ERROR_NO_INTERNET	Server cannot be contacted
ERROR_LICENSE_CORRUPTED	License cannot be decrypted using host fingerprint
ERROR_LICENSE_EXPIRED	License exists but it is expired
ERROR_LICENSE_CONFLICT	License is valid but it is currently used by another host
ERROR_LICENSE_NOTFOUND	License cannot be found at server side
ERROR_LICENSE_NOTACCEPTABLE	License is still valid but local copy is outdated

Table 1-2: Predicted error codes

1.6 Solution requirements

This part of the thesis should give an overview of the desired functionalities of the system in general but also some requirements analyzed from different angles. It is distinguished between general, security, licensing and design requirements. Each of these requirements categories is consecutively numbered and marked. The initials “GR” stand for the general requirements”, “SR” for security requirements, “LR” for licensing system requirements and “DR” for design requirements. All four tables contain the names and numbers of the requirements, as well as additional columns for the explanation and reason why it was needed.

1.6.1 General requirements

Some of the general requirements which are made by the company using this licensing system in the future are described and reasoned in the following Table 1-3. These requirements are important as they can increase profit as well as making the protected software more valuable.

	Name	Explanation	Reason
GR1	License instance overview	The company using this licensing system requires having detailed information about each sold license. In addition it is required to have an overview of the software usage.	Creating this system should primarily help preventing software piracy out of economical reasons. Additionally, the usage overview can be used for further marketing strategies.
GR2	Easy distribution of licenses	Management and distribution of end user licenses should be easy for the software company. In the optimal case an activation key should be exchanged by secure media.	In order to make this system work, the owner requires an easy way of adding new license instances. External help from the developer’s of the licensing system should not be needed.
GR3	No interfering with protected software	The protected software should not in any case be interfered with more than it is absolutely necessary. No further restrictions concerning the development of the software should be implied by the usage of the licensing system.	The company needs secure way of protecting its software without having to maintain the protecting system all the time.
GR4	No non-compulsory protection	For now, the system should not allow non protected software usages at all. Nevertheless, it should be able to extend the implementation for that in the future if necessary. Later on it might be necessary to enable and disable the	Some of the company’s most trusted users might not have to go through all the protecting process of the software. When the licensing system actually gets used, it should be possible to

		protection of the software.	adapt that and make it possible for known and trusted customers to use the software without any inconveniences. Anyways at the moment there is no such feature, as it is not part of the company's requirements yet.

Table 1-3: General requirements of the system

1.6.2 Security requirements

Table 1-4 shows the analyzed requirements concerning the security of the whole system. They are related to the licensing program, securing it against attackers that might want to bypass the system.

	Name	Explanation	Reason
SR1	Protect owner's intellectual property	As mentioned before, the most important thing is to protect the software and its contained intellectual property by any means. This means, no new user can make usage of the product without having a valid license.	The reason for this is similar as to any other copy-protection. Program features should be protected from software piracy making it difficult for an intruder to gain knowledge and make use of the program.
SR2	Restrict multi usage of license	Unintentional usages of one license instance on multiple computers should be prevented. Exceptions should be made only under some conditions and given circumstances. For example when a computer is no more used.	The reason for this kind of protection is, that the company does not want to sell only one license when there is multiple end- hosts working with it. This would mean fatal loss of income.
SR3	Distinguish between Product versions	It should be enabled to differ between software product versions. A valid license should only be used for the purchased product version. Besides that, for a given license the system should not leak any information about other products. Moreover, different major product versions are treated like separated entities.	There are several types of products that are sold by the company which differ in prices and functionalities. Obviously users should be able to use the products that they purchased, but not other ones which could again lead to software piracy.
SR4	Protect user information	Any information about the company and its end- user must be protected when shared over an insecure communication channel.	Any leak of information about the company or its users can have all sorts of negative

		This also has to apply to information which is stored by the licensing program. All the information needs to be handled securely and transported in a secure way	outcome. An attacker should not have access to this sensitive information as they are all very valuable.
SR5	Internet connection needed	One of the company's requirements is that it should not be possible for an end- user to use the program without internet connection. In this case the program should be either stopped or not even started. This is a design decision made after some consulting and meetings with the company.	The company decided this due to security issues. In order to allow software usage without internet connection, global information about license usage would be lost and it would be necessary to store all licensing information at local host. As local stored information are vulnerable to attacks, it is decided to allow usages only with a stable internet connection, which implies global storage of information (cloud).
SR6	Use cryptographic techniques	It is required to use cryptographic techniques for encryption and decryption which are proven to be secure. Most of those techniques can be found in chapter 1.4.	Cryptographic methods are essential in the licensing system, as local information stored at local host and also messages sent over the internet must be encrypted.

Table 1-4: Security requirements of the system

1.6.3 Licensing system requirements

The following requirements shown in Table 1-5, are analyzed from a licensing system point of view, from the modeling of license system to security issues directly concerning the implementation of the system.

	Name	Explanation	Reason
LR1	Model of license instances	The licensing system should build and store license instances according to the existing framework and system working on the user company's site. Besides maintaining sold licenses, it should also store associated information about the customer's company as well as data about host and products.	As the main purpose of the license manager is to maintain license information, all relations and instances need to be geared of the customer's company domain model explained in chapter 1.5.2.
LR2	Data	License instances need to be recorded	For this system to be useful it

	dependencies of licenses	with all associated information that is needed for further tasks. The application is responsible to extract all that relevant data. Data about the license: - A unique ID for each license - Which product is protected with a license - Who is the license owner (company, host) - Optional password for the license instance when trying to use it on another computer - Expiration date of license, so it can be deactivated - Usage Log – like information about how and when the license is used Owners of license: - This data is collected from the end- users who purchased licensed products and run them on their host - Information about the end-users - name, location and description - Information about the host running a licensed product including all needed data for the host fingerprint Types of Programs: - The company which use the software licensing system, develops more than one product so each one has to be uniquely identified - Products are identified by ID, name and descriptions - Besides that all the activated and supported features of a product need to be maintained	is needed to have all that information to ensure the functionality and security. All that data needs to be stored for the actual verification process of a license instance
LR3	Edit database information	Not only does the license manager need to store and extract all the data mentioned in License Requirement 2 (LR2), but it also	As well as requirement LR2 this one counts to the main basic operation without which

		needs to be able to add and modify this data. All data associated with one entity of license should be editable and again safely stored so it can be used later on	that licensing system cannot work properly. All license data needs to be changed at one point or another.
LR4	Data format for license	As license information is very sensitive data, it has to be stored in a secure form. It is important to keep this in mind when storing and extracting the license data. The probability for the execution of successful attacks that leak information about the system itself or data of users, should be minimized.	It is crucial to make the license format secure in terms of IT – Security, as this protects safety of data and reduces vulnerability to attacks of all sorts.
LR5	Secure connection	In addition to secure storage of license information, any communication over the internet has to be encrypted and protected. Encryption mechanisms are required in order to make that possible. Despite any high security firewalls, potential attacks need to be avoided the best way possible.	As communication over the internet is a base functionality of this system, it is required to manage them securely. If this is not done correctly, the user's company information can be leaked and used for bad purposes.
LR6	Registered hosts	It was decided by the company, that the licensing system works Per Host rather than Per User. Meaning, that LMS is taking information about the host hardware rather than checking user information (which is questionable). The risk of unintentional usages by third parties on a registered host should be covered by the owner of a registered host computer. This could be done by protecting the access to computer with a personal secure user password.	The reason for having a host-protected licensing system is that it is considered a way more secure. As the content of the software is highly valuable, it should not be allowed to run it simultaneously on more than one computer.
LR7	Online activation based	In chapter 1.3.1 the differences between local installed and online activation based schemes were discussed. The company requires the system to be based on online activations communicating with an activation service (web service) rather than keeping all information locally on host.	Again the reason for this decision is, those online activation based systems are more secure in the owner's point of view. From a client's point of view, it might be considered less secure because information sent over the internet is always vulnerable to attacks. Nevertheless the main goal and functionality of this system

			is to restrict multiple usages of one license on different hosts. Therefore it is decided to make a communication with an implemented server over the internet.

Table 1-5: Licensing system requirements

1.6.4 Design requirements

The following requirements were detected in terms of designing the licensing management system. All design requirements are listed in Table 1-6.

	Name	Explanation	Reason
DR1	Client side programming language	It is required by the company, to implement the client side using C/C++ as this is the language used for the part where the licensing system needs to be added.	The reason for this decision is, that it is necessary to integrate that part of the implementation in the already existing project developed in C/C++.
DR2	Shared database schema	It is required that one instance of relational database and the same database schema will be used for all end- users, rather than implementing separated relational databases and schemas for each one and providing complete isolation.	The reasons for using shared database and shared database schema is that it needs less resource, avoids difficulties in implementation and reduce maintenance effort of the database system.
DR3	Web service for node- locked licensing	The company requested a web service which enables node- locked licensing and can communicate with the C/C++ implemented client side. The implementation of web service and chosen programming language should enable future extensions.	The node- locked licensing solution is chosen because of security issues. Software piracy due to multiple license usages should be minimized in that way. This is also one of the core topics of this thesis and enables a controlled licensing usage.

Table 1-6: Design requirements of the system

2 Functionality, architecture and user Interface

As discussed before (see Chapter 1.5.4) the technical requirements show four main tasks that are identified for the licensing system licensing:

- Product Start: Action at application start. In a standard case, the license is already installed at host. The check of license will be done (local and remote).
- Product Register: The main task of licensing. Changing the state of non-licensed product into licensed product.
- Product Resolve: Resolving license conflict. This case can happen when user wants to start application using license which is assigned (node-locked) to another host. In general this is not allowed, however using secret password the case can be resolved.
- Product Stop: Action at application end.

In Figure 2-1 one can see the designed workflow. “Product Start” will execute “Product Register” if the local license is not valid. “Product Register” will execute “Product Resolve” code if there is any license conflict. This is the main stream of the workflow implemented in the solution.

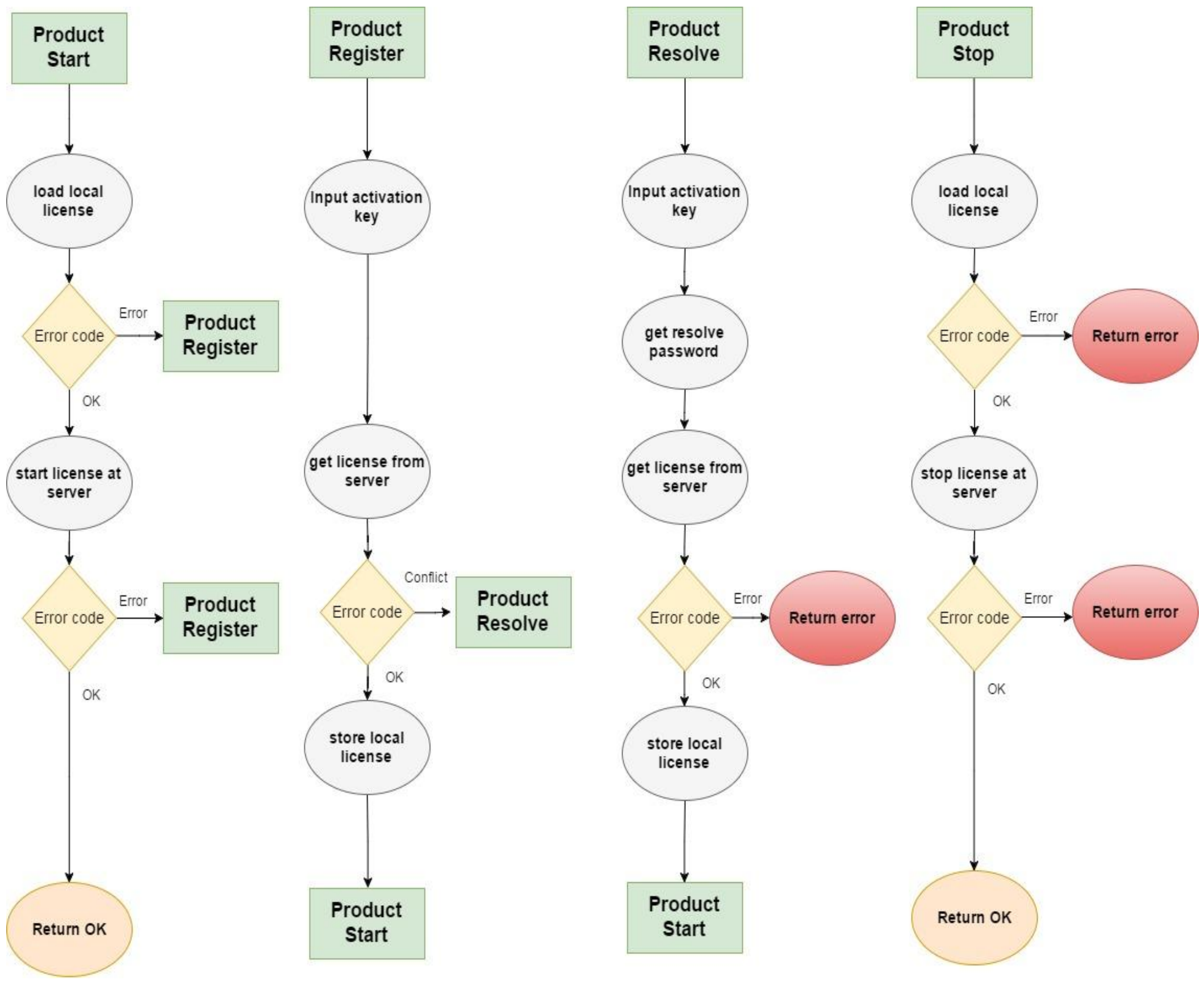


Figure 2-1: Designed workflow

(©Gloria Janjic, 2017)

2.1 Software components

Two software components are identified and developed in this project.

The first component is **ClientAPI** library, developed using standard C++ language. The purpose of this API is to “hide” the implementation of license protection at client side and to provide a simple set of functions to be called from the main program. ClientAPI is directly linked with an application executable running at the user host, loaded in the same process. Library is linked as a static library, which is chosen due to higher level of security protection against cracking attack than the dynamic library.

The second component is the server side of implementation, a **Web Service**, developed as standard a *Node.js* server application. *Node.js* platform is chosen due to its rich set of functionalities.

In both components it is necessary to include Client-Server communication, encrypting and wide operating system calls. In addition, a GUI (graphical user interface) is required at client side and SQL database is needed at server side.

It is required that this solution is portable at all standard operating systems. Therefore only portable third party components have been included in the project. The next table shows chosen components on client and server side – all of them portable to all standard platforms (Windows; Mac; Linux). For the purpose of this project, the implementation of the client side is developed for Windows only, but porting to other platforms is expected to be a rather straight forward task.

Table 2-1 shows all library versions and licenses, needed on client and server side for different components of the implementation.

Component	***ClientApi (Client Side)***	***Web Service (Server Side)***
Coding	Standard C++	Standard JavaScript
Operating System ***GUI***	wxWidgets library version 3.0.2 wxWidget license C++	Node.js 7.9.0 (module OS) MIT- License JavaScript
Client Server Communication	Lib curl version 7.53.1 MIT/X derivate license C	Node.js 7.9.0 (module HTTP) MIT- License JavaScript
Encryption	Crypto++ version 5.62 Boost Software License C++	Node.js 7.9.0 (module crypto) MIT- License JavaScript
Database	N/A	Node.js 7.9.0 (module sqlite3) MIT- License JavaScript

Table 2-1: Libraries used for client- and server side

2.2 Third part libraries – client side

The Client API library is developed using standard C++ language and libraries. For specifics tasks third part libraries are used: wxWidgets for OS and GUI, LibCurl for client/server communication tasks and Crypto++ for cryptographic tasks. All third party libraries are portable to all standard platforms.

Full source code of all third party libraries at the client side have been downloaded and compiled at Windows host using Microsoft Visual Studio 2010 C++ Compiler.

wxWidgets is a free, open- source and mature C++ library that lets developers create applications different platforms including Windows, Mac OS X and Linux.

Curl is a multiprotocol client- side URL transfer library that is free, highly portable and working on several platforms. Besides that it is compatible with IPv6, supports a complete documentation for usage and is therefore a widely used C- based library. [31]

Crypto++ [32] on the other hand is a free C++ library of cryptographic schemes and is written by Wei Dai. It includes functions for stream and block ciphers, message authentication codes like VMAC, HMAC, GMAC, CMAC, CBC- MAC, key agreement schemes like Diffie- Hellman, methods for public- key cryptography and hash functions like SHA-1, SHA-2 . Besides that it includes additional features like the generation of pseudo random numbers, prime number generation and verification and some non-cryptographic algorithms. It offers a User Guide and supports compilers like VS2003- VS2015 and Xcode 3.0 – 7.3 to mention those that are relevant to the working environment in this thesis.

In order to install the library in the right way, it needs to be extracted in a folder and build from source.

2.3 Third part libraries – server side

The Web Server, which is the server side of implementation, is implemented using *JavaScript*, *Node.js* and npm, which with its rich set of modules covers all necessary features. As *Node.js* is running on all platforms, this component is already fully portable.

Node.js is an asynchronous *JavaScript* runtime and used to build network applications. It is efficient and lightweight and its package ecosystem npm (node packet manager) is the largest one of open source libraries. Npm can be used to install code but also to manage sharing and distribution of code. [33]

More details about all the used *Node.js* modules can be found in Chapter 3.3.1.

[31] (Stenberg)

[32] (Dai)

[33] (Joyent, 2017)

2.4 SQL database

For the purpose of this project SQLITE database is chosen. Sqlite3 is the most used embedded SQL database engine in the world, which unlike other SQL database does not have a separate server process.

2.5 Development environment

In order to compile the project source code on windows host, some tools needed to be installed.

Compiler: Visual Studio 2010 [34]

In order to program and debug Visual Studio 2010 was used on a Windows host. Visual Studio is an IDE from Microsoft, used to develop computer programs, web/mobile apps and web sites/services. It is a detailed and efficient environment to produce code and contains a Visual Studio Debugger which allows you to find and debug local and remote programming bugs. Microsoft offer official C++ compilers which are connected to Visual studio, called Visual C++. The version of Visual studio 2010 contains a visual C++ 10.0 compiler

Project maker: CMAKE version 3.8.0 [35]

CMake was created by Kitware and is used to make projects and create dependencies between files and directories. It is an open- source program that enables to build and control the compilation of a software. It generates makefiles and workspaces that can be used afterwards.

Subversion Client: Tortoise SVN

Tortoise SVN is the subversion client that is used in the project to safely store and distribute the source code project data among several host computers.

2.6 Installation and usage

The user of this development is a software company, which wants to license its own applications. In order to properly use this licensing system, it is necessary to use the solution both at client and server at side.

On the client side it is crucial to:

- Link the application's main program with provided ClientAPI library
- Add calls to Client API methods (for license control)

On the server side it is necessary to:

- Install *Node.js* 7.9.0
- Install required *Node.js* modules

[34] (msdn)

[35] (Kitware)

- Install Web Server from this project
- Create new license database at server
- Fill the database with application features, company info
- Fill the database with license instances
- Start server and put in Cloud

For the purpose of database creation and management a set of functions has been implemented in *Node.js* code.

2.7 Usage of Client API

To properly use the library it is necessary to include “LicClientApi.h” in the C++ code of the main application. Before using the library, the API should be initialized and uninitialized before the program exits.

Initialization:

```
#include “LicClientApi.h”

int main( int argc, const char * argv[] )
{
	// init API
	LicClientApi::ApiInit();

	// Now Api calls can be made

	// Uninit API
	LicClientApi::ApiUninit();

}
```

Just few lines of code should be added to activate licensing. This code looks like this:

```
#include “LicClientApi.h”
………
{
	// create API instance
	LicClientApi api;

	// start product
	if(    api.ProductStart(    "MyCompany",    "MyProduct    1.0"    )    ==
	LicClientApi::ERROR_OK )
	{
		// program started
		// do something, ask for license features

		// stop product
		api.ProductStop("MyCompany", "MyProduct 1.0");
	}
}
```

In the above code, “MyCompany” is the name of software provider (as defined in license database) and “MyProduct 1.0” is the name of application (also defined in license database).

2.8 Hardware fingerprint

In modern hardware technology, choosing unique hardware information becomes more and more difficult. Usages of virtual machines are constantly increasing and hardware information for virtual machines is mostly software emulated. Therefore in this work it is chosen to collect a set of hardware information, instead of just relying on one or two references. The uniqueness of hardware is based on a set of information about the hardware. The hardware fingerprint implemented in this project consists of DISK-ID, Volume-ID, CPU-ID, MAC-ID, WIN-ID and the Host Name.

Disk-ID:

Each disk drive is manufactured with a hard disk serial number of the main hard drive of a host computer. On a windows machine it can be found using a command prompt and typing in “wmin diskdrive get serialnumber”. A DISK-ID can look like this *“Serial:3JV4T8VE”.*

Volume-ID:

The Volume-ID represents a unique serial number that is assigned to an optical disk or formatted hard drive and is represented in a hexadecimal form. This number gives the exact date and time of when the formatting operating. On a windows machine one can look this number up, using the word “vol” when located in the C drive. It usually has this format *“C806E27A”*

CPU-ID:

This ID allows a processor to be uniquely distinguishable and it contains information about the producer of and some additional features. A CPU-ID looks something like this “*GenuineIntel-BFEB-FBFF-0002-06A7-0000-0000-0000-0000*”

MAC-ID:

The MAC address stays for media access control address and is a unique identifier on the data link layer and consists of 48-bit address space. It is a unique hardware identifier of the host computer in a communication network. Typically it has this format *“00:21:6A:12:28:A3“* where the order of the characters is not always the same.

***Win-ID*:**

By this name this paper references to the Windows Product ID which looks look this *„00371-OEM-9309421-26869“*

Host name:

This is the name of the host computer used for the license instance used as an identifier for the network. An example of a name would be *“DELLSERVER.MyCompany.local”* but it really depends on the name convention of the host user.

Using the above mentioned hardware information as base for hash fingerprint provides relative high uniqueness.

2.9 Use cases

Let’s look into a few actual use cases which can occur from the user’s side of view. The scenarios will be explained in further detail. As the implemented system works with registered host, rather than registered users the use cases will be distinguished between hosts. Obviously, hosts are run by persons that are referred to as end- users. Firstly the use cases for unregistered hosts will be looked into, which is followed by the registered use case scenarios and all additional error use cases.

2.9.1 Unregistered hosts

Hosts that are new to the software and have not registered before are known as unregistered host. As the LMS has not yet communicated with these hosts, a secure way of installing the licenses has to be given. The following scenarios describe what the use case scenarios are in this case.

a.) Unregistered host, no activation key

The first time a user wants to start the program on a not registered host, the licensing system asks for an activation key. Without this key no further actions can be made. If the user does not have an activation key, he cannot use the protected software.

What happens behind the visible process is that before asking for an activation key the licensing system already checks for existing licenses by looking in the windows registry database. In this case, as the host is not registered nor has the user typed in any activation key, he cannot use the software.

If a license instance was bought, but the key was not delivered by email or any other media, the user is recommended to contact the company directly to clear the issue.

If no license for the protected software was purchased, users wanting access to the software should inform their selves about existing products and fees. In some cases, owners of the software might provide trial versions for highly interested users. Of course these guidelines are made by the software company who owns the product.

b.) Unregistered host, valid activation key, license free

In this scenario the user that wants to work with the protected software, tries to access it by opening the program. Again the licensing system checks for already existing licenses, which is not the case in this use case scenario.

The first time a user starts the program, the host is not yet registered. The starting process fails due to missing license and registration. Afterwards the registration process is automatically called. Again this process is not graphically visible to the user, as it is done secretly working with encrypted data and the implemented remote web service. What happens behind the visible actions, is that the fingerprint of the host hardware is made and used for certification at local and server side. Besides that, it is checked remotely for possible license conflicts. After the successful registration of a host, the product can be started and used.

c.) Not registered, valid activation key, license used, right password

In the previous scenario the licensing instance was not used before and therefore called “free”. On the other hand, this case covers the usage of a license which is already assigned to another host. Therefore it is referred to as a “used” license. In order to continue with a usage of the software, the license needs to me “moved” from the other host to the current host. Moving the license means that the license information saved on the remote server, is updated with the hardware based fingerprint of the current unregistered host. In this particular use case, it is assumed that the user knows the correct password to continue the usage.

d.) Not registered, valid activation key, license used, wrong password

This use case differs to the one above only by the fact that the user does not type in the correct password. Again this can happen due to honest mistakes like wrong typing or attacks where an unauthorized user wants to start the program by using license instances on another host.

2.9.2 Registered hosts

As one might tell, registered hosts are the ones that used the software before and try to reuse it. All host information is already stored by the LMS and checked for correct further usages. The following use cases are describing the main possible use case scenarios, once a user starts the program on a registered host.

a.) Registered host, license correctly assigned

In this case the user sits on the host computer where he had installed the license instance, and tries to start the product. Using the hardware- locked node principal the licensing system extracts information about the hardware details of the host computer and decrypt local stored license. This information is finally compared with the stored fingerprint on the server side. In this case the user is run on the same host which is assigned to the license - there is no problem and the program can be started and used.

b.) Registered host, license incorrectly assigned, correct password

Once a user registers a host, he should be able to use the product on the computer which installed the license. In this particular case the host was correctly registered, but in the meantime the license is “moved” to another host. The system figures out that a wrong host is used and starts another process called the resolving one. Nevertheless, additional checks must be made in order to decide whether the product should be run or not.

The user is asked to type in the activation key once again, followed by the question for the resolving conflict password which is set beforehand by the user. Activation key and password are checked on the server side, which is again invisible for the user.

Given that the user types in the right password, the license is “moved” to calling host, and he is able to start the product without further notices.

c.) Registered host, license incorrectly assigned, wrong password

A user is trying to run a license instance on another host then the registered one, but user doesn’t have the correct password.

As in this scenario the user does not type in the right password, the program cannot be started and stops.

2.9.3 Other problems

As there is several error cases needed to be handled by the licensing system which are discussed in chapter 1.5.7, some additional use cases could be established. These concern the internet connection as well as the correctness of the used licensing instance.

a.) No internet connection

In this use case scenario, the end- user that wants to run the software has no internet connection. This could be due to issues on the local computer or the internet connection used by this user. As the LMS depends on a functioning internet connection, that user will not be able to start the software. If the client host computer sends a request to the server and does not get any response message, it is assumed that there is not internet connection.

b.) License expired

All licenses have a stored expiration date, after which it not allowed to start and use the program. This expiration date is stored at server side, usually connected with the purchase process. Therefore, when a user with an expired program license tries to start the program, the LMS will automatically check expiration date. If the license has expired, further usage of the software will not be allowed.

c.) Licenses corrupted

It can happen that the license on the used host cannot be decrypted using host fingerprint. This means that the current host fingerprint is different compared to the fingerprint made at the time of the license registration.

Case 1:

In this case, a regular user tries to start the software with a computer which out of some reason, does not have the same host hardware information like the registered one. This case can occur if there is a hardware change on the host. For example: disk has been replaced and new disk has a different disk ID, which changes the fingerprint.

Case 2:

Another case could be that an attacker copies license information from a registered host to an unregistered one. As another host has different hardware components, its fingerprint is different and the license cannot be decrypted.

d.) License not found

In this scenario, a user tries to start a program with an activation key that cannot be found. This can occur due to typing mistakes or randomly guessed license keys from an attacker. Obviously, in this case the system must not be started and any further access should be denied. If legally purchased license keys are not working and found, one should contact the company which uses the protecting licensing system.

e.) License not acceptable

There is another case when the license cannot be accepted as the local copy of it is outdated. This can happen when the company organization modified its license features and local copies do not contain the modifications. A typical case would be: one or more features of the license are purchased at the later stage, which is reflected in remote server database. The local copy of the license is still in “old” state, therefore this should be resolved.

3 Implementation/Solution

This chapter provides information about how the theory got developed in practice. The licensing solution and decision process are described. Besides that, this chapter is looking into the implementation of the project but also keeping an overview on the conceptual, general level.

This is the most detailed part of the thesis, containing details about the implemented system and of the solution to the given requirements.

3.1 Server Side Database

This sub chapter looks into the design of the SQL tables. They will be discussed and analyzed and the choices of the database system will be explained.

3.1.1 Implemented tables

In order to store and manage data used for the software licensing system, all information are stored in a relational SQL database. Therefore it was needed to create several tables for users, licenses, products and hosts. The following text explains the structure of the SQL tables and their relationship.[36]

Program and Program's features:

It is necessary to differ between the features and functionalities a user can or cannot use. These are usually imposed by the company role of a user, restricting and controlling the specific usage of the software. Those features were needed to add in a table CALLED PROGFEATURES which is the child table of the actual program table PROG. Each product which is licensed has his name, properties and list of individual features which can be activated and configured in the license instance.

Information about product features is represented in the tables *PROG* and *PROGFEATURES* where table PROG is the parent table, whereas PROGFEATURE is a dependent table referencing PROG table using foreign key PROG_ID.

```
CREATE TABLE PROG (
	PROG_ID      INTEGER          PRIMARY KEY,
	PROG_NAME    VARCHAR(255)     UNIQUE,
	PROG_TYPE    VARCHAR(255),
	PROG_DESC    VARCHAR(255)
);
```

[36] (w3schools, 2017)

The program ID in this case is used as a PRIMARY KEY constraint and therefore identifies this record uniquely. This primary key cannot contain a NULL value and has to be unique.

The other constraint which is used in the table above but also in the following ones is the UNIQUE constraint. It ensures that all values in a column are different to each other (PROG_NAME).

Both this constraints provide uniqueness for a column or a set of columns. One can have many UNIQUE constraints in table, but only one PRIMARY KEY.

```
CREATE TABLE PROGFEATURE (
      PROGFEATURE_ID       INTEGER PRIMARY KEY,
      PROG_ID              INTEGER,
      PROGFEATURE_NAME     VARCHAR(255),
      PROGFEATURE_ACTIVE   VARCHAR(255),
      PROGFEATURE_VALUE    INTEGER,
      PROGFEATURE_DESC     VARCHAR(255),
      UNIQUE(PROG_ID,PROGFEATURE_NAME),
      FOREIGN KEY(PROG_ID) REFERENCES PROG(PROG_ID)
);
```

The PROGFEATURE table uses the PROG_ID from the referenced table PROG as a FOREIGN KEY. These keys are used to link tables together where the FOREIGN KEY points to the PRIMARY KEY of the referenced table.

User and Company information:

As a user of a software it is very likely, that you work for a company which uses multiple instances of the licensed software. Therefore tables USER and COMPANY are interacting with each other.

```
CREATE TABLE COMPANY (
      COMPANY_ID      INTEGER         PRIMARY KEY,
      COMPANY_NAME    VARCHAR(255)    UNIQUE,
      COMPANY_ADDRESS VARCHAR(255),
      COMPANY_ZIP     VARCHAR(255),
      COMPANY_CITY    VARCHAR(255),
      COMPANY_STATE   VARCHAR(255),
      COMPANY_COUNTRY VARCHAR(255),
      COMPANY_SALT    VARCHAR(255),
      COMPANY_DESC    VARCHAR(255)
);
```

The COMPANY table contains a PRIMARY KEY (COMPANY_ID) and a UNIQUE constraint containing the name of the company (COMPANY_NAME). On the other hand the USER table is connected to that table integrating a FOREIGN KEY with its reference.

```
CREATE TABLE USER (
      USER_ID       INTEGER         PRIMARY KEY,
      COMPANY_ID    INTEGER,
      USER_NAME     VARCHAR(255),
      USER_DESC     VARCHAR(255),
      UNIQUE(COMPANY_ID,USER_NAME),
      FOREIGN KEY(COMPANY_ID)  REFERENCES COMPANY(COMPANY_ID)
);
```

An additional, unique constraint is applied to a pair of columns (COMPANY_ID,USER_NAME) as a composite key.

Host and Host information:

Each host computer is identified by his ID, his name and his fingerprint where the latest contains hardware information. As it is common that hardware can be partially changed during the host’s life (e.g disk changed after failure), this is a one-to-many relationship. Therefore two tables were created, where one is the host itself with his ID and name and the other one contains all the needed values for the fingerprint.

```
CREATE TABLE HOST (

      HOST_ID       INTEGER          PRIMARY KEY,
      HOST_NAME     VARCHAR(255)
);
```

Again the HOST table is functioning as a parent table to the HOSTINFO table using PRIMARY and UNIQUE constraints and referencing the HOST table in HOSTINFO by using a FOREIGN KEY.

```
CREATE TABLE HOSTINFO (

      HOSTINFO_ID        INTEGER          PRIMARY KEY,
      HOSTINFO_KEY       VARCHAR(255)      UNIQUE,
      HOSTINFO_NAME      VARCHAR(255),
      HOSTINFO_CPUID     VARCHAR(255),
      HOSTINFO_DISKID    VARCHAR(255),
      HOSTINFO_VOLID     VARCHAR(255),
      HOSTINFO_MACID     VARCHAR(255),
      HOSTINFO_WINID     VARCHAR(255),
      HOSTINFO_WINDID    VARCHAR(255),
      HOSTINFO_ACTTIME   VARCHAR(255),
      HOST_ID            INTEGER,
      FOREIGN KEY(HOST_ID)      REFERENCES HOST(HOST_ID)
);
```

License usage and features:

The implemented database does not only contain information about the license itself but also its license features which are generally defined in the table PROGRAMFEATURE and taken over. Three tables LIC, LICFEATURE and LICUSAGE are used to describe all that information. The license table LIC needs to be connected to almost all the other tables in the database. Again the LIC table is used as a parent table for two child tables called LICFEATURE and LICUSAGE. License usages are tracked to provide the software provider with log- like data showing when and how the license instances were used. This is needed because even when users are authenticated it is necessary to detect possible frauds by reporting usages to the customers of this software. With this information it can be checked for unused licenses which can avoid additional costs.

The column LIC_KEY contains the activation key for the license instance. This information should be given to the end user in order to properly license the software on his host. LIC_DUEDATE is a column which stores the expiring date of the license. Each license is protected with a password (LIC_PWD). This password will be asked by the ClientAPI in the resolve case that license is already installed at one host (HOST_ID) and end user would like to move it to another host.

```
CREATE TABLE LIC (

      LIC_ID             INTEGER   PRIMARY KEY,
      PROG_ID            INTEGER,
      COMPANY_ID         INTEGER,
      HOST_ID            INTEGER,
      LIC_NUM            INTEGER,
      LIC_KEY            VARCHAR(255),
      LIC_PWD            VARCHAR(255),
      LIC_TYPE           VARCHAR(255),
      LIC_DUEDATE        VARCHAR(255),
      LIC_ACTTIME        VARCHAR(255),
      UNIQUE(COMPANY_ID,PROG_ID,LIC_NUM),
      UNIQUE(PROG_ID,LIC_KEY),
      FOREIGN KEY(PROG_ID)     REFERENCES PROG(PROG_ID),
      FOREIGN KEY(COMPANY_ID)  REFERENCES COMPANY(COMPANY_ID),
      FOREIGN KEY(HOST_ID)     REFERENCES HOST(HOST_ID)
);
```

Three FOREIGN KEYS are inserted in the LIC table, which connect the table to PROG, COMPANY and HOST discussed before. As the tables COMPANY and HOST are parent tables to USER and USERINFO, the LIC table is indirectly also connected to those.

```
CREATE TABLE LICFEATURE (

      LICFEATURE_ID      INTEGER   PRIMARY KEY,
      LIC_ID             INTEGER,
      LICFEATURE_NAME    VARCHAR(255),
      LICFEATURE_ACTIVE  VARCHAR(255),
      LICFEATURE_VALUE   INTEGER,
      LICFEATURE_DESC    VARCHAR(255),
      UNIQUE(LIC_ID,LICFEATURE_NAME),
      FOREIGN KEY(LIC_ID) REFERENCES LIC(LIC_ID)
);
```

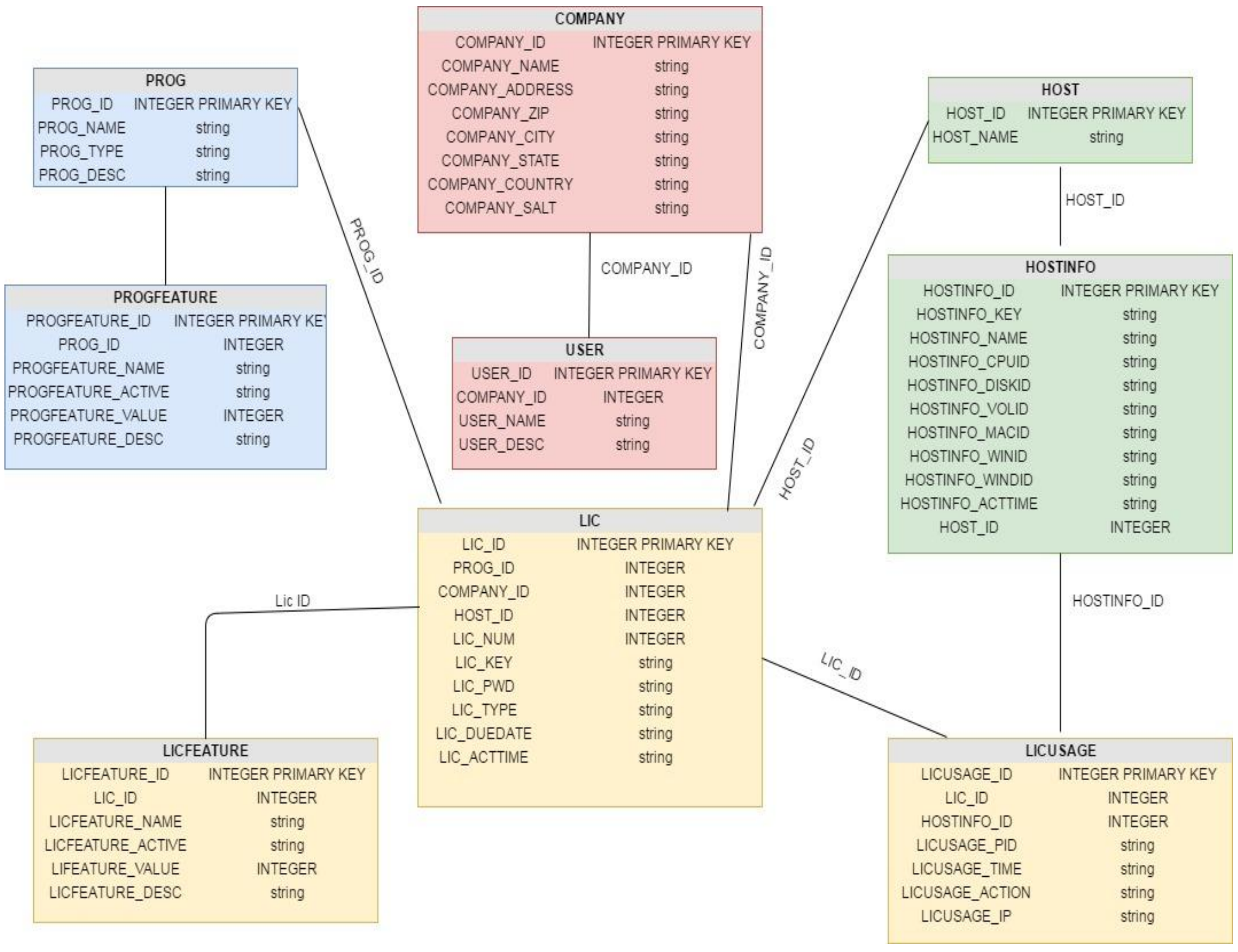
The LICFEATURE contains a UNIQUE constraint based on two column entities LIC_ID and LICFEATURE_NAME where the first one is taken out of the referenced table LIC. To complete the uniqueness of the table a PRIMARY KEY is added.

```
CREATE TABLE LICUSAGE (

      LICUSAGE_ID     INTEGER       PRIMARY KEY,
      LIC_ID          INTEGER,
      HOSTINFO_ID     INTEGER,
      LICUSAGE_PID    VARCHAR(255),
      LICUSAGE_TIME   VARCHAR(255),
      LICUSAGE_ACTION VARCHAR(255),
      LICUSAGE_IP     VARCHAR(255),
      UNIQUE(LIC_ID,HOSTINFO_ID,LICUSAGE_TIME),
      FOREIGN KEY(LIC_ID)       REFERENCES LIC(LIC_ID)
      FOREIGN KEY(HOSTINFO_ID)  REFERENCES HOSTINFO(HOSTINFO_ID)
);
```

The LICUSAGE table contains two FOREIGN KEYS connecting it to the table LIC and HOSTINFO and a UNIQUE constraint with three components LIC_ID, HOSTINFO_ID and LICUSAGE_TIME.

Figure 3-1 gives an overview of all implemented tables in the SQL database and their relations to each other.



PROG	
PROG_ID	INTEGER PRIMARY KEY
PROG_NAME	string
PROG_TYPE	string
PROG_DESC	string

PROGFEATURE	
PROGFEATURE_ID	INTEGER PRIMARY KE
PROG_ID	INTEGER
PROGFEATURE_NAME	string
PROGFEATURE_ACTIVE	string
PROGFEATURE_VALUE	INTEGER
PROGFEATURE_DESC	string

COMPANY	
COMPANY_ID	INTEGER PRIMARY KEY
COMPANY_NAME	string
COMPANY_ADDRESS	string
COMPANY_ZIP	string
COMPANY_CITY	string
COMPANY_STATE	string
COMPANY_COUNTRY	string
COMPANY_SALT	string

USER	
USER_ID	INTEGER PRIMARY KEY
COMPANY_ID	INTEGER
USER_NAME	string
USER_DESC	string

HOST	
HOST_ID	INTEGER PRIMARY KEY
HOST_NAME	string

HOSTINFO	
HOSTINFO_ID	INTEGER PRIMARY KEY
HOSTINFO_KEY	string
HOSTINFO_NAME	string
HOSTINFO_CPUID	string
HOSTINFO_DISKID	string
HOSTINFO_VOLID	string
HOSTINFO_MACID	string
HOSTINFO_WINID	string
HOSTINFO_WINDID	string
HOSTINFO_ACTTIME	string
HOST_ID	INTEGER

LIC	
LIC_ID	INTEGER PRIMARY KEY
PROG_ID	INTEGER
COMPANY_ID	INTEGER
HOST_ID	INTEGER
LIC_NUM	INTEGER
LIC_KEY	string
LIC_PWD	string
LIC_TYPE	string
LIC_DUEDATE	string
LIC_ACTTIME	string

LICFEATURE	
LICFEATURE_ID	INTEGER PRIMARY KEY
LIC_ID	INTEGER
LICFEATURE_NAME	string
LICFEATURE_ACTIVE	string
LIFEATURE_VALUE	INTEGER
LICFEATURE_DESC	string

LICUSAGE	
LICUSAGE_ID	INTEGER PRIMARY KEY
LIC_ID	INTEGER
HOSTINFO_ID	INTEGER
LICUSAGE_PID	string
LICUSAGE_TIME	string
LICUSAGE_ACTION	string
LICUSAGE_IP	string

Figure 3-1: Overview of implemented SQL database with dependencies

(©Gloria Janjic, 2017)

3.1.2 SQLITE queries

Along all the existing SQLITE queries, the implementation’s most important ones used were INSERT, SELECT and UPDATE as for all the tables above it needs to be possible to add, find and update items. Most of these tasks were implemented separately for each of the tables in the database.

Add functions:

Add functions are implemented as follows using methods from the async *Node.js* module (see Chapter 3.3.1) discussed later on. The method does not get any parameters and adds a new entry to the table HOST with a unique integer PRIMARY KEY host_id.

```
module.exports.Host.Add = async function () {

    return new Promise( function(resolve, reject) {

        global_db.serialize(function() {

            // INSERT new host
            var stmt = global_db.prepare("INSERT INTO host(host_id)
VALUES(null)");
            stmt.run( function( err ){
                if( err )
                {
                    throw( err );
                    var host_id = 0;
                    return host_id;
                }

                var host_id = this.lastID;
                resolve( host_id );
            });
            stmt.finalize();
        })

    });
};
```

In order to make a method as a publicly available function, the feature *module.exports* is used. To ensure that SQL lines are executed in order, the method *serialize* from the global database class was used. Afterwards the SQLITE query is prepared and run, throwing an exception if one occurs. In the end sqlite3 needs to have a method *finalize* called, which deletes the prepared statement in order to avoid resource leaks.

Find functions:

The structure of the find method is similar to the add query function besides the fact that, it gets a parameter which in this case is *name*.

```
module.exports.Prog.Find = function( name ) {

    return new Promise( function(resolve, reject) {

        var sql = "SELECT prog_id FROM prog WHERE prog_name = ?";
        global_db.get( sql, name, function(error, row) {

            // prepare return value
            var prog_id = 0;
            if( row ) {
                prog_id = row.PROG_ID;
            }

            // return value using promise
            resolve( prog_id );
        });
    });
};
```

This particular function finds a program by looking for a program name which equals to the *name* parameter in the table PROG. The result of this method is a return value which is the wished *prog_id*.

Update functions:

The update function is shown by another example which is used for the table HOSTINFO. As discussed before, this table stores all hardware information about a computer which is registered with a license. In order to update the value of the CPU ID with a number, two parameters are delivered in the method.

```
module.exports.HostInfo.CpuID = async function ( hostinfo_id, cpuid ) {

    return new Promise( function(resolve, reject) {

        global_db.serialize(function() {

            //UPDATE property
            var stmt = global_db.prepare("UPDATE hostinfo SET  hostinfo_cpuid = ? 
WHERE hostinfo_id = ?");
            stmt.run( cpuid, hostinfo_id, function( err ){
                if( err )
                {
                    throw( err );
                }
                else if( this.changes != 1 )
                {
                    throw( "Changes != 1")
                }

                resolve();
            });
            stmt.finalize();
        })

    });
};
```

The query of the above code updates the entry of the table HOSTINFO with the HOSTINFO_ID with a CPU_ID and stores that value in this column.

3.1.3 Database Creation

The set of *JavaScript* methods has been developed for programmatically updating of server side database. All methods are exported in module “LicDb.js”. With the following *JavaScript* code lines, a new license database can be created:

```
// load LicDb module
var licdb      = require('./LicDb.js');

// create new license database
await licdb.Database.Create( "mydatabase.db" );

// do something with database …

// close database
await licdb.Database.Close();
```

When opening the existing database the function *licdb.Database.Open()* is called, which opens that database in the correct way. This method should be used whenever the database needs to be opened for further work. Therefore, creating additional methods to do that is not needed. The function *Create()* will create a new database file and initialize the database schema. Compared to *Create()* the function *Open()* only opens the existing database; the table schema is already inserted and not created in the method.

The next code describes how a new company can be added along with its name, address, zip code, city, state, country location and a special salt. The company is inserted in the following way and the properties set:

```
//////////////////////////////
// Add company
//////////////////////////////

var company_id = await licdb.Company.Add( "My Company" );
await licdb.Company.Address( company_id, "Jakominiplatz 12" );
await licdb.Company.Zip( company_id, "8045" );
await licdb.Company.City( company_id, "Graz" );
await licdb.Company.State( company_id, "Styria" );
await licdb.Company.Country( company_id, "Austria" );
await licdb.Company.Salt( company_id, "My Company Salt" );
await licdb.Company.Desc( company_id, "My Company Description" );
```

Once the company is inserted in the database, users from that company could be added. An example code of how to add users to the company looks like this:

```
//////////////////////////////
// Add few company users
//////////////////////////////

var user_id;
user_id = await licdb.User.Add( company_id, "user1@mycompany.at" );
          await licdb.User.Desc( user_id, "This is description for user1" );
user_id = await licdb.User.Add( company_id, "user2@mycompany.at" );
          await licdb.User.Desc( user_id, "This is description for user2" );
user_id = await licdb.User.Add( company_id, "user3@mycompany.at" );
          await licdb.User.Desc( user_id, "This is description for user3" );
```

As discussed before, companies can of course produce and protect more than one software product. If these software products need to be used and distinguished, they have to be added into the table. To insert one of those, the following sample code needs to be executed. It adds the product along with its name and a decryption.

```
//////////////////////////////
// Add company program
//////////////////////////////

var prog_id;
prog _id = await licdb.Prog.Add( company_id, "My Software Program1" );
           await licdb.Prog.Desc( product_id, "Description of My Software Prog1"
);
```

Each software product can have one or more product features. Each individual feature can be active or disabled and contains a configuration value and description. Using the ClientApi, the software product can read features and its properties. A few product features are inserted with following example code.

```
/////////////////////////////
// Add program features
//////////////////////////////

var feature_id;
feature_id = await licdb.FeatureAdd( prog_id, "My program, Feature1" );
             await licdb.FeatureActive( feature_id, true );
             await licdb.FeatureValue( feature_id, "On" );
             await licdb.FeatureDesc( feature_id, "Description for Feature1" );

feature_id = await licdb.FeatureAdd( prog_id, "My program Feature2" );
             await licdb.FeatureActive( feature_id, false );
             await licdb.FeatureValue( feature_id, "Off" );
             await licdb.FeatureDesc( feature_id, "Description for Feature2" );

feature_id = await licdb.FeatureAdd( prog_id, "My program Feature3" );
             await licdb.FeatureActive( feature_id, true );
             await licdb.FeatureValue( feature_id, "Limit=22" );
             await licdb.FeatureDesc( feature_id, "Description for Feature3" );
```

By using *FeatureAdd()*, a feature is added in the table. The function *FeatureActive()* sets the value on *false* or *true*, whereas the *FeatureActive()* method is set on "*ON*" or "*OFF*". The function *FeatureDesc()* inserts a description for the feature. In the example above, the active feature was added with the value "*On*", the *disabled* feature 2 set with the value "*Off*" and the *active* feature 3 with the value "*Limit=22*".

After having created company and product, it is a rather easily and straightforward implementation to generate several licenses. In the following example code, it is shown how 3 licenses could be added.

```
/////////////////////////////
// Add 3 licenses
//////////////////////////////

var lic_id;

lic_id = await licdb.Lic.Add( company_id, prog_id, 1 );
               licdb.Lic.Type( lic_id, "EVALUATION" );
               licdb.Lic.Pwd( lic_id, "secret password for license 1" );
               licdb.Lic.DueDate( lic_id, "31.12.2018" );
               licdb.Lic.Key( lic_id );

lic_id = await licdb.Lic.Add( company_id, prog_id, 2 );
               licdb.Lic.Type( lic_id, "STUDENT" );
               licdb.Lic.Pwd( lic_id, "secret password for license 2" );
               licdb.Lic.DueDate( lic_id, "31.12.2018" );
               licdb.Lic.Key( lic_id );

lic_id = await licdb.Lic.Add( company_id, prog_id, 3 );
               licdb.Lic.Type( lic_id, "COMMERCIAL" );
               licdb.Lic.Pwd( lic_id, "secret password for license 3" );
               licdb.Lic.DueDate( lic_id, "31.12.2018" );
               licdb.Lic.Key( lic_id );
```

3.2 Client Access API Library

This part is discussing the interface functions, HTTP library and cryptopp. Besides that the implementation in C/C++ is explained. Like in the description of the Web Service which will follow in chapter 3.4, these four methods are looked into separately: product register, product start, product conflict, product stop.

```
class LicClientApi
{
public:
      enum ErrorCode
      {
            ERROR_OK,
            ERROR_USER_CANCEL,
            ERROR_NO_INTERNET,
            ERROR_LICENSE_CORRUPTED,
            ERROR_LICENSE_EXPIRED,
            ERROR_LICENSE_CONFLICT,
            ERROR_LICENSE_NOTFOUND
      };
public:

      static void   ApiInit();
      static void   ApiUninit();

      ErrorCode    ProductStart(    const    std::string&    company_id,    const
std::string& application_id );

      ErrorCode    ProductStop( const std::string& application_id );

      ErrorCode    ProductRegister(    const    std::string&    company_id,    const
std::string& application_id );

      ErrorCode    ProductResolveConflict(   const   std::string&   application_id,
const std::string& activation_key );
};
```

3.2.1 Client side - Product Register

The first step in using the software is the registration of the license. This step normally occurs after the installation of the software at the user host and during the first program start. In this case the user should already own a software activation key, received either per email or any other media. The activation key looks like a standard SHA1 hash, formatted inside brackets in five groups of eight hex digits:

```
{B65CD32E-0A83E943-2C791952-CA06ACDC-19A5989E}
```

Each license instance has its own activation key, stored in the server sided database. It is however allowed that the same activation key can be used for more than one "Product". For example, if a Software Company has two products (let's say Product1 and Product2) and the end customer buys one license of each product, it is possible to configure the licensing system to use the same activation key for both products. This significantly simplifies user and license management, while it is possible to configure complete host installations with one activation key.

At the server side the Software Provider can, if desired, truly limit the number of software activations either for the same license or for the same activation key. Also, activation keys can be "marked as invalid" for one license and replaced with a new one.

Step by step workflow:

In order to explain the tasks which are done in this method, the following workflow shows a step by step of all implemented methods.

1. [Client] ***Call Client API***
   a. The application is calling the Client API method "Product Register" with the "CompanyName" and "ProductName"

2. [ClientAPI] ***Get activation key***

   a. Read the product activation key from register database
   b. Set read activation key as default
   c. Ask user for the activation key
   d. Return ERROR_USER_CANCEL if user cancels

3. [ClientAPI] ***Prepare AES-CFB key***
   a. Generate a random AES key
   b. Get RSA public key from server
   c. Encrypt AES key using RSA and server public key

4. [ClientAPI] ***Prepare fingerprint:***
   a. Get hardware info
      i. Read Disk-ID
      ii. Read Vol-ID
      iii. Read Cpu-ID
      iv. Read MAC-ID
      v. Read Win-ID
      vi. Read host name
      vii. Read host time
   b. Store hardware information as a string in XML format

5. [ClientAPI] ***Prepare HTTP request:***
   a. Insert header line "*App-Cipher*" containing RSA encrypted AES key
   b. Insert header line *"App-Group"* containing plaintext company id
   c. Insert header line "App-Key" containing AES encrypted activation key
   d. Insert header line "App-Lic" containing empty string
   e. Insert header line "App-Name" containing plaintext application name
   f. Insert header line "App-Pwd" containing empty string
   g. Insert header line "App-Pid" containing AES encrypted process ID
   h. Insert header line "Host-Info" containing AES encrypted XML hardware info
   i. Set URL to `"https://company.name.at/License/Register"`

6. [ClientAPI] ***Contact license server***
   a. Send HTTP request to server and get response
   b. If fails return ERROR_NO_ INTERNET
   c. If HTTP status code 404 return ERROR_LICENSE_NOTFOUND
   d. If HTTP status code 409 return ERROR_LICENSE_CONFLICT
   e. If HTTP status code 503 return ERROR_NO_INTERNET
   f. If HTTP status code is not 200 return ERROR_LICENSE_NOTFOUND

7. [ClientAPI] ***Extract license***
   a. Extract content of response header "App-Lic"
   b. Decrypt "App-Lic" using AES key into license memory XML file

   c. If license is not a valid XML file return ERROR_LICENSE_CORRUPTED

8. [ClientAPI] ***Check license correctness***
   a. If activation key from XML file does not match requested one return ERROR_LICENSE_CORRUPTED
   b. Compare due date from the license file and return ERROR_LICENSE_EXPIRED if license is expired

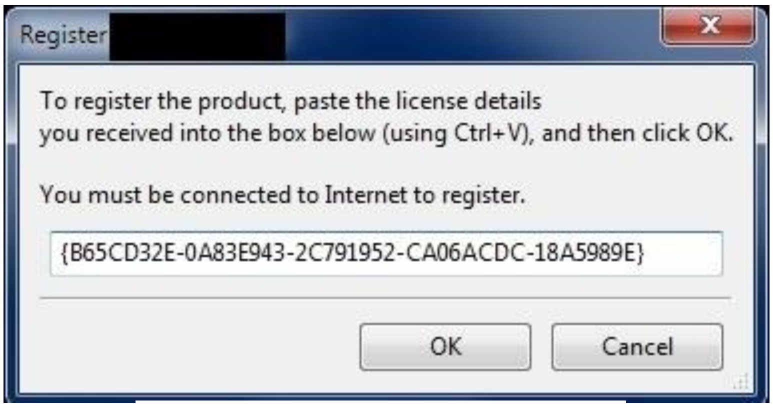
9. [ClientAPI] ***Update register database***
   a. Encrypt license XML file using AES and hardware fingerprint as a key
   b. Create folder in register database called “CompanyName”
   c. Create child folder “ApplicationName”
   d. Create child folder “Install”
   e. Create child item ”App-Key” with activation key as a content
   f. Create child item “App-Lic” with encrypted license as a content

10. [ClientAPI] ***Return ERROR_OK***

Implementation details of step by step workflow: [37]

So, in this scenario, the user has started the Software Product for the very first time. As a result, the Software Product is calling the ClientApi method “Product Start” which checks Windows Registry (wxWidgets) and detects that there is no installed license for the product. Subsequently the method “Product Register” is automatically called **(Step 1).**

In **Step 2**, a user is asked to enter the activation key, using GUI from wxWidgets library. Figure 3-2 shows the dialogue where the user is supposed to enter the activation key.


Register [redacted]

To register the product, paste the license details you received into the box below (using Ctrl+V), and then click OK.

You must be connected to Internet to register.

{B65CD32E-0A83E943-2C791952-CA06ACDC-18A5989E}

Figure 3-2: Ask for activation key

(©Gloria Janjic, 2017)

[37] (curl, 2010)

After entering the activation key, **Step 3** generates a random AES-CFB key and encrypts it using RSA server public key, which is done using the Crypto++ library.

Hardware information is collected in **Step 4**, using *standard C++ libraries* and the *wxWidgets* library and stored in XML format. As an example of how extracting hardware looks on the client side, the following code shows how the volume ID of a host can be extracted:

```
std::string  LicHost::getVolID( int drive )
{
	std::string info;

	// volume name
	char ch = 'C' + drive;
	char name[16];
	sprintf( name, "%c:\\", ch );

	// get volume info
	DWORD serialNum;
	if( GetVolumeInformationA( name, NULL, 0, &serialNum, NULL, NULL, NULL, 0 )
)
	{
		char buffer[256];
		sprintf( buffer, "%0X", serialNum );

		info = buffer;
	}

	return info;
}
```

In **Step 5** the HTTP request is prepared, which is done using the *curl library*. How to do that is shown in the code snippet below.

The method *curl_easy_init()* is the first function to be called to get back a CURL easy handle. Afterwards this handle needs to be used as an input in order to call other functions from that interface.

```
// prepare HTTP request
CURL *curl = curl_easy_init();
if( curl == NULL )
	return ERROR_NO_INTERNET;

curl_easy_setopt(curl, CURLOPT_URL, URL_REGISTER);
curl_easy_setopt(curl, CURLOPT_TIMEOUT, 20L);

struct curl_slist *header = NULL;

std::string line = "Content-type: text/html; charset=utf-8";
header = curl_slist_append( header,  line.c_str() );

line = "App-Cipher: " + key_cipher_base64;
header = curl_slist_append( header, line.c_str() );
// printf( "key_cipher_base64: %s\n", key_cipher_base64.c_str() );

line = "App-Group: " + company_id;
header = curl_slist_append( header, line.c_str() );

line = "App-Name: " + application_id;
header = curl_slist_append( header, line.c_str() );
```

```
line = "App-Key: " + aes.EncryptBase64( app_key.ToStdString() );
header = curl_slist_append( header, line.c_str() );

std::string pwd = "pub1tdv";
line = "App-Pwd: " + aes.EncryptBase64( pwd );
header = curl_slist_append( header, line.c_str() );

std::string pid = wxString::Format( "%ld", wxGetProcessId() );
line = "App-PID: " + aes.EncryptBase64( pid );
header = curl_slist_append( header, line.c_str() );

std::string host_info = LicHost::MakeXml();
line = "Host-Info: " + aes.EncryptBase64( host_info );
header = curl_slist_append( header, line.c_str() );

curl_easy_setopt(curl, CURLOPT_HTTPHEADER, header );

/* open the header file */
std::vector<std::string> header_response;
curl_easy_setopt( curl, CURLOPT_HEADERFUNCTION, header_callback );
curl_easy_setopt( curl, CURLOPT_HEADERDATA, &header_response );
```

In order to tell the *libcurl library* how to behave, *curl_easy_setopt()* is called which is primarily used in the setup phase of the curl library. The setups made in this function are usually valid for all further transfers that make use of this handle.

The next method used and discussed is *curl_slist_append()* which appends a string to a list of strings. Therefore the linked list needs to be passed, as well as the new list returned from this function which is copied in this variable when the function returns.

In **Step 6** the request is send to the licensing server and onwards the response is checked for an error which distinguishes between having no internet connection, the license not being found or a license conflict occurring.

```
CURLcode res = curl_easy_perform( curl );


if( res == CURLE_COULDNT_CONNECT )
{
        curl_easy_cleanup(curl);
        curl_slist_free_all( header );
        return ERROR_NO_INTERNET;
}

if( res == CURLE_OPERATION_TIMEDOUT )
{
        curl_easy_cleanup(curl);
        curl_slist_free_all( header );
        return ERROR_NO_INTERNET;
}

if( CURLE_OK != res )
{
        curl_easy_cleanup(curl);
        curl_slist_free_all( header );
        return ERROR_NO_INTERNET;
}
```

```
// get HTTP code
long codep;
curl_easy_getinfo(curl, CURLINFO_RESPONSE_CODE, &codep);

// License not found
if( codep == 404 )
{
	curl_easy_cleanup(curl);
	curl_slist_free_all( header );

	return ERROR_LICENSE_NOTFOUND;
}

// License conflict
if( codep == 409 )
{
	curl_easy_cleanup(curl);
	curl_slist_free_all( header );

	return ERROR_LICENSE_CONFLICT;
}

// Service Unavailable
if( codep == 503 )
{
	curl_easy_cleanup(curl);
	curl_slist_free_all( header );

	return ERROR_NO_INTERNET;
}

// Licence not found if not status 200
if( codep != 200 )
{
	curl_easy_cleanup(curl);
	curl_slist_free_all( header );

	return ERROR_LICENSE_NOTFOUND;
}
```

The method *curl_easy_perform()* is used after all the *curl_easy_init()* and *curl_easy_setopt()* calls were made. It performs the actual transfer which is described in these options and must be called with the same handle.

In order to end a *libcurl easy handle* the *curl_easy_cleanup()* is called as the opposite of the *curl_easy_init()* function discussed before. Obviously both functions need to be called with the same handle. After calling this function all connections this handle has used, are closed.

To make sure all traces to previously linked lists to *curl_slist* are removed by calling curl_slist_free_all(). This function does not return any values.

In **Step 7, 8 and 9** the license is extracted from the HTTP response header and checked for validity. Therefore the license file is decrypted first and loaded into the memory before it is checked.

```
// decrypt license file
std::string app_lic_plain  = aes.DecryptBase64( app_lic_cipher );

// load license into memory
wxXmlDocument xml_lic;
wxStringInputStream stream( app_lic_plain );
xml_lic.Load( stream );

if( !xml_lic.IsOk() )
{
	curl_easy_cleanup(curl);
	curl_slist_free_all( header );

	return ERROR_LICENSE_CORRUPTED;
}

// get root
wxXmlNode* root = xml_lic.GetRoot();
if( root == NULL )
{
	curl_easy_cleanup(curl);
	curl_slist_free_all( header );

	return ERROR_LICENSE_CORRUPTED;
}

// check if license expired ?
if( RemainingDays( root ) <= 0 )
{
	curl_easy_cleanup(curl);
	curl_slist_free_all( header );

	return ERROR_LICENSE_EXPIRED;
}

// read application key
std::string lic_act_key = GetActKey( root );

// should be same
if( app_key.compare( lic_act_key ) )
{
	curl_easy_cleanup(curl);
	curl_slist_free_all( header );

	return ERROR_LICENSE_CORRUPTED;
}
```

Finally, in **Step 9** the encrypted license and activation key are stored in the nodes of Windows Registry Database (wxWidgets) as follows:

```
// update config
wxConfig config( application_id, company_id );
config.SetPath( L"Install" );

// encrypt license file using fingerprint
aes.SetKey( LicHost::MakeFingerprint() );
std::string app_lic = aes.EncryptBase64( app_lic_plain );
```

```
// update
config.Write( L"App-Key", wxString(app_key) );
config.Write( L"App-Lic", wxString(app_lic) );
```

3.2.2 Client side - Product Resolve

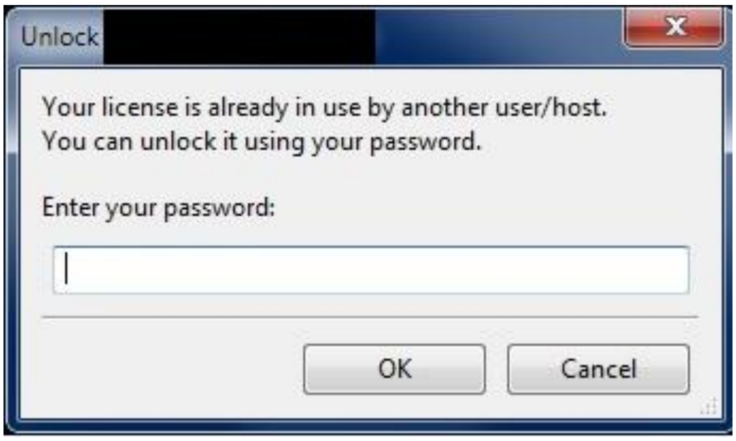
In some circumstances the user wants to install a product at an unregistered host, even though it is already installed and activated at a registered host. There is only a few cases, in which this should be allowed:

- The host on which the software is already installed and activated has a hardware failure (burned in fire or any other catastrophic case)
- The host hardware has been changed and treated as a new host, while the hardware fingerprint doesn't match.

For this purpose, the method "Product Resolve" is implemented. It should be called in the case where method "Product Register" returns the error code ERROR_LICENSE_CONFLICT. This error code is detected by the License Server when the license is already activated at another host (different of the host making registering request).

The basic idea of resolving the conflict is to ask the user for a secret password, in addition to the license activation key. Each license has a "resolving password" stored in the database at the server side. Therefore, in case of a license conflict, the server can "move" licenses from one host to another if corresponding password matches.

The implementation of "Product Resolve" method is mostly similar to the method "Product Register" in the Client API. The only difference is in **Step 2**, where the user is asked for resolving his password (in addition to activation key). This dialogue windows is shown in Figure 3-3.


Unlock [redacted]

Your license is already in use by another user/host.
You can unlock it using your password.

Enter your password:

Figure 3-3: Ask for resolve password

(©Gloria Janjic, 2017)

The following workflow summarizes all steps of the method “Product Resolve”:

1. [Client] ***Call Client API*** (same as in Product Register)

2. [ClientAPI] ***Get activation key and password***
   a. Read product activation key from register database
   b. Set read activation key as default
   c. Ask user for the activation key and password
   d. Return ERROR_USER_CANCEL if user cancels

3. [ClientAPI] ***Prepare AES-CFB key*** (same as in Product Register)

4. [ClientAPI] ***Prepare fingerprint*** (same as in Product Register)

5. [ClientAPI] ***Prepare HTTP request*** (same as in Product Register)

6. [ClientAPI] ***Contact license server*** (same as in Product Register)

7. [ClientAPI] ***Extract license*** (same as in Product Register)

8. [ClientAPI] ***Update register database*** (same as in Product Register)

9. [ClientAPI] ***Return ERROR_OK*** (same as in Product Register)

3.2.3 Client side - Product Start

Each time when application is started Client API method “Product Start” is called by the application in **Step 1**.

If the application is already registered – and this should be a normal case - license key and encrypted license are already stored in Windows Register Database. Both license key and encrypted license are loaded in **Step 2**.

In **Step 3** the local license is checked for validity; it is first decrypted using host fingerprint and secondly checked the license matches activation key.

Now the HTTP request is prepared in **Step 4** and server is contacted in **Step 5.** The encrypted loaded license is send to server in request header “App-Lic”. Server must check if the license is still valid and if the license content is not outdated.

If no problem is reported by server, license is checked for expiration in **Step 6**.

The following workflow summarizes all steps of the method “Product Start”:

1. [Client] ***Call Client API***
    a. The application is calling the Client API method product start with the company and product name

2. [ClientAPI] ***Get registered license and key***
    a. Read product activation key from register database
    b. Read encrypted license from register database
    c. Return ERROR_LICENSE_NOTFOUND if fails

3. [ClientAPI] ***Check local license***
    a. Get hardware info
        i. Read Disk-ID
        ii. Read Vol-ID
        iii. Read Cpu-ID
        iv. Read MAC-ID
        v. Read Win-ID
        vi. Read host name
        vii. Read host time
    b. Generate hardware fingerprint using hardware information
    c. Decrypt license using AES-CFB mode and hardware fingerprint as key
    d. If fails return ERROR_LICENSE_CORRUPTED
    e. Check if application key matches application key in license
    f. If fails return ERROR_LICENSE_CORRUPTED

4. [ClientAPI] ***Prepare HTTP request***
    a. Store hardware info as string in XML format
    b. Insert header line “*App-Cipher*” containing RSA encrypted AES key
    c. Insert header line “*App-Group”* containing plaintext company id
    d. Insert header line “App-Key” containing AES encrypted activation key
    e. Insert header line “App-Lic” containing AES encrypted license in XML format
    f. Insert header line “App-Name” containing plaintext application name
    g. Insert header line “App-Pwd” containing empty string
    h. Insert header line “App-Pid” containing AES encrypted process ID
    i. Insert header line “Host-Info” containing AES encrypted XML hardware info
    j. Set URL to `"https://company.name.at/License/Start"`

5. [ClientAPI] ***Contact license server***
    a. Send HTTP request to server and get response
    b. If fails return ERROR_NO_ INTERNET
    c. If HTTP status code 404 return ERROR_LICENSE_NOTFOUND
    d. If HTTP status code 406 return ERROR_LICENSE_NOTACCEPTABLE
    e. If HTTP status code 409 return ERROR_LICENSE_CONFLICT

   f. If HTTP status code 503 return ERROR_NO_INTERNET
   g. If HTTP status code is not 200 return ERROR_LICENSE_NOTFOUND

6. [ClientAPI] ***Check license correctness***
   a. If activation key from XML file does not match requested one return ERROR_LICENSE_CORRUPTED
   b. Compare due date from the license file and return ERROR_LICENSE_EXPIRED if license is expired

7. [ClientAPI] ***Return ERROR_OK***

3.2.4 Client side - Product Stop

The method “Product Stop” is called by the application right before stopping the program execution. The purpose of this call is to update information at the server side when the application is stopped, mainly to measure the duration of the software usage.

Compared to the other methods, this one is the only silent method. This means, that in the case of any error the user does not receive any message. The reason is obvious – as the user is stopping the software anyways, it is irrelevant if anything went wrong with the license, internet, contacting server etc. The method is identical to “Product Start”, but silent.

The following workflow summarizes all steps of method “Product Stop”:

1. [Client] ***Call Client API*** (same as in Product Start)

2. [ClientAPI] ***Get registered license and key*** (same as in Product Start)

3. [ClientAPI] ***Check local license*** (same as in Product Start)

4. [ClientAPI] ***Prepare HTTP request*** (same as in Product Start)
   a. Store hardware info as string in XML format
   b. Insert header line “*App-Cipher*” containing RSA encrypted AES key
   c. Insert header line *“App-Group”* containing plaintext company id
   d. Insert header line “App-Key” containing AES encrypted activation key
   e. Insert header line “App-Lic” containing AES encrypted license in XML format
   f. Insert header line “App-Name” containing plaintext application name
   g. Insert header line “App-Pwd” containing empty string
   h. Insert header line “App-Pid” containing AES encrypted process ID
   i. Insert header line “Host-Info” containing AES encrypted XML hardware info
   j. Set URL to "https://company.name.at/License/Stop"

5. [ClientAPI] ***Contact license server*** (same as in Product Start)

6. [ClientAPI] ***Check license correctness*** (same as in Product Start)

7. [ClientAPI] ***Return ERROR_OK***

3.3 Web Service

Firstly this chapter explains the solution implementation step by step divided into four main distinguished case methods. Afterwards the design of the implementation will be discussed along with some further details to the languages and tools used.

The implemented web service works as a standard HTTP web service installed at the local back server computer, running behind an Apache front server. The front server (Cloud) is expecting client requests as follows:

```
"https://company.name.at/License/Start"
"https://company.name.at/License/Stop"
"https://company.name.at/License/Register"
"https://company.name.at/License/Resolve"
```

These requests are forwarded to the local background server computer where the web service is running.

```
"http://localserver:15108/License/Start"
"http://localserver:15108/License/Stop"
"http://localserver:15108/License/Register"
"http://localserver:15108/License/Resolve"
```

Typical Apache configuration can be as follows:

```
<VirtualHost company.name.at:80>
        ServerAdmin webmaster@localhost

        ProxyPreserveHost On

        ServerAdmin webmaster@localhost

        ProxyPreserveHost On

        <Location /License>
                ProxyPass        http://localserver:15108/License DisableReuse=On

                ProxyPassReverse http://localserver:15108/License

                Order allow,deny
                Allow from all
        </Location>
< /VirtualHost >
```

After receiving a client request the web server reaction can be distinguished into 4 different methods which are: product start, product register, product conflict and product stop. These methods are going to be discussed in further detail in the following sub chapters.

3.3.1 Modules in Node.js

Nodej.js has a rather simple module loading system, which treats each file as a separate module. Depending on the *Node.js* version used, some of the following modules described are already in the standard library, ready to be used whereas others need to be installed additionally.

For installing additionally needed modules, it is recommended to do that with a command line instruction in the folder where you need the module. Using *npm* to do so the command looks like this:

```
➢  npm install <package_name>
```

Using functions of a module in data files one first has to export its constructor into a variable which looks like this:

```
var MyModule = require('MyModule');
```

async [38]

In order to understand why this module was needed, the difference between *blocking vs. non- blocking and synchronous vs. asynchronous coding* has to be examined shortly. Blocking operations occur, when executing a *JavaScript* process in *Node.js* has to wait for another non- *JavaScript* operation to be completed. Most blocking operations are used with the synchronous methods from the standard library but native modules also appear to use them. Generally speaking, non- blocking methods execute code asynchronously whereas blocking methods execute it synchronously.

JavaScript in general is using asynchronous execution, which is a problem when working with databases where you need most function calls to be synchronous. It is possible to use callback functions to make an implemented code execute asynchronously, but this easily creates code that is not readable and non- intuitive. Therefore for this project, an additional module called "async" was used in the implementation, making it easier to develop and read asynchronous code looking synchronous. This module allows the user to wait for asynchronous methods to finish before executing another task. In order to use that this implementation uses the keywords "*async*" and "*await*".

sqlite3 [39]

The module used to create and interact with a server side database is called "sylite3", which in this project was installed per command line using *npm*. It works with the following versions of *Node.js* v0.10.x, *v0.12.x*, *v4.x*, and *v5.x*. This module binds an asynchronous and non-blocking SQLite3 version for *Node.js*.

[38] (Node.js Foundation, 2017)

[39] (Orlando Vazquez)

SQLite is a database engine that does not need a server in order to run and is widely popular. Using this module, one can create SQLite databases and perform SQL operations on them

xmlbuilder

In this thesis an encrypted license is exchanged between client and server in an XML format. This module enables the server side program to convert a string into XML format and prepare it for sending it over the internet to the client.

This format includes a root node, several children nodes and additional elements created and filled out with necessary information about the license.

fs

The module's name "*fs*" stands for File System and in this project is used to handle the file containing the database. All methods have synchronous and asynchronous forms which allow reading and writing of the database file.

http [40]

This is a built-in module in *Node.js* and allows data to be exchanged and transferred over HTTP. To use a HTTP servers and clients in *Node.js* it is necessary to install this module. After requiring this module, the server component of this project is created in the following way, listening to port 15108.

```
http.createServer(ListenFunction).listen(15108);
```

As *Node.js* is aiming to provide all functionalities of HTTP applications, the HTTP API allows parsing messages into headers but it does not parse the actual header. HTTP headers are represented in an object like this where all keys are lowercased.

```
{ 'content-length': '123',
'content-type': 'text/plain',
'connection': 'keep-alive',
'host': 'mysite.com',
'accept': '*/*' }
```

url

URL strings can contain different kinds of meaning information which are divided into components like protocol type, authentication, hostname and portm pathnames and hashes. When this URL string is parsed it results in a URL object which contains the information about these components.

[40] (Node.js Foundation, 2017)

This module called “url” provides methods to parse URL strings and handle URL objects. In the implemented solution of this thesis the needed method was obtaining the path name of the URL object so later on it can be distinguished between the four different methods License Register, License Start, License Resolve and License Stop.

crypto

As it was needed to implement lots of cryptographic techniques, the crypto module which is in the standard library of *Node.js* was used for this purpose. It provides cryptographic functionalities that include hash methods, as well as functions for encryption and decryption and verifying functions.

It supports methods needed for AES, RSA and SHA1 encryption and decryption used for securing the provided licensing system. Additionally there exists the CFB encryption block cipher mode (see Chapter 1.4.2) for AES used in the implementation.

xml2js

On the server side implementation the xml file was parsed using methods from the module “xml2js” which is a rather simple XML to *JavaScript* converter. It is used to convert all host information saved in XML format into a *JavaScript* object which is later on read and modified.

3.3.2 Server side – basic

The source code of Web Service is implemented in two *JavaScript* files. All database related functions are stored in a file called *“LicDb.js”* end exported in a *module.exports* object. The main application is placed in *“LicServer.js”*. At the beginning of *“LicServer.js”* all extern modules are imported using a standard *JavaScript* **require** call:

```
//////////////////////////////
//
// Modules
//
//////////////////////////////

var http      = require('http');
var url       = require('url');
var crypto    = require('crypto');
var xml2js    = require('xml2js');
var licdb     = require('./LicDb');
var constants = require("constants");
var fs        = require("fs");
```

The main method of the Web Server is opening a license database and starting the HTTP server at the given port. The path of the license database and listening port are given as arguments. This is part of a Web Server configuration stored in a configuration file. The listening callback function "*ListenFunction*" is registered as a callback function, and will be called each time when new requests are detected.

```
function main( database_filename, port ) {

      licdb.Database.Open( database_filename );

      http.createServer(ListenFunction).listen( port );
      console.log('Server: Started');
}
```

When the new client request arrives, the listening function will classify a request by URL and transfer request to the corresponding function.

```
//////////////////////////
// HTTP listening function
//////////////////////////

function ListenFunction(request, response){
      var uri = url.parse(request.url).pathname;

      if(uri === '/License/Start'){
            ProductStart(request, response);
      }
      else if(uri === '/License/Stop'){
            ProductStop(request, response);
      }
      else if(uri === '/License/Register'){
            ProductRegister(request, response);
      }
      else if(uri === '/License/Resolve'){
            ProductResolve(request, response);
      }
      else{
            UnknownURL(request, response);
      }
}
```

Client requests are analyzed within a separate function. The response is prepared and returned to the client.

3.3.3 Server side - Product Register

The listening function "*ProductRegister*" is declared using special the keyword "*async*". This keyword is a standard extension of *Node.js*, started with version 7.6.0 and younger. Each function declared as "*async*" will allow using the keyword "*await*" in its body. This will be used each time when it needs to be "*waited*" until a SQLITE database finishes its asynchronous execution of SQL statement(s).

```
//////////////////////////
// Product Register
//////////////////////////

async function ProductRegister(request, response){
        // code …
}
```

In **Step 1** of the workflow, the database transaction is started.

Per design, the request header "*App-Cipher*" should contain the AES-CFB key, encrypted with the public key. All other data transferred within request and response is encrypted using this AES-CFB key. Therefore at the very beginning, the header "*App-Cipher*" is extracted in **Step 2 (a)**:

```
// prepare data container
var record = new Object;

// read encrypted AES key
record.app_cipher_base64 = request.headers['app-cipher'];
if( !record.app_cipher_base64 )
{
        console.log( "App-Cipfer doesn't exist")

        response.writeHead(404, {'Content-Type': 'text/plain'});
        response.write('404 Not found\n');
        response.end();
        return;
}
```

After extracting the "*App-Cipher*" header, it must be decrypted using the RSA private key, which is known. The private key is stored at the server side in a PEM format and it is a standard part of server configuration. A PEM encoded file can be opened with any standard text editor. It contains very distinct headers and footers, and the content is encoded in Base64. After reading the private key from PEM file, *JavaScript* object containing private key data looks something like this:

```
var public_key = {
"key" : "-----BEGIN RSA PRIVATE KEY-----\r\n"+
        "MIHdMA0GCSqGSIb3DQEBAQUAA4HLADCBxwKBwQDEQiPiUFhNFdiOyRgHWeRqK26ReZMQJLFW\r\n"+
        "WcrlBFIL0CEmR8wMKE9MVoAe3CQ80c5/E4AElLCKLiL7dTGtOoD6Ytn95df7MEHzcuI83i/n\r\n"+
        …
        "KrBE5P45FaWVDb5tUem8D7Q5nXLHto7kXjAVXuG0DMYpROX3TYFMe3TDrp0AyZ8di4z8wLfP\r\n"+
        "eviT2vPlpLJ0TYcVC6Eg7SC1cWYefPYspYcDRM4ZqGr4/nQBDQKN7VnGKx6BvjIXisqruYOA\r\n"+
        "UUJkXQ8CARE=\r\n"+
                          "-----END RSA PRIVATE KEY-----\r\n",
"padding" : constants.RSA_NO_PADDING }
};
```

So it is an object with two items – "key" containing Base64 encoded private key and "padding" defining how padding is used. Knowing the RSA Private Key, the RSA decryption can be done. For all cryptography needs, the *Node.js* module *Crypto* is used, which is a part of standard *Node.js* installation. The decryption is done in **Step 2 (b)**:

```
// convert Base64 to bytes
record.app_cipher = Buffer.from(record.app_cipher_base64, 'base64');

// decrypt AES key using RSA private key
record.app_aes = crypto.privateDecrypt( private_key, record.app_cipher );
```

```
// convert AES key to Base64
record.app_aes_base64 = Buffer.from(record.app_aes).toString('base64');
```

After a successful RSA decryption, the result is a AES-CFB key. All other response headers will be extracted in **Step 3**:

```
// Read application group
record.app_group = request.headers['app-group'];

// Read application name
record.app_name = request.headers['app-name'];

// Read application key
record.app_key_cipher_base64 = request.headers['app-key'];
record.app_key_cipher = Buffer.from(record.app_key_cipher_base64, 'base64');

// Read user password
record.app_pwd_cipher_base64 = request.headers['app-pwd'];
record.app_pwd_cipher = Buffer.from(record.app_pwd_cipher_base64, 'base64');

// Read application process ID
record.app_pid_cipher_base64 = request.headers['app-pid'];
record.app_pid_cipher = Buffer.from(record.app_pid_cipher_base64, 'base64');

// Read host info
record.host_info_cipher_base64 = request.headers['host-info'];
record.host_info_cipher = Buffer.from(record.host_info_cipher_base64, 'base64');
```

Now all headers using AES-CFB can get decrypted, also in **Step 3**:

```
// Prepare Uncipher IV vector
// In real implementation, this IV vector should be uniquely chosen and not zero
var iv  = Buffer.from([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])

// Decrypt App-Key
var aes = crypto.createDecipheriv( 'aes-192-cfb', record.app_aes, iv );
record.app_key = aes.update( record.app_key_cipher_base64, "base64", "utf-8" );
record.app_key += aes.final();

// Decrypt App-Pwd
var aes = crypto.createDecipheriv( 'aes-192-cfb', record.app_aes, iv );
record.app_pwd = aes.update( record.app_pwd_cipher_base64, "base64", "utf-8" );
record.app_pwd += aes.final();

// Decrypt App-Pid
var aes = crypto.createDecipheriv( 'aes-192-cfb', record.app_aes, iv );
record.app_pid = aes.update( record.app_pid_cipher_base64, "base64", "utf-8" );
record.app_pid += aes.final();

// Decrypt Host-Info
var aes = crypto.createDecipheriv( 'aes-192-cfb', record.app_aes, iv );
record.host_info = aes.update( record.host_info_cipher_base64, "base64", "utf-8"
);
record.host_info += aes.final();
```

The host information is transferred in *XML format*. It is decided to convert it from XML to

JSON, as working with *JSON* format is very easy in *JavaScript*. This is done in **Step 4** using the module mentioned before in chapter *3.3.1* xml2js:

```
// convert host info from XML to JSON
var parser = new xml2js.Parser();
parser.parseString( record.host_info, function (err, result) {
            record.host_info_json = result;
      });
};
```

In **Step 5** the host information is extracted from XML data

```
// extract host item from fingerprint node
var fingerprint  = record.host_info_json["Host-ID"]["Fingerprint"];
record.disk_id   = fingerprint[0].$['Disk-ID'];
record.vol_id    = fingerprint[0].$['Vol-ID'];
record.cpu_id    = fingerprint[0].$['Cpu-ID'];
record.mac_id    = fingerprint[0].$['Mac-ID'];
record.win_id    = fingerprint[0].$['Win-ID'];
record.host_name = fingerprint[0].$['Host-Name'];
record.host_time = fingerprint[0].$['Host-Time'];
```

In the next **Step 6 (a)** it is searched for a program with a given name, and "*404 Not found*" is returned if this procedure fails:

```
// find program by name
var prog_id = await licdb.Prog.Find( record.app_name );

// Program not found
if( !prog_id )
{
      console.log( "ProductRegister: Error: Prog not found" );

      response.writeHead(404, {'Content-Type': 'text/plain'});
      response.write('404 Not found\n');
      response.end();

      return;
}
```

Now the license for a given program has to be extracted **Step 6 (b)**. For this purpose a pair of information is used which are the program id (prog_id) and license activation key, extracted and decrypted from HTTP header. These two values build composite SQL key, used as secondary key in the SQL table "*Lic*":

```
// find license by name
var lic_id = await licdb.Lic.Find( prog_id, record.app_key );

// License not found
if( !lic_id )
{
      console.log( "ProductRegister: Error: License not found" );

      response.writeHead(404, {'Content-Type': 'text/plain'});
```

```
        response.write('404 Not found\n');
        response.end();

        return;
}
```

In order to decide whether a license can be assigned to a host or not, the information if a license is already assigned to any host is needed – **Step 6 (c)**:

```
// get host with license
var host_lock_id = await licdb.Lic.HostGet( lic_id );
```

Finally it must be known if the host, which is requesting the license, is already registered in the database or is a completely new. The basic information which is needed is a hash value made out of information about the host – this is the so called host fingerprint. Again, one can use the crypto module and prepare the *SHA1* typed *hash* as follows in **Step 7 (a)**:

```
// create host_key
var sha1 = crypto.createHash('sha1');
sha1.update( record.disk_id );
sha1.update( record.vol_id );
sha1.update( record.cpu_id );
sha1.update( record.mac_id );
sha1.update( record.win_id );
sha1.update( record.host_name );
var host_key_hex = sha1.digest('hex').toUpperCase();
var host_key = "{" +
                   host_key_hex.substring(0,8) +
                   "-" +
                   host_key_hex.substring(8,16) +
                   "-" +
                   host_key_hex.substring(16,24) +
                   "-" +
                   host_key_hex.substring(24,32) +
                   "-" +
                   host_key_hex.substring(32,40) +
                   "}";
```

The final hash looks something like this: {B65CD32E-0A83E943-2C781952-CA06ACDC-18A5989E} and is unique for different host hardware. The table HOSTINFO in the database is using host hashes as unique column. Using this created hash, the database is searched for the host with a same hardware in **Step 7 (b)**:

```
// get host info id
var hostinfo_id = await licdb.HostInfo.Find( host_key );

// get host id
var host_id = 0;
if( hostinfo_id )
{
        host_id = await licdb.HostInfo.HostGet( hostinfo_id );
}
```

The registration request is accepted or refused **in Step 8**. If the license is already assigned to one host, and that host is different than the one calling for the license registration, the license registration will be refused:

```
// license is already assigned
if( host_lock_id && (host_id != host_lock_id) )
{
      console.log( " License is already assigned on other machine ");

      response.writeHead( 409, {'Content-Type': 'text/plain'});
      response.write( “409 Conflict” );
      response.end();
      return;
}
```

Before finally assigning the license to the host it must be checked in **Step 9** if it is a new one. If yes, a new row must be added in the table HOST and HOSTINFO as well as filling all columns with data:

```
// create new host
if( !host_id )
{
      console.log( "creating new host ");

      // add new host
      host_id = await licdb.Host.Add( );
      await licdb.Host.Name( host_id, record.host_name);

      // add new hostinfo
      var hostinfo_id = await licdb.HostInfo.Add( host_key );

      await licdb.HostInfo.HostID( hostinfo_id, host_id );
      await licdb.HostInfo.Name( hostinfo_id, record.host_name );
      await licdb.HostInfo.CpuID( hostinfo_id, record.cpu_id );
      await licdb.HostInfo.DiskID( hostinfo_id, record.disk_id );
      await licdb.HostInfo.VolID( hostinfo_id, record.vol_id );
      await licdb.HostInfo.MacID( hostinfo_id, record.mac_id );
      await licdb.HostInfo.WinID( hostinfo_id, record.win_id );
      await licdb.HostInfo.WindID( hostinfo_id, record.win_did );
      await licdb.HostInfo.ActTime( hostinfo_id, record.host_time );

}
```

In the next **Step 10** the license is assigned to the host and the activation time of the license is updated:

```
// assign host to license
await licdb.Lic.HostID( lic_id, host_id );
await licdb.Lic.ActTime( lic_id, record.host_time );
```

The final action in **Step 11** is to log the “*Register*” activity in the LICUSAGE table. The main reason of storing this information is to have the possibility to follow histories of license usages over the time:

```
// log usage
var licusage_id = await licdb.LicUsage.Add();

await licdb.LicUsage.LicID( licusage_id, lic_id );
await licdb.LicUsage.Time( licusage_id, record.host_time );
await licdb.LicUsage.Action( licusage_id, "REGISTER" );
await licdb.LicUsage.HostInfo( licusage_id, hostinfo_id );
await licdb.LicUsage.Pid( licusage_id, record.app_pid );
var ip = request.headers['x-forwarded-for'] || request.connection.remoteAddress;
await licdb.LicUsage.IP( licusage_id, ip);
```

At this stage all changes in the database are finished and the product licensing procedure is now completed. Therefore the database transaction is committed in **Step 12**.

In **Step 13** the license file is generated in *XML format*, using corresponding functions from the LicDb.js module and encrypted with AES-CFB cipher:

```
// generate XML license
var xml = await licdb.Lic.XML( lic_id );

// Encrypt XML License
var aes = crypto.createCipheriv( 'aes-192-cfb', record.app_aes, iv );
var xml_cipher = aes.update( xml, "utf-8", "base64" );
xml_cipher += aes.final( "base64" );
```

In **Step 14** the encrypted license is returned in a response header "*App-Lic*":

```
// prepare response
response.setHeader( "App-Lic", xml_cipher );
response.writeHead( 200, {'Content-Type': 'text/plain'});
response.end();
```

Step by step workflow:

The following workflow shows a step by step of implemented methods of Product Register.

1. [server] ***Begin transaction***

2. [server] ***Get client AES key***
   a. Read request header "*App-Cipher*"
   b. Decrypt AES key using RSA private key
   c. If something fails: rollback and return error message "404 Not found"

3. [server] ***Get other request headers***
   a. Read header "App-Group" that contains the application group
   b. Read header "App-Name" that contains application name
   c. Read and decrypt header "App-Key" that contains application key
   d. Read and decrypt header "App-Pwd" that contains the user password
   e. Read and decrypt header "App-Pid" that contains the user process ID
   f. Read and decrypt header "Host-Info" that contains the information about the host in XML format
   g. If something fails: rollback and return error message "404 Not found"

4. [server] ***Check validity of host XML***

5. [server] ***Extract host information from XML data***
   a. Read host "Disk-ID" containing disk ID of host computer
   b. Read host "Vol-ID" containing volume ID of host computer
   c. Read host "Cpu-ID" containing cpu ID of host computer
   d. Read host "Mac-ID" containing mac ID of host computer

   e. Read host “Win-ID” containing Win ID of host computer
   f. Read host “Host-Name” containing the name of the host
   g. Read host “Host-Time” containing the current time of the host in GMT+0 reference

6. [server] ***Get app, license ID and licensed host ID***
   a. Search for program ID in database using “App-Group” and “App-Name”
   b. Search for license ID in database using program ID and “App-Key”
   c. Get host ID in database with assigned license ID (can be NULL)
   d. If something fails: rollback and return error message “404 Not found”

7. [server] ***Get host ID with host fingerprint***
   a. Create host key by using SHA1 as hash function on all host information elements (except Host-Time)
   b. Search if host with same fingerprint already exists in database

8. [server] ***Check if license can be assigned***
   a. Case 1: License not used
      i. License can be assigned
   b. Case 2: License already used at same host
      i. License can be assigned
   c. Case 3: License already used at other host (resolve case)
      i. rollback and return message “409 Conflict”

9. [server] ***Update host information in database***
   a. If host doesn’t exist, add new host in database with all information

10. [server] ***Update license information in database***
    a. Set license that license is assigned to the host
    b. Set license activation time to “Host-Time”

11. [server] ***Update license usage information in database***
    a. Insert new usage information using license ID, activation time, host ID, host process ID, host IP address, action type = “RESOLVE”

12. [server] ***Commit transaction***

13. [server] ***Generate license***
    a. Read license information from database into XML file
    b. Encrypt license using AES key

14. [server] ***Prepare HTTP response***
    a. Add response header “App-Lic” with encrypted XML license
    b. Return message “200 OK”

3.3.4 Server side - Product Resolve

Within “*ProductResolve*” the conflict situation must be resolved. This occurs in the case when the license is already assigned to a host and the user is trying to assign it to another host. Per design, the same license cannot be used at more than one host.

Therefore, in such a situation, the conflict can be only resolved by “moving” the license from one host to the other. The listening function “*ProductResolve*” is declared similar to “*ProductRegister*” and is therefore also very similar in its execution steps. The main difference is in **Step 8 –** where the check is done if the license can be assigned to the host or not. For this purpose, the resolve password (which is received by asking the user at client side) is compared with the resolve password in database. The licensing is successful only if the passwords match:

```
// try to use license on other machine
if( host_id != host_lock_id )
{
      console.log( "try to use license on other machine ");

      // get pwd from database
      var lic_pwd = await licdb.Lic.PwdGet( lic_id );

      // compare passwords
      if( lic_pwd != record.app_pwd )
      {
            response.writeHead( 409, {'Content-Type': 'text/plain'});
            response.end();
            return;
      }

      // reset locked host
      host_lock_id = 0;
}
```

Step by step workflow:

The following workflow shows a step by step of implemented methods of Product Resolve.

1. [server] ***Begin transaction*** (same as in Product Register)
2. [server] ***Get client AES key*** (same as in Product Register)
3. [server] ***Get other request headers*** (same as in Product Register)
4. [server] ***Check validity of host XML*** (same as in Product Register)
5. [server] ***Extract host information from XML data*** (same as in Product Register)
6. [server] ***Get app, license and licensed host ID*** (same as in Product Register)
7. [server] ***Get host ID with host fingerprint*** (same as in Product Register)
8. [server] ***Check if license can be assigned***
   a. Case 1: License not used
      i. License can be assigned
   b. Case 2: License already used at same host
      i. License can be assigned
   c. Case 3: License already used at other host (resolve case)
      i. Get license password from database
      ii. Compare with password “App-Pwd”
      iii. If password doesn’t match: rollback and return message “409 Conflict”
      iv. License can be assigned
9. [server] ***Update host information in database*** (same as in Product Register)
10. [server] ***Update license information in database*** (same as in Product Register)

11. [server] ***Update license usage information in database***
    a. Insert new usage information using license ID, activation time, host ID, host process ID, host IP address, action type = “RESOLVE”

12. [server] ***Commit transaction*** (same as in Product Register)

13. [server] ***Generate license*** (same as in Product Register)

14. [server] ***Prepare HTTP response*** (same as in Product Register)

3.3.5 Server side - Product Start

The “*ProductStart*” method is called at each start of already registered software. At the client side the license is already properly installed and it is successfully decrypted using host fingerprint. However, several checks must be checked at the server side. Since the last successful start the license state can be different at server side:

- The license can be “moved” to another host – current host is not registered anymore;
- The license content can be “changed” – the content of local license is outdated;

The **steps 1 – 7** in workflow are same as in “Product Register”. The main difference is in **Step 8,** where the above two checks are executed.

Message “*200 OK*” is returned only in the case if the license is still there and not outdated.

<u>Step by step workflow:</u>

The following workflow shows a step by step of implemented methods of Product Start.

1. [server] ***Begin transaction*** (same as in Product Register)

2. [server] ***Get client AES key*** (same as in Product Register)

3. [server] ***Get other request headers*** (same as in Product Register)

4. [server] ***Check validity of host XML*** (same as in Product Register)

5. [server] ***Extract host information from XML data*** (same as in Product Register)

6. [server] ***Get app, license and licensed host ID*** (same as in Product Register)

7. [server] ***Get host ID with host fingerprint*** (same as in Product Register)

8. [server] ***Check if license can be started***
    a. If host ID with assigned license is NULL return error message “404 Not found”
    b. If host ID with assigned license is not equal to calling host ID return error message “404 Not found”
    c. Read license information from database into XML file
    d. Compare with received license and return error message “406 Not Acceptable”

9. [server] ***Update license usage information in database***
    a. Insert new usage information using license ID, activation time, host ID, host process ID, host IP address, action type = “START”

10. [server] ***Commit transaction***

11. [server] ***Prepare HTTP response***
    a. Return message “200 OK”

3.3.6 Server side - Product Stop

The “Product Stop” method is called at each exit of the already registered license. At the client side the license is already properly installed, it is successfully decrypted using host fingerprint and execution of the application is started.

Theoretically, in the time between a program start and program stop, the license state could be changed on the server side – similar as explained in “Product Start”. In this case the license could be moved to another host or the content of the license could be changed.

However, due to practical reasons it is decided to not pay further attention to the detected problem at the client side.

Step by step workflow:

The following explanation shows a step by step workflow of Product Stop.

1. [server] ***Begin transaction*** (same as in Product Register)

2. [server] ***Get client AES key*** (same as in Product Register)

3. [server] ***Get other request headers*** (same as in Product Register)

4. [server] ***Check validity of host XML*** (same as in Product Register)

5. [server] ***Extract host information from XML data*** (same as in Product Register)

6. [server] ***Get app, license and licensed host ID*** (same as in Product Register)

7. [server] ***Get host ID with host fingerprint*** (same as in Product Register)

8. [server] ***Check if license can be stopped*** (same as in Product Register)

9. [server] ***Update license usage information in database***
    a. Insert new usage information using license ID, activation time, host ID, host process ID, host IP address, action type = “STOP”

10. [server] ***Commit transaction*** (same as in Product Register)

11. [server] ***Prepare HTTP response***
    a. Return message “200 OK”

4 Security Issues [41]

Lacks of security in a software system can be devastating and lead to economical issues but also legal ones. It is really difficult and almost impossible to claim that a software protection is 100% secure and working, as unexpected attacks can always occur and be invented. As it is not possible to foresee all these problems of a system, some main and most likely issues will be discussed.

Client device:

Not all security issues are from the licensing system side as one might think. Security of systems means, that all components involved have to be securely stored and used. This also implicates, that in the case of multiple users working on one computer, problems can occur if users do not log out when finished. In this case this can lead to wrong intended usage of a software. As mentioned before, this is not a real security issue from the licensing framework.

Another criteria for a smoothly running licensing system, is to ensure fingerprints can be used for authentication of a license instance. As for that it is needed to retrieve hardware information of serial numbers of the host computers, it has to be enabled that slight changes like hard disk replacements do not stop the software from working.

Database isolation:

Generally speaking three types of database approaches can be designed, where each has advantages and disadvantages.

In the first case there are *separated databases* for each end user, which makes this system the most secure way. It allows separation and limitation from the beginning for each end user's own data by eliminating the risk of combining that information in the same database. Disadvantages might be the maintainability on the server side and this centralized management approach becomes inconvenient. It could be an alternative solution for cases in which the isolation of data is needed like for banks or financing solutions. In case this project's implementation is going to be used for these kind of solutions, this is something that might be considered for changing.

In the second case the database and its design provide a *shared schema* for all end users, where end users are identified by ID- columns. Advantages of this method are that it does not require that many resources as the other solutions and is rather simple in terms of implementation. Disadvantages might be, that customizing parts of the database for specific users is very difficult in this case. In the implemented licensing system of this thesis, that factor is not decisive as those changes are not expected for this particular protected software which is why this approach was chosen.

The third method for the database design could in fact be an improvement to the existing implemented one. It uses the same database for all end users, but with separated schemas for each one of them. This method is performance wise probably the best approach, as it

[41] (Wikberg, 2010)

reduces the needed recourses needed (regarding the first method “separated database”) but at the same time allows single servers to adapt rather easy for users with specific requirements (regarding the second method).

In this particular case of implementation the second method discussed was secure enough, but regarding this research it might be taken into consideration to change to the third method. This would happen in the case of further work regarding selling the software for itself to other software companies.

Local/Network attacks

The key itself is not stored in any data cache or storage on the computer but is created in runtime while using the software. This avoids attacks on local key extraction on the end – user client side.

Nevertheless some classic network attacks could be expected like brute – force attacks on the password which in the case of the thesis would be an attack on the method Register Resolve. As advanced firewalls might prevent part of these attacks from succeeding, in this case a highly secure configuration setting was chosen.

License security

If licenses were to be sent as plaintext messages sent over an insecure network, one could expect a worst case scenario attack for this licensing system. It could lead to attacks on the communicating client and servers, as well as leaked information about companies and hosts. This would give the attacker access to customer’s data and lead to fatal scenarios. It could increase to a big economical problem and loss of big amount of money. For that reason, no plaintext messages are sent over the internet in this hardware- locked licensing system. The license itself is encrypted before transporting it via HTTPS/HTTP protocols.

5 Requirements evaluation

As discussed in chapter 1.6 which described the solution requirements of the implemented project, now the implementation results will be discussed and evaluated.

5.1 General requirements

First it will be analyzed which of the general requirements listed in chapter 1.6.1 were actually fulfilled and how.

GR1 which was about having an overview of the instance was implemented but could be extended in forms of log files. All usages of the licenses are recorded and distinguished between “REGISTER”, “START”, “RESOLVE” and “STOP”. For this purpose an extra table called LICUSAGE is provided.

GR2 which requires the proposition of easy distribution of licenses is also enabled by making the system based on an activation key which can be shared by E-Mail or any other choice of the company.

According to ***GR3*** the system was implemented in a way, that does not make any restrictions to the existing framework and program features. It was integrated in the program by using programming languages that allow that and does not show any kind of limitation to the implementation of the protected software.

It is possible to meet the ***GR4*** requirement, which is talking about deactivating the protection system for special, trusted users. This can be done by making special activation keys, allowing it to start the program without any further checks. Again, this is rather risky as end-users, even if they are trusted persons could unintentionally give away that key making it able for attackers to use the licenses without any limitations.

5.2 Security requirements

The following part evaluates the implementation of the security requirements mentioned in chapter 1.6.2, which are supposed to secure the system. As almost no system could ever be 100% secure, it cannot be assured that the following requirements are fully supported. Nevertheless it is outlined what was done in order to protect it in the best possible way regarding the provided resources.

SR1 is a requirement which expresses the important value of protecting knowledge and work of the owner of a software. As a requirement evaluation the whole resulting system has to be taken into consideration. While implementing it, the main goal was to secure the product and make it more or less impossible for an attacker to run it without the approval of the owner. Therefore it should not be possible to extract specific functionalities or calculation of the program.

SR2 refers to restricting license usages to one host for each license instance. This is enabled by implementing a system which supports hardware- locked licensing only. This functionality was implemented by using hardware fingerprints and checking hardware information before every program start. If the hardware information does not match, no access to the program is allowed, until the licensing conflict is resolved.

Distinguishing between different software products or different versions of the same software products, as required in **SR3,** is fully supported in the provided implementation. Licensed products can be used only if its product type and product version matches the global stored license information, purchased by the license owner. This is enabled by having additional entries in the server side database, which add product types/versions to the license instances.

All information sent over the internet is encrypted, which also covers information about hosts and their companies. Therefore one can state that requirement ***SR4*** is implemented and fulfilled. This is enabled by using a standard private-public RSA key infrastructure and additional AES- CFB encryption layer. This makes it difficult for an attacker to gather any kind of encrypted information.

Due to security issues the company decided, to allow usage of the licensed program only with a stable internet connection as described in ***SR5***. As soon as the server does not properly reply to the request of the client, it is assumed that internet connection is broken and an error message is returned. This principle is consistently followed in the implementation of the licensing system.

SR6 is met as the implemented system is strictly using proven cryptographic methods like AES, RSA and SHA. They are proven to be sufficiently secure for nowadays implementation. Besides that, wide accepted cryptographic libraries were used, the Crypto *Node.js* module on the server side as well as *Crypto++ C++ library* on the client side.

5.3 Licensing system requirements

All licensing system requirements from chapter 1.6.3, are going to be evaluated now.

LR1 requires storing all important information about sold licenses as well as the information about the customer’s company, hosts and products. As a database is designed which covers all that data in tables- this requirement was successfully implemented as desired. Besides that it fits well with the already existing implementation and does not limit the framework in any way.

In order to additionally save all dependencies and associated information to the license instance as asked in ***LR2,*** the implemented relational database works with deep relationship between SQL tables using *PRIMARY, SECONDARY* and *FOREIGN* keys. By referencing each child table to its parent table it is enabled to have all association requested in the requirement. This is implemented for licenses, owners of the licenses (company, host) and types of programs with corresponding program features.

In order to make a properly functioning system it was necessary to provide mechanisms for adding, modifying and finding information in SQL tables as required in ***LR3***. This is ensured by developing set of *JavaScript* methods, based on SQL queries and SQL database schema. The provided set of methods allows modifying and management of database content on the server side.

All information sent over the internet must be formatted in a way which makes it easy and secure for client as well as server to extract it and prepare it for communication. This is also stated in the requirement ***LR4*** which expresses the need for practical but secure format of information storage. This was solved by using *XML format* as a base format, encrypted by *AES cipher*. The client side extracts, decrypts and loads information in memory *XML* file by receiving information or opposite by sending. The server uses the same process, with additional layer of conversion *XML* <-> *JSON* format, while *JSON* is naturally supported by *JavaScript*.

Establishing a secure connection between client and server is of big importance to the implemented system, as stated in ***LR5***. Missing of secure connection could lead to major problems with information leaks, as well as problems when running the program. This secure connection is enabled by installing a license server which is running behind standard apache server. Apache server overtakes complete HTTPS secure layer role. In addition to this, all information sent/received over the HTTPS channel is encrypted in the implementation.

Requirements ***LR6*** and ***LR7*** are met by taking hardware information about the host that is trying to use the software and checking that by communicating with a web service. The detailed process of this is discussed in chapter 3.

5.4 Design requirements

Additionally to all requirements mentioned previously, the company also made some restrictions and requirements concerning the design of the system listed in chapter 1.6.4. The evaluation of these requirements is following now.

The first design requirement ***DR1*** refers to the chosen programming environment for the client side which in this case is required to be C/C++. This requirement is met by using C/C++ with some additional third party libraries that allow implementing the necessary functionalities. These libraries are *wxWidgets*, *libcurl* and C*rypto++, as* discussed and explained in chapter 2.2.

Requirement ***DR2*** insists on a shared relational database which shares one schema for all users. This requirement is met by implementing server side with only one database. *Node.js* with an additional module *Sqlite3* is chosen. *Node.js* is a very powerful *JavaScript* runtime and is considered as one of the most efficient for this kind of systems.

One of the main design requirements was to create a web service that enables node- locked licensing as mentioned in ***DR3***. This was achieved by using *Node.js* together with several additional modules (see Chapter 3.3.1) like module *Crypto* for encryption tasks or module *Http* for standard communication tasks. An additional secure layer of communication client-server is achieved by choosing Apache server to act as a front HTTPS server.

6 Methodology

This chapter aims to describe the actual research process and working process of the given work in a textual as well as a graphical way. For this procedure it was necessary to perform major steps which are: find requirements, research work, create a concept, implement basic structural model, iteratively revise model and adapt it, evaluate work and design theoretical work.

1. ***Find requirements:***

In order to correctly extract the company's requirements for the project implementation, multiple meetings and discussions with the responsible engineers were necessary. The accepted requirements are made by weighting the features of the already existing framework of the company and the most important functionalities that need to be implemented. After fully understanding the company's needs, some modifications in company requirements needed to be made, as not everything was possible to implement in the given time with the provided technical requirements.

2. ***Research work:***

Finding requirements and doing research was partly done in parallel as they needed to be repeatedly adjusted to each other. The research was done by having a look into external available techniques, which are the current state of art. Therefore internet website content, books along with other thesis and technical papers were taken into consideration in order to get a picture of the existing methods of implementation on the global market. Selected references with relevant techniques and proposals were written down for the actual documentation which this thesis provides. In this step several possible combinations of client- server programming languages, platforms and libraries were thought through and discussed with experts in the field.

3. ***Create a concept:***

After creating a general overview of what was needed and how this could be done, the next step was to create a detailed concept of the practical part, which could later on be implemented. Most of the decision making about the structure of the implemented work was done here. This concept already provided major information about each component and their communication. Deep investigation was done about major implementation details, in special which programming languages and additional libraries should be chosen. Again the goal was to cover all needs and functionalities for the required use cases with the available resources. Most of the documentation created in this step was afterwards used for the theoretical work.

4. ***Implement basic structural model:***

In this step the first tries of implementing the system were made. After researching and creating a concept, it was necessary to try those theories also in the practical work. It was tried to close one cycle of communication between all the three components. One can mention that this was also one of the most time- consuming parts. To make one of the mentioned methods like Product Start work, the whole structure of the model had to work including client- server side and the database.

The trial and repeat work was done by improving the system to function properly. During the development of the system, it has been found that some parts of implementations are more difficult to implement as initially predicted by the accepted requirements. Some of these items needed to be changed and revised, which brings this step very close to the next one.

5. ***Revise and adapt:***

As the name of this step says, implementation details had to be adapted and reconsidered as the system was developed. Some error and problematic scenarios are not predictable in advance, despite all possible research work that was preliminarily done. This step was working alternately with step 4 mentioned before which is working on the implementation. The combination of those two steps made it possible for the implementation to run as desired.

6. ***Finalize practical work***

After the implemented system started working, it was needed to get it tested and finally cleaned up. This means that the program was tested on several server – host scenarios and checked if it really works as intended. As during programming several test prints and breakpoints were added in code just for making debugging easier, all those were removed. In addition, more detailed comments were added and code, that was not as properly readable as wished, was simplified making it more readable and understandable.

7. ***Designing theoretical work:***

Having the practical work done, it was time to design the master thesis itself and design the most suitable table of content which covers all topics used and discussed. As mentioned before, documentation from Step 3 was taken and mostly used for the introduction chapter. Besides that it was needed to decide what chapters to add and what information was needed to back up the practical work. After careful research on other thesis written by predecessors of colleagues, a TU Graz template was used to write the thesis itself.

8. ***Writing master thesis***

Once the structure of the document was clear, the introduction chapter which contains the biggest part of the background theory was written. From time to time the design of the document needed to be partially adapted, as some chapters were added or changed. Onwards the description of the theoretical part was created in form of documentation, giving the reader a clear picture of what was done but also why things were done and how. Some code snippets were necessary to add, nevertheless it was tried to keep that on the minimum. After those two main parts were covered, all the other things were added like abstracts, future work and chapters mostly in the very beginning and end of thesis.

Graphical overview:

Figure 6-1 shows an overview of the work process in order to have this resulting master thesis. It shows steps 1-8 in a more graphical way beginning with finding requests to the actual writing of the master thesis. The documentation of the practical work includes a documentation of what was done as well as the theoretical background and researching topics that were needed to be looked into in order to implement such a system properly.

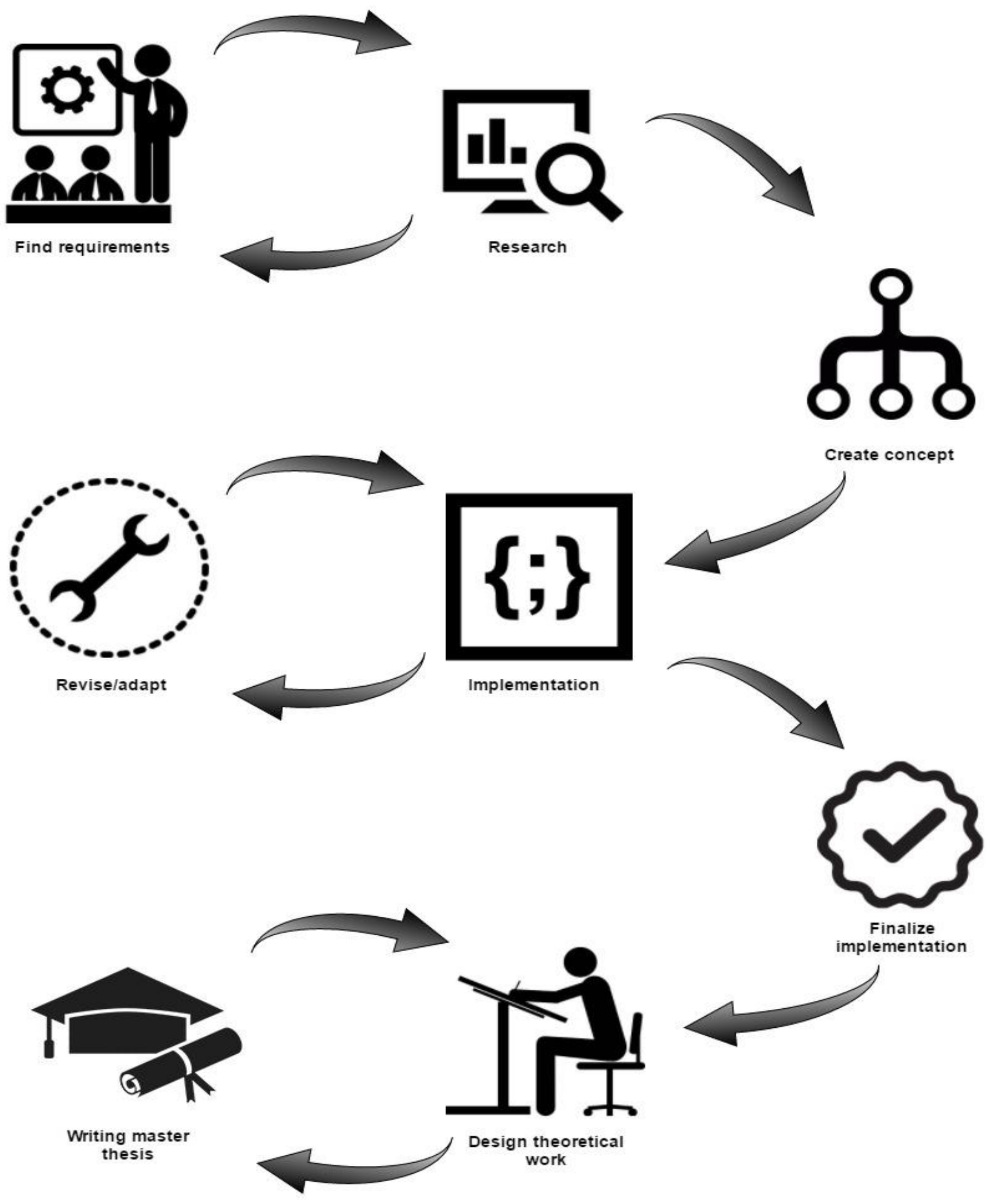


Figure 6-1: Methodology of work

(©Gloria Janjic, 2017)

7 Lessons Learned

This is a summary of the theory, developments and evaluation of the above work and which lessons could be learned from it. They are distinguished into recommendation for the process itself, as well as some technical lessons learned and how to be prepared personally for doing a master project.

7.1 Process/Development

In order to make things easier, the most important step during the process of creating a functioning system is to properly do ***research***. It is recommended to do research already at the initial state, so one can get an estimation of existing solutions and the amount of work which will be needed. It is optimal to use all sorts of media instead of just choosing one. Only relying on web sites and online media might not be the best idea, as information is not always uniform and correct. On the other hand, books might not be as up-to-date as websites, but most of the time they contain information that is uniform, correct and very detailed. Therefore, different kind of sources should be used to make a qualitative research which could help later on in the process.

When developing a system, it first needs to be clarified which ***requirements*** need to be met. Therefore it is recommended to discuss requirements with the people in charge, as soon as possible and to negotiate major and minor items, if needed. Some of the requirements are easier said than done, working in the cloud complicates things even though it might not be that important for all aspects of the whole system. It is always a good idea to have background knowledge and come up with alternative requirements in discussions.

Negotiating and creating a ***concept*** of such a big system, takes more time than one might think. All parties need to be satisfied with the outcome and still the developer needs some freedom when implementing a system. This means, that it is really important to keep some options open as some things could only be understood when doing the implementation.

In general it is really difficult to ***predict implementation time*** needed for specific components. This is even harder for programming languages or libraries you haven't used before. A smart thing to do would be, to arrange enough time for this part of the process.

During the implementation of the system it was learned, that it is easier to build a framework ***skeleton*** first and then add more detailed functionalities later. This might not apply for every implemented project, but in this case it was proofed to be correct.

7.2 Technical

As mentioned before, ***C/C++*** was used as a programming language for the client side implementation. It has to be mentioned, that using C/C++ might need a little bit more time as all third party C/C++ libraries are relative complex and coding is long, but it is also a very straight forward programming language.

Node.js is a very powerful and widely used *JavaScript* runtime and it is used for the server side of the implemented project. Some problems occurred while developing. It was not expected that coding with *Node.js*, due its asynchronous execution, becomes so complex when synchronous- like coding is needed. But after further research, this issue was fixed rather easily.

This documentation and master thesis used Microsoft Word as a ***writing tool***. This decision works great until ~80 pages are reached. For this project, the additional feature of “document structure” was used, which shows a table of content on the left of the document. While this is a rather helpful tool when working on big documents, it might as well cause problems out of unknown reasons and unexpected bugs. Anyways, it was learned that one could consider some alternative writing tools like *LaTex* when creating longer documents.

7.3 Personal

In order to properly work at a project implementation and master thesis document, it is important to be personally and mentally prepared. Therefore it is needed to ***make a plan*** of what needs to be done, in order to keep an overview in mind. Simultaneously it is important not to focus too much on what is left to do, as this might be very overwhelming. On the contrary it is better to keep a clear head, by keeping in mind what is already done.

An additional lesson learned, is not to hesitate ***asking experts*** for input or help. Most people are happy to help out. In general one can say, that any input from experts could be helpful to go on with further development.

In order to begin with the theoretical part of the master work, it is a big help to write a ***table of content*** describing the structure of the thesis and what is going to be discussed in what chapter. This is useful, because when writing a document it is necessary to have some guidelines. A detailed table of content is helping to focus on more important aspects rather than describing less important topics in a too precised matter.

Another very helpful recommendation could be to ***find books and thesis that are related*** to the topic you are elaborating. No matter if the solution of the project is partially or completely different, it could be an inspiring factor to organize and structure own thoughts.

8 Suggestions for Future Work

This chapter covers some possible improvements suggestions that could be applied to the project of this thesis. It covers some details about general ideas, as well as its security and implementation for future work.

8.1 IT Security

As the LMS is supposed to protect the software, it must be mentioned why the IT security of the whole system is important for the current as well as the future work.

Maintenance of cryptographic techniques

Even though an algorithm might be considered “secure” nowadays, it could easily happen that it will not stay that way in the future. Usually algorithms in IT security are firstly only attackable in theory, but no one really knows how long it will take to implement theoretical attacks into practice.

This is due to the increasing computational efficiency of devices which attackers might use in the near future. Some encryption algorithms are only counted secure, as breaking them would take way too long with the current state of the art. Nevertheless, methods which need millions of years to be cracked now, could be breakable in a reasonable time with better computer performance in the future. It is said that processor performances get better every year, which would allow performing attacks faster. As the licensing system relies on secure cryptographic methods, maintaining and adapting them to future possible attacks is necessary.

Run more tests

Additional IT Security test could be made, by giving the system to professional IT- security experts. This way some potential lacks in security could be found and fixed.

Database isolation

This is a more specific way of increasing the IT – security for the implemented project and shows a potential way of how the software could be approved on the sever- side database. The solution is to go away from the solution where one database with the same schema is used for all end- users, and changing it to a solution where there is one database but different schema for each user as discussed in (see Chapter 4).

Consider changing to MySQL

For the current implementation, SQLite worked absolutely fine as the main goal was to make the system work. For future work on the other hand, it might be considered to change the database implementation from SQLITE to MySQL. This is due to the additional feature of MySQL which can protect the access of a database and add some additional security to the communication.

8.2 Implementation

The structure of the program should remain the same, as it is not in the interest of the company to make huge changes in the implementation. Anyways, some additional features could be added giving more options for other clients of the LMS.

Usage without internet connection

For now, the system does not support usages without an internet connection as the company explicitly required internet connection. Anyways, if the program is about to expand to be more generic, this feature could be added in order to make it usable for other organizations that want their programs to run also without any internet connection.

License expiration +30 days

This is a step that could be fixed very easily, if it would be the wish of future users. In this case, there could be a warning message to the end user 30 days before the license expires. With this, unexpected problems with the usage of the software, due to forgetting when the license expired, could be avoided. For sure, it will increase user satisfaction.

Non- compulsory protection

As the current system does not allow unprotected usages of the software, this might be considered as a step for future work. Some companies might have some very trusted users which should be able to work with the software without protection. So it might be necessary to make this a feature which can be enabled and disabled in the most secure way possible. Nevertheless, there is not a 100% secure way to do that and the risks to that should be clear.

8.3 Prepare LMS for the global market

In order to make this system also work for wider set of users or company organizations, it needs to be further developed. The goal would be to make the existing project user friendly by simplifying the usage of the LMS. This could be done with it the following steps.

Adapt LMS

Other companies which might add this licensing system to their existing software, most likely will have different requirements as well as existing frameworks. This means that the system would need to be adapted in terms of usability (see Chapter 8.4). It would be necessary to provide the feature of porting the system to all existing platforms as well as creating a GUI for users of the software.

Selling as software service

This thesis project provides a headstone for future work including selling this software as a service itself. It is believed that this would be a rather profitable project, as for the selling method it is not planned to just sell the software service for a one- time fee. In fact the revenue can be increased constantly, keeping control on the owner of the software licensing system. This can be achieved by doing upgrades of the software and selling licenses one by one and maybe even do trainings for customers. In this case the main goal would be to make this project suitable also for other, already existing solutions.

Check current market

In order to prepare the current software for the global market, it would be necessary to check details of licensing systems which are out there already. Some features have to be identified, which make the licensing system stand out. Unique valuable advantages compared to other software have to be identified or taken into consideration for future development.

Promote unique advantages

Once some unique features are specified for the LMS, they need to be further developed. This will make the product stand out and more appealing for future clients.

Find clients

Once the LMS is turned into a sellable software service, the market has to be expanded by finding new clients for the software. This might involve some additional stuff for marketing and sales which have the needed knowhow in order to do that and succeed.

8.4 Usability

Even though this was not at all the focus of this work, it might be necessary to make some improvements in terms of usability once the company decides to make the LMS available also for other organizations.

Porting to other platforms

Even though the server side of this project is completely developed in *Node.js*, which is portable with all platforms, the situation on the client side looks a bit different. Future work could cover this problematic issue and make the whole system run on all platforms. In case this system is used also for other companies, this would increase the usability of the system by far.

Surveys

Once the LMS is prepared for usage also by other companies, it would be helpful to do some usability surveys. Creating a user- friendly environment should enhance clients, which are normally going to be software companies, to use the licensing system on a daily basis.

9 Summary and Outlook

Creating a secure and properly functioning licensing system is a challenging task. On the today's worldwide market, most software is protected with a licensing system, but users usually only see a little part of what is really happening and what kind of technology is used behind that whole process. Licensing is needed in order to provide a secure sharing of software and to protect the owner's intellectual property. Not only can an implementation o such a system save lots of money but also give the software's creator a clear picture of how, where and when his product is used. It is undoubted an approach that is recommend for any owner of a valuable program that needs to be protected from software piracy.

In this particular case of licensing which was created in the thesis, a server side node locked licensing approach was developed. In order to develop this kind of licensing, it is needed to collect hardware information of the host running the product, like Disk-ID, CPU-ID, Volume-ID, MAC- ID etc. This avoids unintended usages of one software license on several host computers and should provide appropriate security. Using this project's result, companies have complete control and overview over the usage of the software.

One of the most time consuming parts of working on this project was to create a big picture of all components needed and come up with a way of connecting them. The worldwide market of programming languages and libraries that could be used for this purpose ranges widely. Not all of them are useable and suit the purpose of the requirements. To enhance a working implementation solution that perfectly fits, a lot of research before the actual work had to be done.

The system is based on a commonly known principle of Client-Server using request and responses for communication. Besides that there is a need of storing and modifying data information about the license and its users wherefore a database has to be created. In order to add the new implementation into the already existing environment the following design decisions were made after careful evaluation. The client side *ClientAPI* library of this project was completely implemented using *C++* and some additional libraries which are *wxWidget*, *Libcurl* and *Crypto++*. Server side web service and database functionalities where developed fully with *Node.js* which is a commonly used and powerful tool working with Client-Server applications.

One of the core topics of this thesis was cryptography as a lot of information needs to stay secret while transported over the internet. Plenty of cryptographic techniques were taken into consideration until the decision fell on some classical, widely- used algorithms like *RSA*, *AES* and *SHA1*. These ensure a highly secure way of sharing keys, data and licenses. Nevertheless it is very difficult to claim an algorithm is 100% secure as with increasing computational capacities it gets easier to attack systems. This project simply covers all IT-security issues that were required by the company at this particular moment. Anyways enhancing the IT security is also recommended as future work.

Bibliography

{cc}codeconquest. (n.d.). *Client Side vs. Server Side.* Retrieved from http://www.codeconquest.com/website/client-side-vs-server-side/

Bastion. (2015). *Bastion*. Retrieved from http://www.bastioninfotech.com/license-locking-types.html

Carlsson, K. (2014). *Developing an efficient software protection and licensing scheme.* Gothenburg, Sweden: Chalmers.

Chung, L. (n.d.). *utdallas.* Retrieved from https://www.utdallas.edu/~chung/SA/2client.pdf

contributors, W. (2016, Dezember). *wikipedia*. Retrieved from https://en.wikipedia.org/w/index.php?title=Proprietary_software&oldid=755474753

Cryptopp. (2016, November). *Cryptopp*. Retrieved from CTR Mode.

curl. (2010). *The libcurl API*. Retrieved from https://curl.haxx.se/libcurl/c/

Dai, W. (n.d.). *Cryptopp*. Retrieved from https://www.cryptopp.com/

Encyclopedia, T. F. (2016, October). *Wikipedia*. Retrieved from https://en.wikipedia.org/w/index.php?title=Fingerprint_(computing)&oldid=747065925

Higashi, M. (2013, October). *ciphercloud*. Retrieved from https://www.ciphercloud.com/blog/cloud-information-protection-symmetric-vs-asymmetric-encryption/

Ireland, D. (2016, December). *DI Management Services*. Retrieved from http://www.di-mgt.com.au/rsa_alg.html

ITWissen. (2013, November). *Webservice*. Retrieved from http://www.itwissen.info/definition/lexikon/Webservice-WS-web-services.html

Joyent. (2017). *nodejs*. Retrieved from https://nodejs.org/en/

Kitware. (n.d.). *CMake*. Retrieved from https://cmake.org/

Kyrnin, J. (2016, February 16). *about tech*. Retrieved from HTTPS: http://webdesign.about.com/od/http/g/bldefhttps.htm

LimeLM. (2017). *wyDay*. Retrieved from https://wyday.com/limelm/

Lynch, V. (2016, July 29). *The ssl store*. Retrieved from hashed out: https://www.thesslstore.com/blog/difference-sha-1-sha-2-sha-256-hash-algorithms/

MD5 decrypt. (2015, November). Retrieved from http://md5decrypt.net/en/Sha1/

Mendel, F. (n.d.). Applied Cryptography, presentation L4- Block Ciphers and Modes of Operation .

Microsoft. (2007, October). *Description of Symmetric and Asymmetric Encryption*. Retrieved from https://support.microsoft.com/en-us/help/246071/description-of-symmetric-and-asymmetric-encryption msdn. (n.d.). *Microsoft Developer Networ*. Retrieved from Visual C++: https://msdn.microsoft.com/en-us/library/60k1461a(v=vs.100).aspx

Node.js Foundation. (2017). *Node.js*. Retrieved from Overview of Blocking vs Non-Blocking: https://nodejs.org/en/docs/guides/blocking-vs-non-blocking/

Node.js Foundation. (2017). *Node.js*. Retrieved from HTTP: https://nodejs.org/api/http.html

Orlando Vazquez, E. F. (n.d.). *npmjs*. Retrieved from sqlite3 : https://www.npmjs.com/package/sqlite3

Pelzl, C. P. (2010). *Understanding Cryptography.* Springer.

Rechberger, C. *Key management - Part 2, Lecture to IT- Security SS16.* Technical University of Graz.

Roberts, E. (2009). *stanford.edu*. Retrieved from The Intellectual Excitement of Computer Science: https://cs.stanford.edu/people/eroberts/courses/soco/projects/2008-09/colossus/addativeciphers.html

Rouse, M. (2014, June). *TechTarget*. Retrieved from http://searchcio.techtarget.com/definition/software-license

Rouse, M. (2017, March). *TechTarget*. Retrieved from http://searchsecurity.techtarget.com/definition/Advanced-Encryption-Standard

Rouse, M. (2017). *TechTarget*. Retrieved from Diffie-Hellman key exchange (exponential key exchange): http://searchsecurity.techtarget.com/definition/Diffie-Hellman-key-exchange

Shamir, I. (n.d.). *collaboration.* Retrieved from http://collaboration.cmc.ec.gc.ca/science/rpn/biblio/ddj/Website/articles/CUJ/2003/0308/cuj0308shamir/

Silva, J. E. (2003, Jänner). *SANS Institute.* Retrieved from An Overview of Cryptographic Hash Functions and Their Uses: https://www.sans.org/reading-room/whitepapers/vpns/overview-cryptographic-hash-functions-879

Stenberg, D. (n.d.). *curl*. Retrieved from libcurl: https://curl.haxx.se/libcurl/

tech support alert. (2015, November). Retrieved from http://www.techsupportalert.com/content/learn-how-use-windows-registry-editor-regedit-one-easy-lesson.htm

techopedia. (n.d.). *Hash Function*. Retrieved from https://www.techopedia.com/definition/19744/hash-function

Tezer, O. (2014, February 14th). *digitalocean.* Retrieved from https://www.digitalocean.com/community/tutorials/sqlite-vs-mysql-vs-postgresql-a-comparison-of-relational-database-management-systems

tutorialspoint. (2017). *tutorialspoint*. Retrieved from Block Cipher Modes of Operation: https://www.tutorialspoint.com/cryptography/block_cipher_modes_of_operation.htm

Villanueva, J. C. (2015, May 12). *jscape*. Retrieved from An Introduction to Stream Ciphers and Block Ciphers: http://www.jscape.com/blog/stream-cipher-vs-block-cipher

w3schools. (2017). *w3schools*. Retrieved from SQL Database: https://www.w3schools.com/sql/sql_unique.asp

w3schools. (n.d.). *XML Web Services*. Retrieved from http://www.w3schools.com/xml/xml_services.asp

Wikberg, M. (2010). *Software license management from systemintegrator.* Espoo.

Wiki1. (2016, November). *Wikipedia*. Retrieved from contributors, Wikipedia: https://en.wikipedia.org/w/index.php?title=License_manager&oldid=752099287

Wiki2. (2016, October). *Request- Response.* Retrieved from Wikipedia: https://en.wikipedia.org/w/index.php?title=Request%E2%80%93response&oldid=742662460

Wiki3. (2017, January). *MySQL.* Retrieved from https://en.wikipedia.org/w/index.php?title=MySQL&oldid=758266102

Wiki4. (2017, January). *Web service*. Retrieved from https://en.wikipedia.org/wiki/Web_service

wiki4. (2017, March). *Wikipedia, The Free Encyclopedia*. Retrieved from Windows Registry: https://en.wikipedia.org/w/index.php?title=Windows_Registry&oldid=770790722

Wikipedia-Autoren. (2017, February). *Wikipedia*. Retrieved from Wikipedia, Die freie Enzyklopädie: https://de.wikipedia.org/w/index.php?title=Advanced_Encryption_Standard&oldid=162747625

Wikipeida. (2017, May). *Wikipedia, The Free Encyclopedia.* Retrieved from XOR cipher.

List of Figures

List of Tables

List of Abbreviations

A	Alice
B	Bob
CBC	Cipher Block Chaining
CFB	Ciphertext Feedback
CTR	Counter mode
ECB	Electronic code block
IV	Initialization vector
LMS	Licensing management system
NIST	National Institute of Standard and Technology
OTP	One-time pad
RDBMS	Relational Database Management System

Appendix

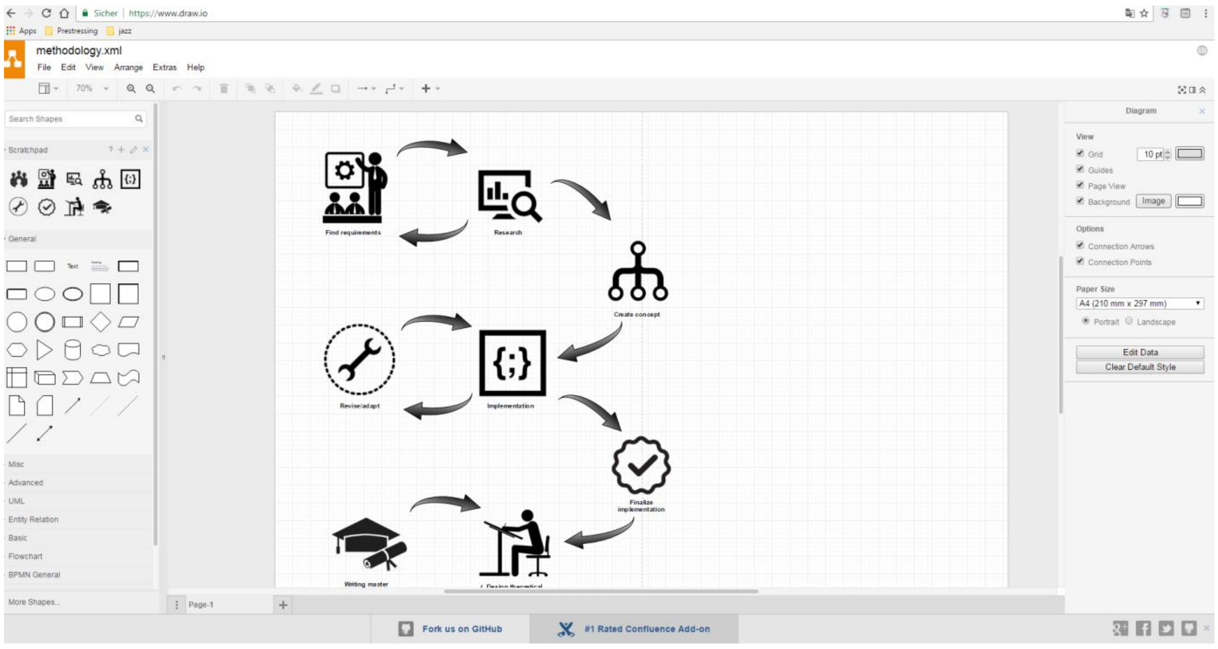
Appendix A: draw.io

Draw.io is a free online diagram software which could be used for flowcharts, org charts, UML and many other types of diagrams. All content created is owned by the one that produced it and can be used for any purposes.

All figures of the thesis which include a copyright of the author were created with draw.io and marked with “(©Gloria Janjic, 2017)”. It has to be mentioned that the author reserves all rights to this figures.

This software was chosen for this purpose, because it offers everything that was needed to create all figures rather easily. One of the advantages is that diagrams can be exported into different file types like XML, as well as JPGE and PNG files. Another very useful feature is, that icons and pictures can be added into the system and included in the diagrams.

The program is very easy and intuitive in usage and absolutely free. Besides that, most versions are wither completely free or only pay-for. It offers online stand- alone versions but also online versions working with Google Drive, Dropbox, OneDrive, Chrome and many more.



In order to try out the program visit https://www.draw.io/, for more information go to https://support.draw.io/.