Developing an efficient software protection and licensing scheme

Master Thesis in Computer Science – algorithms, languages and logic

KRISTOFER CARLSSON

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden, May 2014

Developing an efficient software protection and licensing scheme
KRISTOFER CARLSSON

Examiner: DAG WEDELIN

Department of Computer Science and Engineering
Chalmers University of Technology
SE-412 96 Göteborg
Sweden
Telephone +46 707687894

Department of Computer Science and Engineering
Göteborg, Sweden May 2014

ABSTRACT

Copyright infringement, in everyday language software piracy, is a problem the software industry has faced from its very beginning. Today, the issue is greater than ever as software sells continue to grow and new information sharing technologies simplify file sharing. This thesis investigates how to develop a robust and efficient technical software protection scheme from the viewpoint of one specific business scenario, namely Company A's. The presented solution applies current state of practice techniques based on recommendations from the industry and research. In particular, we use digital signatures and symmetric ciphers as core technologies to develop a lightweight protection scheme against illegal use and redistribution. Stronger protection schemes aiming to stop the more experienced adversary are analyzed as well. In the final analysis we argue that we found a solution provides adequate function and protection for the given business scenario. The claims are based on a security evaluation together with feedback from the Company A.

Table of Contents

Abstract
Table of Contents
1 Introduction ... 1
1.2 Purpose and Goals ... 1
1.3 Scope and Limitations ... 2
1.4 Questions to Answer ... 3
1.5 Method ... 4
1.6 Report Outline ... 5
2 Technical Background ... 6
2.1 Problem Background ... 6
2.2 Licensing Scenarios and Protection Needs ... 7
2.3 Code Based Protections ... 8
2.3.1 Local Installation Schemes ... 8
2.3.2 Online Activation Schemes ... 9
2.3.3 Applying Cryptography for Security ... 10
2.3.4 Encode Readable License Keys ... 11
2.3.5 Issues with License Key Lengths ... 12
2.3.6 Node-Locking with Fingerprints ... 12
2.4 Network and Online Schemes ... 13
2.5 Code Obfuscation ... 14
2.6 Tamper-Proofing ... 14
2.7 Anti-Debugging ... 15
3 Requirement Analysis ... 17
3.1 The Protected Software Products ... 17
3.2 Licensing Domain Model ... 18
3.3 Distribution Methods ... 19
3.4 Solution Requirements ... 19
3.4.1 Security Requirements ... 19
3.4.2 Usability Requirements ... 20
3.4.3 Miscellaneous Requirements ... 21
3.4.4 License Manager Requirements ... 22
4 Solution Design ... 24
4.2 Solution Overview ... 24

4.2.1 The Activation Service 25
4.2.2 The Activation Module 25
4.2.3 The License Manager 26
4.3 The Protection Scheme 26
4.3.1 License Installation Procedure 27
4.3.2 License Certificate Structure 29
4.3.3 Local Certificate Validation 30
4.3.4 Obtaining Reliable Dates 31
4.4 Alternative Solution Designs 31
4.4.1 Time Limited Local Installation Scheme 31
4.4.2 Server Controlled Scheme 33
5 Implementation and System Design 35
5.1 System Overview 35
5.2 The Activation Service as a WCF Service 35
5.3 Activation Service Software Architecture 36
5.4 Activation Module Software Architecture 38
5.6 License Manager Software Architecture 39
6 Discussion 42
6.1 Security Requirements Evaluation 42
6.1.1 Time Limited Local Installation Scheme 42
6.1.2 Server Controlled Scheme Evaluation 43
6.2 Usability Requirements Evaluation 43
6.2.1 Time Limited Local Installation Scheme 44
6.2.2 Server Controlled Scheme 44
6.3 Threats and Types of Adversaries 45
6.4 Project Goals Feedback 46
6.5 Feedback from Company A 47
6.6 Future Work 47
Bibliography 49
Appendix 1 – Service Contacts 52

1 INTRODUCTION

This thesis is about developing an efficient software licensing solution for a project management software, developed and marketed by a Gothenburg based company. For this public report, the company has chosen to be anonymous and referred to as Company A. Their main software product is an information system offering control over complex project setups by simplifying project tracking, project maintenance and quality assurance. Over the past years the number of partners and active licenses has increased; now reaching numbers where previous methods for maintaining and protecting sold licenses have become inadequate.

Some of Company A's customers use the software together with their own subcontractors. It is especially, but not only, these subcontractors' license agreements that have become hard to overlook and if necessary enforce. This, together with recent and unfortunate events, has raised concerns about how sold software is being used.

Furthermore, the company is looking into expanding business in China, where illegitimate use and the attitude towards illegitimate use is a major issue (Buisiness Software Allicance, BSA, 2012) (Rawlinson & Lupton, 2007). Not only is illegitimate use of software in developing economies more common in general, but risks in business-to-business environments are considered higher. Software applications are to a greater extent cracked, if necessary, by local businesses and redistributed in domestic shadow markets (Buisiness Software Allicance, BSA, 2012) (Nill, et al., 2009). Companies aiming to market products in these high risk markets need to address the problem, even if the software is sold strictly business to business.

Currently the sold software lack any kind of protection mechanism making it very hard to control or even know whether it is used according to given license agreements or not. Copying the software is also rather simple, making it easy, should one want, to use or resell the software illegally.

With this background, this thesis work investigates how Company A should develop tools and protection strategies to ease license administration and protect developed software against illegal use and redistribution.

1.2 PURPOSE AND GOALS

Company A is currently struggling to keep track over where and how sold licenses are being used. To address this problem, one aim is to develop a new license management tool tailored for Company A's licensing model. The tool's task is primarily to ease maintenance and documentation needs by automating unnecessary tasks and gather license information into one place, increasing Company A's control over their sold software. We believe that this will reduce the work put into keeping track of where and by whom the software is being used.

More importantly, the work concentrates around finding a way to protect Company A's software products from usage illegal according to given license agreement. The second goal is therefore to design a technical software protection that hinders illegal use and redistribution, but without reducing usability considerably.

As described by Gaël Hachez in his study of software protection tools, the copyright owner's goals with a technical software protection can be divided into five main categories aiming to hinder illegal use, illegal copying, reverse engineering, illegal reuse and illegal modifications (Hachez, 2003). Using this terminology, Company A's specific goals with the technical protection is to hinder the first two, namely;

1. **Illegal usage**. The act of using a program when not authorized. In our case equivalent to not paying for a license but still somehow using the software.

2. **Illegal copying**. Is the act of replicate and spread the software, allowing illegal usage of the software for others.

With the background that Company A suspects some current customers of using the applications without paying and is attempting to establish business in a high risk market; it is our belief that hindering illegal use and redistribution could increase profits.

Together, these two goals nicely align into one single concept. To develop a solution where Company A’s software products are protected by a technical protection mechanism according to license rules created and maintained by the new license management tool. The overall goal with this project is therefore to investigate how this can be done.

Along the way several techniques used for protecting software from illegal use, copying and modification will be investigated. It is also this work’s purpose to investigating which software protection methods could be used, evaluate theories and methods against each other and propose recommendations tailored for Company A’s business situation, recourses and requirements.

1.3 Scope and Limitations

The work concentrates around meeting a specific company’s needs and requirements; omitting the general case. It is therefore a study on how to solve the specific problems in the context of one specific business scenario, namely Company A’s.

It was set from the beginning that Company A desired a software based protection scheme for their products. Other hardware based protections schemes using passive or active dongles, CD/DVDs or Smart-Cards are therefore not considered.

Furthermore, the decision whether or not to actually develop a copyright protection is out of this work’s scope. The topic is briefly discussed in the problem background, see chapter 2, and there is no clear answer to whether or not the gains of a copy-protection outweigh the disadvantages. Company A have however identified that it might, and to test this theory, this thesis is carried out. Thus is it only meaningful to focus on how to design and develop the best licensing solution possible under the given circumstances.

1.4 Questions to answer

From the beginning, the work was set up around answering a concise set of specific questions listed below. The purpose of the questions was to give the project a more precise path to follow throughout the research and development processes. Most of all they are here to clearly identify and distinguish sub-problems and sub-tasks from the general problem formulation and purpose.

Q1. What copy-protection schemes and strategies are known and used as of today? And how do these work?

Knowledge of the current state of the practice methods and techniques used to solve the same or related problems must be investigated in order to propose a well-designed solution. The goal is to gather today's theories and knowledge on how to build software biased protections.

Q2. How does the license model used by Company A look? And how may it change in the future?

It follows naturally that the desired properties of the licensing management tool and copyright protection depend heavily on Company A's licensing model. It must be studied and carefully considered, together with an effort to account for future changes. Part of the problem is therefore to specify the current licensing model.

Q3. What are the desired security properties of the software protection?

The security properties of the copy-protection refer to its ability to resist an attack, its resistance towards being bypassed and cracked. The desired security properties of the protection features must be defined together with the related usability requirements.

Q4. What are the desired end-user requirements regarding usability and availability?

A high level of security will strongly force the users to only use the software according to the given user terms and conditions. A weaker protection will however offer more freedom from the end-users perspective, but of course make the protection easier to break. The desired outcomes from question 3 and 4 heavily contradict each other, and for this reason, they are the most important. Question 3 and question 4 must be carefully considered together and requires a delicate balancing.

Q5. What high-level, functional and non-functional requirements of the license management tool are there?

While previous questions consider specific properties of the protection strategy the requirements for the license management tool should also be defined. These are the high-level requirements, functional requirements and non-functional requirements.

Q6. How shall the license management tool and software protection be designed?

Design refers to both the interface design and software architecture of the license management tool. It also refers to the protection features to be built into existing applications, including aspects such protection scheme and choice of underlying cryptographic algorithms. The answer to Question 6 shall result in:

- The software architecture and design of the license management tool.
- A design of a suitable licensing scheme for Company A including a description of used cryptographic algorithms together with a relevant motivations.

1.5 METHOD

This chapter describes how the distinct phases were carried out throughout the thesis work. After formulating the six questions introduced in the last chapter, a literature study aimed to answer Q1 was conducted; it collected information about current research and state of practice techniques used to stop illegal usage and copying. The study laid the knowledge basis on which choices and decisions concerning protection strategies relied on later.

Information where gathered using mainly two sources, relevant research and literature on the subject together with other companies' approaches to solve the same problem. Naturally, companies do not reveal their protection strategies since the exposed information can be used to break their protections. Companies selling licensing and copy-protection solutions to other developers are however more willing to disclose information in order to sell. The information gathering relied on reading white papers and technical specification for these companies' software protection services.

Simultaneously with the literature study, a series of unstructured interviews, discussions and meetings were held with various stakeholders from Company A. The goals were to answer Q2, Q3, Q4 and Q5 and by so specify Company A's specific needs and requirements. The licensing model and distribution methods used by Company A were identified first, and then used with the interview material to extract high level requirements for the licensing solution.

Based on the requirements analysis and literature review, a couple of viable protection strategies where proposed and discussed. The designs would be the answer to Q6. One of these was then selected and implemented as a prototype. The development was carried out in a single phase with continues feedback and discussions with colleagues and stakeholders at Company A.

Finally, the resulting prototype solution was tested against the requirement and analyzed from a security and usability perspective. The security analysis primarily considered how secure the copy-protection scheme was, that is how resistant it is against an attempt to bypass it. This was analyzed from the attacker's point of view.

1.6 Report Outline

Chapter 1 (Introduction) sets up the report. Company A’s specific problem background is presented here with the project’s purpose and goals. The approach used to solve the problem is also described here.

Chapter 2 (Technical Background) summarize the findings from the literature study carried out at the project’s beginning. The chapter mainly aims to answer question Q1 from chapter 1.4.

Chapter 3 (Requirement Analysis) describes the results from a series of meetings and informal interviews held with stakeholders at Company A with the aim to map out the licensing model used and high-level requirements. Primarily, the chapter answers the questions Q2 to Q5 defined in chapter 1.4.

Chapter 4 (Solution Design) based on the technical background, the domain model and the high level requirements a few general solution ideas where proposed and discussed. These solution concepts address high level system design and protection scheme choices.

Chapter 5 (Implementation and System Design) addresses implementation specific details, i.e. technology specific considerations, low-level software design etc. of the selected and developed solution concept.

Chapter 6 (Discussion) mainly summarize the security and usability analysis of the prototype solution. It also connects the results to the questions from chapter 1.3 in a discussion and suggests future improvements and paths of development.

2 Technical Background

The knowledge basis on which the thesis work relies is presented here; the content of this chapter summarize the necessary information needed to understand the problem domain and the presented solution. First a general problem background is given, introducing the problem domain from a broader perspective. The most common licensing scenarios are then described together with the infringement threats imposed on them. Finally, the subsequent chapters describe how the most important and common protection strategies address these copyright infringement threats.

2.1 Problem Background

Copyright infringement, in everyday language software piracy, is today a large problem. Even though organizations are beginning to get a better understanding of the problem and how to handle it, it remains a big challenge for many profiting from selling software. According to Business Software Alliance's latest annual study of software piracy, the commercial value of unlicensed software is 13.749 billion dollars in Western Europe and 63.456 billion dollars worldwide as of 2011 (Buisiness Software Allicance, BSA, 2012). The series of BSA's annual studies shows that the proportion of illegally used software compared to the total amount has stayed rather stable over the past decades, actually decreased slightly. Overall sells and usage of software have however increased dramatically over the same period; resulting in a large increase of estimated financial losses over the same period. We can also see that the levels of piracy vary greatly between regions and countries. Mature economies tend to have significantly lower rates of software piracy than emerging economies (Yang, et al., 2013). Although BSA's results are debated in the community, methods for measuring financial losses in particular, it still seems safe to assume that significant losses are present. Losses that are likely to continue as levels of software piracy have dropped very little over the past decade (Png, 2010) (Djekic & Loebbecke, 2007).

In response to the risk of losing potential profits, software companies have made many efforts and attempts to invent methods to restrict unauthorized use of their products. However, these methods have so far proven partially ineffective as essential every copy protection has been defeated and cracked (Conner & Rumelt, 1991) (Djekic & Loebbecke, 2007). Furthermore, it has been shown that introducing an efficient copy protection mechanism often decreases the usability and availability of the software, making the software less user-friendly and less appealing to buy. This can in turn result in an income loss instead of increase. (Conner & Rumelt, 1991) (Nill, et al., 2009) (Kathleen Reavis, 1991) (Alexander Nill, 2007). The task of stopping software piracy and developing copy protections is therefore very problematic; and design incorrectly can cause more harm than good. But it is also an important problem to address as good solutions potentially could return substantial amount of otherwise lost income caused by copyright infringement.

2.2 Licensing scenarios and protection needs

Today’s software solutions use a variety of protection mechanisms. Depending on how a company’s licensing model works, the requirements on the protection vary drastically. Proprietary licensed software is naturally the most common type of software to need a protection. For proprietary licensed software, the publishers and copyright owners retain exclusive legal right over the software while end-users are authorized only to use the software after signing an end-user license agreement, shortened EULA. The EULA usually restricts the rights to use, redistribute and modify the software etc. Normally proprietary software is licensed perpetually or as a subscription. Perpetual licensed software is paid once, and the user or users can then use the product without expiration. Subscription licensed software requires renewal of the subscription after a certain period of time. Whilst which model is best to use depends heavily on the context; the subscription model has become more and more popular (Choudhary, 2007) (IDC Analye the future, 2012).

Both perpetual and subscription based software can be node-locked; meaning locked to a specific computer or installation. In this case a protection feature that binds to the specific installation is required to hinder others from using the same license on another computer. It is also common to license proprietary software so that it is locked to a specific person or user, allowing that end-user but no others to use the software. In such cases, the binding should take use of user specific data instead.

Network or floating license schemes are also used frequently; here the software is only allowed to be used by a certain number of concurrent users in one particular intranet. The protection needed for such setups is one that restricts the number concurrent instances of the program running on the intranet. Currently, node locked and network/floating licensing are the most popular methods to use among software vendors (IDC, 2012).

Shareware, for example demo, evaluation or trail versions of a program, normally allows redistribution but requires payment for continuous or full use (Free Software Fundation, 2013). The purpose is to give end-users a limited test version of the real product in the hope that it will later generate more sells of the actual software. There are a couple of commonly used strategies; the software can be limited to a period of time, limited by functionality or implement so called “nagging-dialogs”. The “nagging-dialogs” refers to a strategy where the application extensively reminds the user to buy the actual product with annoying modal dialogs (Sanaei, et al., 2013).

From a security point of view, shareware needs protection in the same manner as full versions of software in order to stop an attacker from unlocking features or enable usage after trail expiration. If a limited version is used while a full version is for sale; one should remove addition features from the limited versions entirely, not disabling them, making them impossible to unlock. However, that would inevitable create the need of a software patch when the trail version is upgraded to a full version.

2.3 Code based protections

Code based protections are a group of methods using cryptography to stop or hinder illegal use and copying. They offer no protection against illegal modification or reverse engineering but can completely stop illegal use and copying under the assumption that the target executables are not modified (Hachez, 2003) (Sanaei, et al., 2013). This assumption is however a bit naive since an experienced adversary can analyze and modify the executables using software tools and assembler debuggers (Wroblewski, 2002) (Shields, 2008). For strong protection, a code based scheme should preferably be complemented with other protecting strategies such code obfuscation. None the less, depending on context and the assumed adversary's expertise, a code based protection is often enough.

All code based protections rely on built-in verification algorithms that require some unique code to allow execution, unlock additional features or enable whatever is protected. This unique code is either implemented as a text string or as a file. This code contains vital information verified by the protected software. This could be anything desired, for example subscription periods, version numbers or features to enable etc. (Sanaei, et al., 2013) (Jan M. Memon, 2007). Commonly, code based protections are divided into two main categories, local installation based and online activation based schemes, each described in its own section below (IDC Analye the future, 2012) (Djekic & Loebbecke, 2007).

2.3.1 Local installation schemes

Local installation schemes are a collection of strategies with the hallmark that the validation of the license code is carried out on the end-users' machine without communication with other systems. Described on a high level, the scheme outlines as follows.

A license code, implemented as file or text string, is generated by the distributor and sent to the end-user. The needed license information must be embedded into the code such that the protected programs can analyze this data during validation. During program execution, the protected program checks if a code has been installed. If that is the case, the saved code is validated, if not, the user is asked to enter a new code. If the code validation succeeds, the program saves the code for further validations (if not already saved) and allows further execution of the application, or unlocks protected features etc. If the validation fails appropriate actions are taken; this would include informing the user of the error, ask for a new code, not allow further execution of the program etc. Depending on how the application installs, the procedure can be embedded into an installation wizard or alike. (Peyravian, et al., 2003) (Sharma, et al., 2010) (Sanaei, et al., 2013)

The scheme's simplicity is both its strength and weakness. From the end-user's perspective, this scheme does not intrude much on usability or privacy compared to other schemes. The only usability concern is that the end-user must remember or save the given code and use it once for each software installation. Privacy is not intruded at all.

However, in terms of security the scheme has some fundamental flaws which might be appalling. Since the unlocking codes are communicated once from the vendor to the end-user, the code must include all license data needed to validate the license. If short product keys (as a text string) are desired, it is hard to fit all required information into the string. See chapter 2.3.5 for more detailed information. Furthermore, because code generation is done separate from code validation, it is impossible give any user specific input to the code generation algorithm that could bind it to a specific user. The user can therefore reuse the key to install the software on any number of devices, which might be illegal or not, depending on the licensing agreements. A user can also easily redistribute the codes together with the software to others.

Because of this, this scheme alone only features a weak protection against illegal copying and usage. Furthermore, if the key is intended for human use, it must be short and readable making it hard or impossible to preserve the little security the scheme provides.

2.3.2 ONLINE ACTIVATION SCHEMES

In common for all activation schemes are that they all rely on an activation procedure to decide if the installed program is legitimate or not (Sanaei, et al., 2013). Most commonly the distributer constructs a unique activation key for each license, but instead of embedding license data into the key itself, the data is saved together with the key by the activation service. The key is only a unique random number used by the activation service to look up the real licenses data.

When the protected software starts, the program checks if an activation result from a previous activation has been saved. If one is found, the program validates it; otherwise a new activation procedure is initialized. During the activation procedure, the license key and possibly additional data are sent to the activation service. Exactly what this additional data is depends on preferences and policies, it could be unique identifies used to node-look the key to that specific computer or other user specific data.

The activation service receives the request and looks up the license data corresponding to the given key and validates it against any desired requirement. If successful, the activation service generates an activation result with the license data embedded in it. This activation result is very similar to the code generated in the local installation scenario. The result is then sent back to the client, who validates it and saves it for future offline validations. Please refer to the following articles and reports for more information (Anckaert, et al., 2004) (Sanaei, et al., 2013) (Hachez, 2003). Once again, how to build a secure activation result with the use of cryptography is covered in chapter 2.3.3.

From the end-user’s perspective, activation schemes impose more restrictions and possibly raise a few privacy concerns. First of all, the software requires communication with an activation service. The communication is normally implemented over internet to a server maintained by the distributor but could possibly be implemented as an automated telephone or email service where the customer activates his product though a series of code exchanges. The online versions obviously require internet connection, which could be inconvenient in some circumstances.

Furthermore, since the scheme allows user data to be collected from the end-user’s machine during activation, the scheme poses a potential privacy intrusion. Even if no sensible data is sent to the activation service, some customers might find that hard to trust. Also, the activation service could keep track of activation attempts, something people might find it unpleasant that their activity is being monitored by the copyright owner. In all, what is collected from the end-user's machine should be carefully considered.

From the distributor's point of view, the possibility to generate activation results based on input from the end-user's computers opens opportunities for stronger protections. By node-locking a particular key to a specific computer using a unique fingerprint generated from the end-user's computer configuration, it is possible to set a limit on the number of allowed activations per license key (Sanaei, et al., 2013) (Hachez, 2003) (LimeLM, 2013) (Licenturion, 2001) (Kügler, 2009). The methods used to apply node-locking are covered in chapter 2.3.6. If the protected program uses unique user data unlikely to changed, that data could be used in a similar way to lock the key to that specific person.

2.3.3 APPLYING CRYPTOGRAPHY FOR SECURITY

Vital for all code based protection schemes are the algorithms used to generate and validate the codes containing the critical license data, may it be a license key, license file or activation result. The most secure practice is to use public key cryptography, possibly but not necessarily together with symmetric ciphers. However, depending on security and usability requirements, lighter protections relying only on symmetric ciphers may be preferable.

To easier reason about the different techniques, let us define some common cryptographic functions on which most schemes rely.

- $E(k, m)$ is a symmetric encryption function. Input is the key k and plaintext message m. Output is the ciphertext.
- $E^{-1}(k, c)$ is then the corresponding decryption function. Giving the same key k and an encrypted message's ciphertext c as input, the plaintext message m is retrieved.
- $S(k_{pri},\ m)$ is the sign function of a digital signature algorithm where k_{pri} is the private key component and m the message to be signed. Output is the signature.
- $V(k_{pub},\ m,\ s)$ is the matching verification function taking the public key component k_{pub}, the message m that was signed and the signature s as input. Output is a Boolean telling weather the verification was successful (or possibly an error message).

Using these functions, one can implement a secure scheme featuring confidentiality, integrity and authenticity using basic cryptographic principles (Jiutao & Guoyuan, 2010) (Sanaei, et al., 2013) (Edlira Martiria, 2012).

Such a scheme could outlines as follows; the distributor holds k_{pri}, k and license data m_d while the client application holds k_{pub} and k. Typically the keys are embedded in the application's code or stored encrypted somewhere. The license key or file is constructed in the following way by the distributor;

1. If license data is sensitive, encrypt the license data $c_d = E(k, m_d)$. Otherwise $c_d = m_d$.

2. Sign's the resulting ciphertext $s_{c_d} = S(k_{pri}, c_d)$.
3. Construct the license code $k_l = c_d \,||\, s_{c_d}$ using some delimiter.

When client application validates the key the following is done;

1. From k_l extract c_d and s_{c_d} using the same delimiter.
2. Ensure authenticity and integrity by verifying the signature, $V(k_{pub}, c_d, s_{c_d})$.
3. If license data is sensitive, obtain license data by decrypting, $m_d = E^{-1}(k, c_d)$, otherwise we already have $m_d = c_d$.
4. Validate the data in m_d against the specified conditions, which would be subscription dates, version numbers etc.
5. If all steps succeed, the license is valid, otherwise it is not.

Similar techniques are used in many real world solutions such as Microsoft Activation Service, Wibu-Systems's CodeMeter project, SafeNet's Sentinel HASP SL and LimeLm protection services (Licenturion, 2001) (Kügler, 2009) (SafeNet, 2010) (LimeLM, 2013). Please refer to these documents for more the details of real world examples.

The reason to rely on public key cryptography instead of a symmetric authenticated encryption (with or without Message Authentication Codes, MACs) is that the distributor can hold k_{pri} secrete. Should integrity and authenticity rely on MACs or plain symmetric encryption it would be possible for an adversary to extract the secrete key from the executables with an assembler debugger. He could then use the same key to craft new valid codes that would unlock the original program. By using digital signatures the private key k_{pri} is kept secret and this type of attack becomes infeasible given the underlying digital signature algorithm is secure.

Regarding choice of algorithms for E, E^{-1}, S and V; any symmetric cipher known to be secure could be used for encryption and decryption, AES for example (FIPS, 2001). A stream ciphers could possibly be used but note that would require unique key-streams for each end-user to be secure. For the digital signatures, any secure scheme such as RSA-PSS, DSA or ECDSA could be picked. For algorithmic details, refer to (RSA Laboratories, 2012) (FIPS, 2013) (Johnson, et al., 2001), respectively.

2.3.4 ENCODE READABLE LICENSE KEYS

If license keys are to be displayed and used by persons, it is desirable to make them both short and readable. For convenience, letters in the license key should not be case sensitive and easily confused numbers and letters should be avoided. For example number 1 can easily be confused with lowercase l or capital O with number 0 and so on. Furthermore, the probability of generating inappropriate words should be minimized.

A sensible solution, adopted by Microsoft, is to use 2346789BCDFGHJKMPQRTVWXY as alphabet, leaving 015AEILNOSUZ unused (Licenturion, 2001). In addition to what has been said, 5 could easily be confused with S, Z with 2 and using two lowercase n in a row could look like a m; these are therefore removed. Vowels AEIOU are avoided to cancel out the possibility of generating meaningful word in most Latin languages. Altogether, this alphabet has a base of 24. It might however be convenient to relax the readability constraints slightly to reach base 32 since it obviously maps easier into binary data, and a larger base will allow more information to be stored in the key.

2.3.5 Issues with License Key Lengths

Keys intended for human use should also be short, which essentially translates into short ciphertexts and signatures in the scheme describe in 2.3.3. This is however hard to achieve if high security is desired. Depending on what information is stored in the key, it might even be impossible due to the limited space in a short text string. Generating short keys is therefore a tradeoff between security and usability.

Assuming that the license key generator use the proposed alphabet from chapter 3.2.4 and are at most 30 letters long, it follows that 137 bits data can be stored in them, $\log_2(24^{30}) = 137.5$. Even relatively long license keys of 36 letters using a larger alphabet of base 32 would only contain $\log_2(32^{36}) = 180$ bits of data. To guarantee integrity, it is required to hash the license data and then sign the hash with a digital signature. Among known algorithms known to be secure, Boneh–Lynn–Shacham signature scheme (BLS) generates the shortest signatures (Boneh, et al., 2004). The scheme's signatures feature a security of n bits using only $2n$ bits long signatures. If 80 security bits are enough, which is questionable for long term security but enough for this purpose, the signature would be 160 bits long, clearly too long to fit into the 180 bit long key together with any meaningful amount license data. Signature compression and signatures schemes with message or partial message recovery could also be used to shorten the key, but if security must be preserved while usability is kept high; it is simply not possible to use short keys. One must therefore choose between using long keys with large alphabets or license files.

An alternative worth mentioning is to sacrifice configurability in order to get shorter keys. This is done by precompile license data into the distributed software programs. The precompiled license data should preferably be encrypted in a way that makes them hard to retrieve or modify. The license data in the key could then be exchanged with a short code used by the protected application to determine which precompiled license data to use.

One could also sacrifice security by using symmetric ciphers only, dropping the digital signature entirely. That would make it possible to create a shorter key. As pointed out before however, an adversary could then extract the private key used for encryption from the application's binaries and use it to generate new valid license keys.

2.3.6 Node-locking with Fingerprints

Node-locking is primarily useful together with the Online Activation Schemes and relies on unique fingerprints generated based on end-user's computer or device. The general principles of node-locking outline as described below. If interested, see the analysis of Microsoft Windows XP protection scheme (Licenturion, 2001) or other real world examples used by SafeNet, LimeLm and Wibu Systems (SafeNet, 2010) (LimeLM, 2013) (Kügler, 2009) for more in-depth information.

Typically, a hardware fingerprint is generated by the protected software before a license code is created by the activation service. This is done by gathering unique identifiers, labels and id numbers from a variety of hardware components and combining these into one unique fingerprint. The fingerprint is then added to the license data sent with the activation request. If the activation succeeds, the activation service adds the fingerprint to the activation response code when creating it. Later, when the protected software validates the given activation code, the software uses the same algorithm to extract a new fingerprint from the computer it runs on. If this generated fingerprint does not match the fingerprint stored in the activation code, the application is running on an unregistered computer. By applying this scheme, the activation service can count and the number of successful activations with a certain license key and limit it to a specific number. (Sanaei, et al., 2013)

The core of this locking algorithm is the generation and validation of fingerprints. For generation, the hardest part is to choose the right identifiers to construct the fingerprint from. The identifiers must be resistant to change since the same fingerprint is generated at every validation run. For example a particular id number of a hardware component might be a better choice than the CPU clock speed, which can be changed in the BIOS settings. The most convenient approach is to use a combination of MAC addresses, serial and product numbers from various hardware components such as CPU, motherboard, hard drives, network cards etc. Some components lack unique ID's while others do not. For example some processor series feature unique ID numbers per CPU but not all. These values should then be encrypted or encoded into a fingerprint using symmetric encryption or some encoding. One could use a one way hash function to create the fingerprint, but that would remove the possibility to compare each component id individually.

During the validation of a fingerprint, it is generally preferable to do the checks towards each id separately and allow some failures as in Microsoft's approach (Licenturion, 2001). This allows end-user to change some components without invalidating the license.

2.4 Network and online schemes

This class of protections relies on a continuously available connection to a license service contacted at every run. According to IDC’s study this is the most common protection scheme adopted by today’s software solutions (IDC Analye the future, 2012). Normally, it is required that the protected software is either continuously connected or connected during every startup to a global or local server responsible controlling client execution. However, other variations exist, where for example clients’ dynamically acts as server in a peer-to-peer setup as proposed by Vineet Sharma et al (Sharma, et al., 2010). In contrast to previously proposed code based schemes, no codes are saved for later offline validations.

The obvious disadvantage is the dependency on a centralized service. However, in setups where the software already depends on a centralized service these types of schemes are very viable. The security advantages are the same here with activation schemes since the client sends data for validation, which could include fingerprints and user specific data. Furthermore, since all clients are required to connect to a centralized service once or many times during every run, it is possible to keep track on the number of concurrent active clients. This opens the opportunity to enforce floating license agreements.

In the special case where this centralized service is maintained by the copyright owner and does important calculations for the client software, the security is increased further. Mainly because the service maintained by the distributor can be fully trusted and is part of the software solution's functionality, thus hard to replace by a cracker. If the protected software does not require a centralized service, one can still apply the scheme moving important calculations to the service as proposed in this article (Sanaei, et al., 2013).

2.5 CODE OBFUSCATION

While previous discussed schemes try to hinder unauthorized execution by applying checks and validations, code obfuscation tries to conceal program behavior. The goal is to make it substantially harder for an experienced adversary to analyze the program and understand how to remove the real protection against illegal use or copying. In short, code obfuscation protect against reverse engineering, illegal reuse and illegal modifications by modifying syntax while preserving semantics.

Code obfuscation techniques are typically divided into four categories, namely layout obfuscation, data obfuscation, control obfuscation and prevent obfuscation. (Wroblewski, 2002) (Jiutao & Guoyuan, 2010).

- **Layout obfuscation** targets the layout of the source or executable code. For example by removing sensible names from methods, classes and other programming entities. It also refers methods used to eliminate comments and other unnecessary constructs from the binary program (Jiutao & Guoyuan, 2010) (Wroblewski, 2002).

- **Data obfuscation** refers to different data modification strategies. This could be splitting a variable into several, chaining encoding formats or change scope of variables to extend their lifetimes. It also includes reordering of methods and various array modifications such splitting, merging and folding (Jiutao & Guoyuan, 2010) (Wroblewski, 2002).

- **Control obfuscation** modifiers or hides statements controlling execution paths in a program. This includes extending conditions of iteration constructs with unnecessary junk conditions, unrolling loops with fixed counters and reordering independent computation blocks or expressions randomly. It also includes duplication of methods and code blocks (Wroblewski, 2002) (Hachez, 2003).

- **Prevent obfuscation** are protections trying to prevent deobfuscation algorithms and decompiles, the opposite of obfuscation (Hachez, 2003).

2.6 TAMPER-PROOFING

Tamper-proofing techniques aim to prevent illegal modifications by directly preventing modification attempts. In reality, one can say that obfuscation and other techniques indirectly feature a kind of tamper resilience as a side effect. However, this section considers specific techniques targeting just tamper-proofing.

The simplest way of detecting modification is to calculate checksums or hashes of the target executables at real-time and compare these to pre computed results (Collberg & Thomborson, 2002). MD5 is for example an excellent algorithm choice for this purpose. There exist many other more complicated detection algorithms, including sophisticated self-modifying schemes; please refer to Collberg & Thomborson paper (Collberg & Thomborson, 2002), Ten et al (Tan, et al., 2007) or (Kisserli & Preneel, 2011) among others for further reading.

To make it harder for an adversary to remove the tamper-proofing protection, detection should preferable be separated from the response mechanism. The idea is to make it hard to back-track the detection algorithm from the code taking action against a positive detection. This can simply be implemented by setting flags which are checked much later down the execution path, on several places, calling individual response implementations. One can also hide this information in more complex data structures to make the detection flags harder to find (Tan, et al., 2007). To further confuse an adversary, one can apply delayed and ambiguous system failures as proposed by Tan et al (Tan, et al., 2007).

Another method is to use encryption to encrypt parts of the executables and during execution decrypt the program (Jiutao & Guoyuan, 2010). As described by Peyravian et al (Peyravian, et al., 2003) this can be combined with an activation based scheme for a rather strong copy-protection. However, self-decrypting and self-modifying behaviors are not possible in all programming languages; Java for example strictly forbids this self-aware behavior for security reasons. This self-modifying behavior is also a common method used by malicious code to hide itself from virus scanners which is why these programs might flag you application as malicious if this kind of behavior is identified, something very indescribable indeed.

2.7 ANTI-DEBUGGING

Anti-debugging is as the name suggests a group of methods used to stop debuggers from attaching to the target process. On its own, anti-debugging techniques increases the difficulty for an adversary to illegally analyze the executables. Tyler Shields explains the most common anti-debugging techniques on the Windows platform in paper “Anti-Debugging – A Developers View” (Shields, 2008) from which most of this information is collected.

The simplest category is API based anti-debugging which relies on API calls hosted by the operation system. In the Windows environment, a number of documented and undocumented function calls to the Windows API can be used for this purpose. They use direct memory locations, registries entries, indication flags and other information to detect if a debugger is attached to a certain process. Examples include *IsDebuggerPresent* which return whether or not a debugger is attached to the caller process by analyzing its Process Environment Block (PEB). One can also try to output to the debugger using the windows function *OutputDebugString.* If such call is successful, a debugger is certainly attached. All these techniques and many more are described in Shields paper and Ferrie’s rigorous anti-debugging guide (Ferrie, 2011) (Shields, 2008). All API based approaches have one disadvantage though; the API calls are commonly used for this exact purpose. As a result, most analyzing tools with assembler debuggers, such as the popular OllyDbg (Yuschuk, 2014), hook these API calls and return falsified values.

Instead of relying on API calls, it is possible to directly access debug flags in the process' PEB, and by so bypass attempts to hook the particular API call. It is also possible to manually set the value of the particular flag byte that indicates a debugger is applied and fetch the value directly afterwards. If the results are different, an analyzing tool is probably hooking the call. Heaps and other data structures are also created slightly different when a process is being debugged, which is visible in the PEB, these could likewise be analyzed.

It is also possible to detect a debugger by detecting breakpoints. When a breakpoint is added, the debugger changes the chosen instruction with an interrupt instruction, which is later replaced back when the breakpoint is hit. The important point is that the executed code is changed. By calculating a hash on the loaded code in memory at runtime, and compare it with a pre computed hash of the original code, it is possible to reveal the debugger.

A program can also try to debug itself at runtime by hosting a child process. Since a process can only be debugged by one attached process simultaneously, the idea is to catch possible errors indicating that the process is already being debugged. Finally, when a program is being step through manually with a debugger, the execution time is magnitudes slower. By measuring execution times, manual stepping can be revealed.

3 Requirement Analysis

This chapter describes Company A's software products, license model and distribution methods. The purpose is to give enough information and context about the company's challenges and business situation to understand and motivate the solution design later on. The result of this analysis is a set of high level software requirements presented near the end of the chapter.

3.1 The Protected Software Products

Company A's software offer support and control over complex project setups by simplifying planning, quality assurance and project tracking. The customers are companies and organizations how build complex products requiring complex project setups. These projects are naturally very hard to manage and Company A's solutions support the customer organizations by highlighting critical information in time supporting decision making in various levels of project management.

The sold software is strongly connected to a model and process around working with quality assurance, project planning and tracking. Because of this, Company A's business incorporates teaching and supporting customers in using this model where the sold applications are only a part, though very important part. As such, the software products are not standalone applications just delivered to the customer, but information systems (IS) requiring support and adaption of Company A's proposed processes to give high business value. Since the products strictly target other businesses, end-users are exclusively employees working for the company purchasing the system. The important note here is that the software protection only needs to stop another business from illegally use or redistribute the software products.

Now focusing on the actual software applications, they come in two versions; a client version and a monolith version. The monolith version works against a local database installed on the end-user's own computer while the client version uses a client server setup against a shared server. Typically the server service is deployed on the customer's site where clients communicate over the organization's intranet. The possible setups are illustrated in figure 1 below. Except from the back-ends, both applications are very similar sharing user interface and most functionality.

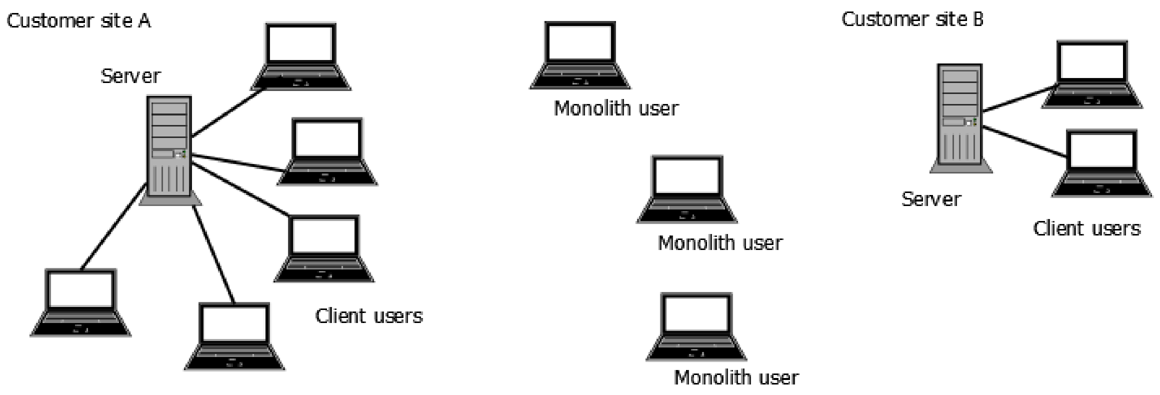


Figure 1. Example of client and monolith setups at two customer sites.

It is also important to note that the back-end software is based on a framework developed and maintained by another company. The protection will therefore primarily focus on protecting the client side code which is fully owned by Company A.

Regarding used programming languages and other technicalities, the applications are develop in Embarcadero's Delphi development environment. Delphi is a complete development environment targeting 32- and 64-bit versions of the Windows platform, Mac OS, iOS and Android (Embarcadero, 2014). Currently Company A's products target only Windows.

3.2 Licensing Domain Model

Company A applies a subscription model where customers renew their subscription after a certain time, typically but not restricted to one year. Included in the subscription are a number of updates each year, support and other services. Some customers buy very many licenses while other buys only a few. If several licenses are sold to the same company at the same time, the licenses are normally bounded into a package with the same subscription period, support etc.

To describe the license model we define seven types of entities; namely *customer organization*, *subscription bundle*, *licenses*, *contact person*, *monolith installation* and *client installation*. These seven domain entities define the entire license model by a set of mutual 'one to many' and 'on to one' relations shown in figure 2.

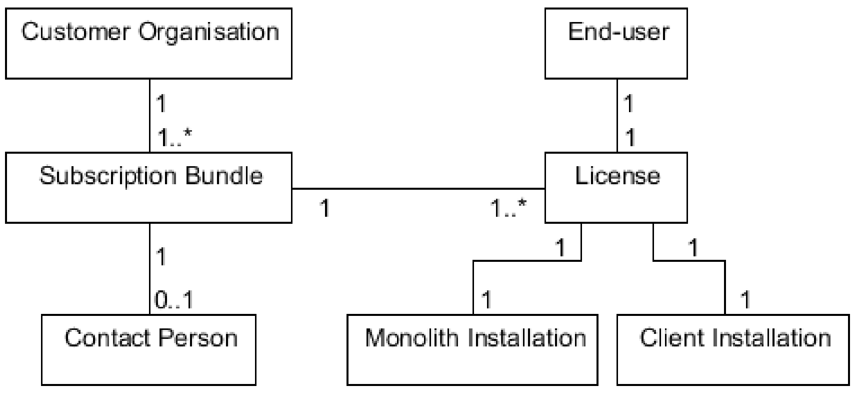


Figure 2. Domain model over Company A's licensing model.

A *license* is defined as the rights for one *end-user* to install and use one copy of both software applications over the license's subscription period. Therefore each *license* is associated to exactly one *end-user*, *monolith* and *client installation*. Naturally, a *monolith*- or *client installation* refers to one installation of each application, respectively. These installations are typically on the same computer, the end-user's working station, but if desired the monolith and client could be installed on different computers. If the end-user wishes to change computer or reinstall the products, he typically contacts Company A who supplies him with the latest version of the software as described in next chapter.

The *end-user* is simply the person who the *license* is licensed. The *customer organization* is the company or organization that ordered one or more licenses. This order can be seen as bundle of one or more licenses, with the same subscription period, thus are orders packed into on *subscription bundle* if the subscription period and *customer organization* is the same.

The *contact person* is the person responsible communication with Company A, this could be the same person as the *end-user* for small customers or a representative in larger setups. The *contact person* is connected to one of the customer organizations where he or she works. However, each customer organization may have several contact persons for different licenses bundles, thus is he connected to a *subscription bundle* instead.

3.3 Distribution Methods

The method used to distribute the server service, client and monolith applications to end-users varies between customers. Some customers prefer to use their own system for maintaining and distributing software to their employees. These customers receive agreed versions of the applications on certain delivery dates. The sent software contains the executables, manuals, various installation and configuration files. These files are then distributed to the end-users through the organization as they please.

Many buy fewer licenses or prefer not to distribute their software through an internal system. Instead Company A's consultants distribute the applications either by sending the required files by email or by actually helping installing the programs on site. The consultants must always have access to the latest versions of the files; files that are copied and emailed to several end-users under different licenses.

Either way, the important point is that in both described scenarios the same files are delivered to many end-users, and that customers shall be able to install the programs with or without the help from Company A's consultants. Furthermore, it is desired not to change the way of distributing the software, aspects which a software protection must take into account.

3.4 Solution Requirements

This section lists the requirements extracted together with Company A with respect to the described background from previous chapters. The requirements are categorized into four categories; security, usability, miscellaneous and license manager requirements. The three first categories refer to requirements of the copy-protection scheme while the fourth category summarizes the functional and non-functional requirements of the license manager tool. Together they give a high level overview of the desired functionality over the entire licensing solution.

3.4.1 Security Requirements

These are the desired security requirements related to the protection mechanism. Security refers to the system's resistance against an attack trying to break or bypass the protection scheme.

ID	Name	Description	Motivation
R1	Protect business value	The client and monolith program shall be protected such that its business value cannot be utilized without	Basically the purpose of any copy-protection, the programs features shall be protected from illegal use while letting

		providing a valid license.	legal users use the program.
R2	Subscription period restriction	The client and monolith program shall be protected such that its business value cannot be utilized outside the subscription period of a valid license.	The licenses are sold as subscriptions, for the protection to be meaningful, we consider it necessary to stop usage of the programs if the license period has expired, or theoretically not yet started.
R3	Installation count restriction	The client and monolith program shall be protected such that its business value cannot be utilized by more persons than the installed license allows.	In the current license model, the software solutions are sold as per user. To protect the software form illegal copying and redistribution, restricting the allowed number of users per license is required.
R4	Product version dependency	A valid license shall only be valid for a configurable number of software products identifiers.	To allow configurability for future products and product versions a license shall only be valid for a specified set of product IDs. This allows Company A to stop already generated license codes to be valid for future versions of the software.
R5	Program files independency	The protection shall not depend on anything unique about specific program files of the same program and program version.	With current distribution methods, several customers will be delivered will the exact same executable and program files. If the distribution were done exclusively by a web service, it could alter the files individually for each end-user for stronger protection. But this is not possible.
R6	Stop license generators	Given the distributed application files it shall not be possible to extract enough information to craft new valid licenses that can unlock for the original programs.	The point here is to stop an adversary from being able to create a key-generator or equivalent without first modify the actual program files, which is a more sophisticated attack.
R7	Secure treatment of license data	Any license information stored and transmitted by the solution programs must be protected. That is, a third party shall not be able access any sensitive license data or codes; confidentiality and integrity shall be provided.	The license data can be sensitive. Product keys or files that that unlock the software are valuable and shall be handled securely and delivered securely.

3.4.2 Usability Requirements

Here are the requirements describing the software protection's usability impact on the client and monolith application. As discussed in Chapter 3, building a protection without interfering with usability is contradictive; these requirements are therefore important in order to regulate and protect the usability from the end-users' perspective.

ID	**Name**	**Description**	**Motivation**
R8	Internet access prerequisites	The protection shall not depend on continues online connection. Nor should it require internet connection more than once during license installation.	Though the vast majority of end-users do have continuous internet connection, it is not always the case. Therefore, to use the programs, an internet connection should not be required. However, during the installation procedure an online connection is acceptable. The software is after all delivered electronically over internet in almost all cases.
R9	Notifications and warning dialogs	Close to the actual expiration date the applications shall display a warning dialog at every startup till the user chooses to not show it more. After the	The meaning is to clearly inform the user about the current license's status. Giving adequate information in time to contact their supplier so that no issues arise.

		expiration date a new window shall appear, telling that the license has expired. The dialogs shall provide information of days left of license etc.	
R10	Business value after license expiration	After the license expired, full functionality shall be provided by the applications in two extra months. This shall only apply if a previously valid license was installed. During these two months, warning dialog shall appear at startup, informing that the license has expired and that the program soon locks important functionality.	For many customers, the application is business critical. As an extra measure to avoid that the program suddenly restricts itself at an unfortunate timing, it shall be possible to use the program two months unlicensed if the previous license was valid.
R11	Displayable license information	End-users shall at any time, if a license has successfully been installed, be able to view their license information from the monolith and client application. This would be days left to expiration, activation date, organization name, contact person at the company and the specific licensee name.	For convenience, end-users shall be able to easily see their license information at any time.
R12	Clear license installation process	The license installation process shall be clear and simple, meaning that it shall consist of a few simple steps where the user inputs required information. Information that should be explained clearly. What is happening at every step with the provided data shall also be clear. Finally shall contact information for support to Company A be supplied clearly.	To minimize the inconvenience associated with chosen copy-protection scheme, the installation procedure should be designed as simple and clear as possible.

3.4.3 Miscellaneous Requirements

Here are a few but important functional requirements that neither relate to security or usability.

ID	**Name**	**Description**	**Motivation**
R13	Easily distributed licenses	Company A shall easily be able to deliver licenses to end-users by email or by any means the current software is delivered.	Given the distribution methods used by Company A, it is required to easily distribute the codes or files that unlocks the software together with the actual product without any significant overhead.
R14	Optional copy-protection	Without further work, Company A shall easily be able to disable the entire software protection to for the protected programs.	It is our belief that there is no need for a protection in low risk markets or for trusted customers. To avoid unnecessary inconvenience, Company A will continue to deliver an unprotected version o to trusted markets and customers.
R15	No overhead on further development	The software protection shall not imply any further restrictions or overhead when developing the software product. Ideally, the protection solution shall be its own separate module of the program that does not interfere more than necessary with the rest of the code base.	Due to Company A's rather small size, it is desired to create protection that is easily maintained. Even though many of the most powerful protection schemes rely on continuous updates and changes, these are not preferred if not entirely automated.

3.4.4 License Manager Requirements

The table below summarizes the last eight requirements targeting the license manager tool. Both functional and non-functional requirements are mixed in the table.

ID	Name	Description	Motivation
R16	License entities and their relations	The license manager shall model Company A's license entities according to the domain model described in chapter 3.2. That is, the application shall be responsible for maintaining sold subscription bundles, individual licenses, products, customer organizations, end-users and contact persons in the specified way.	The main purpose of the license manager is to maintain the licenses information used by Company A. Thus must it model the entities and relations corresponding to the Company A's domain model.
R17	Associated data of license entities	Each license entity shall be associated with its relevant license data. The application shall maintain the following information related to a subscription bundle: o The subscription period for subscription licenses. o The number of subscriptions allowed for license bundle o Whether a license bundle has been billed or not o Whether a license bundle it has been paid or not. o A reminder date o A text comment And the following information related to products. o The product name o A identification number o The following information related to customer organizations; o The organization name And the following information related to contact persons; o Persons name o Address o Phone number o Email o A text comment	Apart from entities and how they are related, the license data associated with these must be part of the system for it to be useful.
R18	Create, modify and delete license entities	Though the GUI, it shall be possible to create, modify and delete the different license entities described in R16. Their relations shall also be editable according to the domain model. Data associated with entities (described in R17) shall also be editable.	For the license manager to be useful, the license data must be created and editable though an easy to use GUI.

R19	List and filter license entities	It should be easy to list the different license entities and view their license data and relationships with other entities. These lists shall feature filtering to against important license data.	The amount of license data to manage is already large and will likely continue to grow over time. Thus it is important to be able to display the license data in grids featuring filtering to easily distinguish the data of interest from the rest.
R20	Print and export license data	The different GUI views containing lists of filterable license entities shall be printable and exportable to Microsoft Excel.	Needs to save the license information or send selections of it outside the license manager exist. For example when on meetings or emailed to coworkers.
R21	Email settings and automated mailings	The application shall be able to automatically send emails to the administrator of the license manager. It shall exist two emails types. A reminder email when a date has passed a specified reminder date and another when the license actually expired. These shall be sent to specific email list which can be edited. The content of these mails should also be configurable.	To ease the work of maintaining the licenses and licenses, it is desired by Company A to be able to set up automatic mailings. Reminder emails should be sent administrators so that they are remained and inform contact persons that need to be informed.
R22	Password protection	The application shall be password protected.	The license manager will contain sensitive data, thus must it be password protected.
R23	Secure connections	Any internet communication done by the license manager shall be over a secure connection. SSL support is required.	The license manager will need to communicate with a centralized service to get access to the license database; this connection must be secure for the same obvious reason as with R22.

4 Solution Design

In this chapter we put the theory into practice, describing the complete licensing solution developed for Company A's goals and needs. The two alternative solution designs considered along the selected one are also described briefly in chapter 4.4. All descriptions are on a rather conceptual level without minding architectural or implementation specific details, subjects discussed in chapter 5. Mainly the solution's protection scheme is described with a focus on security. Furthermore, the different software components of the system are introduced to give a complete high level picture over the solution design.

4.2 Solution Overview

The solution relies exclusively on technical software protection techniques in order to meet the security requirements from chapter 3.4.1. Particularly it uses a version of the online activation scheme described in chapter 2.3.2 to protect against illegal use and copying. No particular strategy has been applied to deal with illegal modification or debugging attempts where the adversary uses an assembler debugger to analyze and finally alter the protected executables. The solution thus relies on encryption only to achieve security, which is a sufficient to protect the unmodified software but naturally lacking protection against more sophisticated attacks. The reasoning behind the choice of omitting other protection strategies is discussed in chapter 6.3.

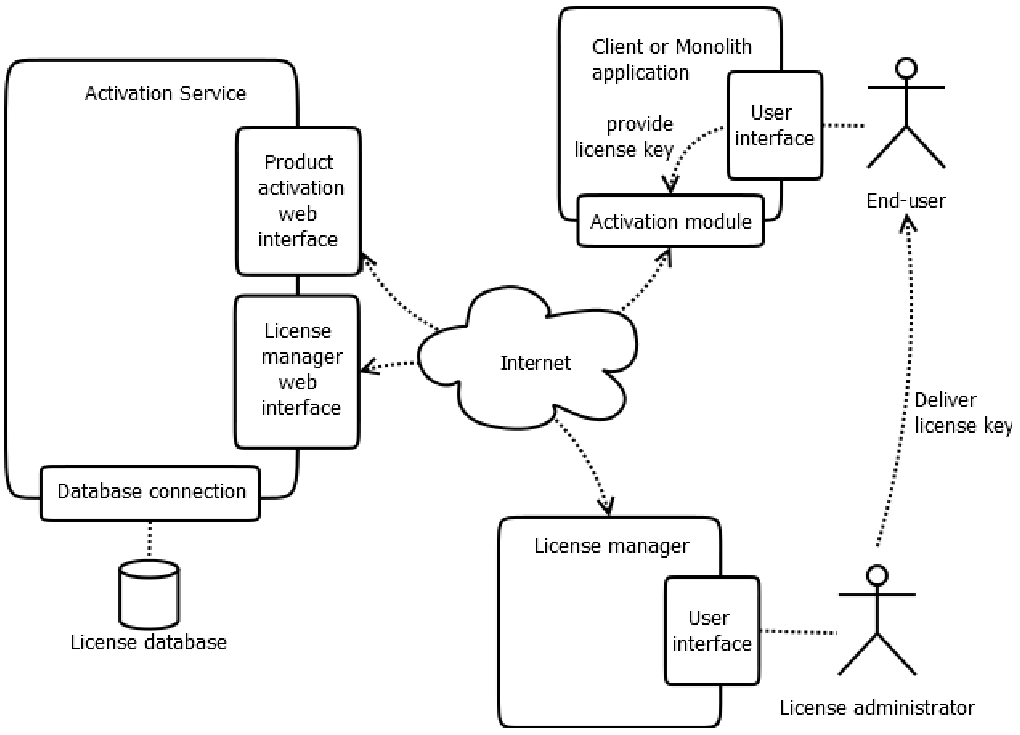


Figure 3. Conceptual overview of the online activation scheme.

The entire system consists of three distinct components, a web service, a local activation module and a license manager, each described in its own section below. Figure 3 above illustrates the conceptual overview of this three components and how they interact with different actors.

4.2.1 The activation service

The activation service's task is to manage and store most of the license data used by the licensing solution, meaning it controls the various license entities and their mutual relations according to the licensing model described in chapter 3.2. Naturally, all the data is saved in a database, see figure 3 above. In order to access the license entities, either in administration purposes or during license installation, the activation service hosts two protected web interfaces. The two other components, the license manager and the activation module connect to one web interface each in order to fetch or modify license data according to their corresponding tasks.

The product activation web interface specifies the communication with the protected software applications during the license installation procedure. Basically, it specifies the communication protocol used when the client or monolith applications activates a license key to unlock itself. Through this interface, the activation service verifies if an activation request is valid and creates a signed certificate for the calling application. These certificates are later used by the calling application to verify if it is legitimate offline. The protection scheme is explained in greater detail in chapter 4.3.3.

The other web interface, the license manager web interface in figure 3, specifies the communication protocol through which administrative tasks are carried out. The license manager developed for Company A's administrative needs connects to it in order to get, create, delete or modify license data.

4.2.2 The activation module

The activation module refers to a set of software components in the software with the distinct purpose to handle license related tasks. This module hinders an unauthorized user (or rather, a not activated client or monolith installation) to be utilized or redistributed. The module is responsible for several things, most importantly the following;

- Guiding the end-user though the license installation process, involving connecting to the activation interface and activate the software. This would be equivalent with meeting R12.
- Validating if previously license installation attempts are still valid or not, this is the backbone of the copy-protection.
- Inform and guide the end-user with appropriate messages in case problems associated with an installed license occur. This could be when expiration dates are about to expire or the installed license is not valid for any reason. See requirement R9 and R12 regulating this.
- Letting the end-user acquire license information associated with a successful activation as necessary to meet requirement R11.

Exactly how tasks such as saving activation results, validating license keys etc. are performed is covered in chapter 4.3.

4.2.3 THE LICENSE MANAGER

The license manager is the desktop application with which Company A's license administrators will manage organizations, licentiates, license keys and other license entities stored and maintained by the activation service. This tool will primarily be used to configure the activation service such that customers can activate their product when delivered. Secondly it will aid Company A in their tasks to manage license entities, meeting requirement R9.

4.3 THE PROTECTION SCHEME

At application startup, the license module tries to find an installed license. The stored license is in this case a custom generated license certificate containing a set of variables, including a digital signature created by the activation service. These certificates should not be confused with X.509 certificates or other specified standards; they are custom formatted and specifically designed for this licensing system, see chapter 4.3.1 to 4.3.3 for how they are generated and verified.

If a license certificate is found, it is validated locally by the license modules' validation procedure. Should no certificate be found, a new license activation process is initialized. If the activation procedure succeeds, the newly installed certificate is validated as if it were present from the beginning. A successful certificate validation results in a normal application start while a failed validation results in an appropriate error message followed by a new license installation procedure. Figure 4 is a flowchart describing this startup process.

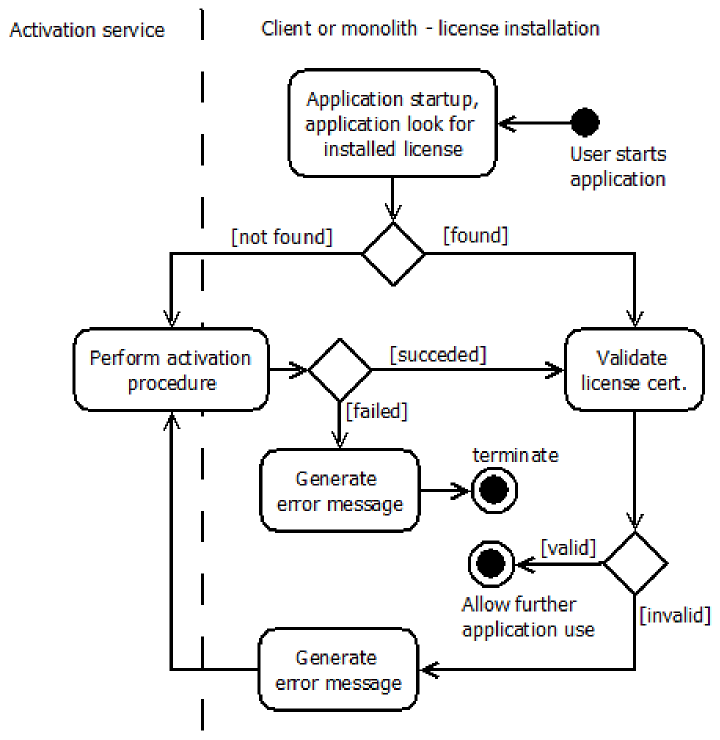
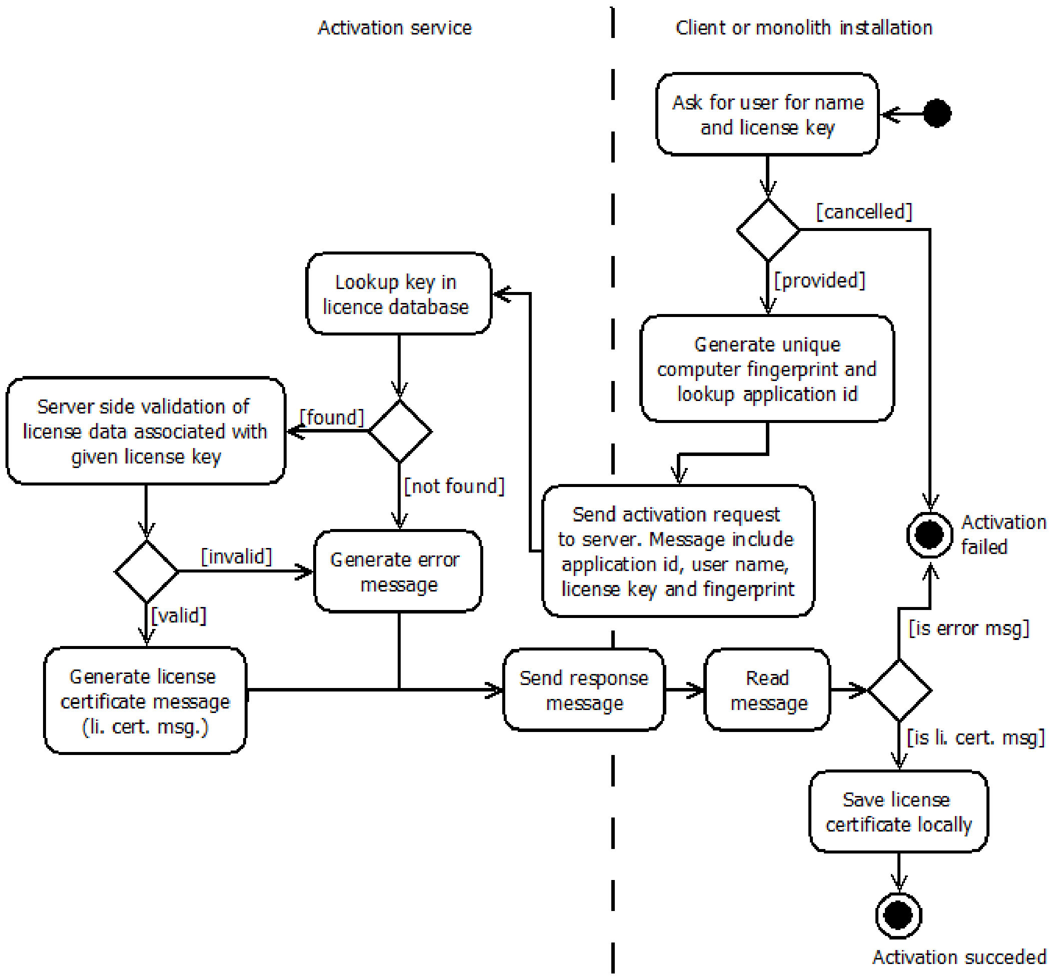


Figure 4. The initialization of the activation procedure in the protection scheme.

4.3.1 License installation procedure

The activation procedure starts by asking the end-user for a license key and user name. The license key is simply a short text string formatted as recommended in chapter 2.3.4 using a limited alphabet and hyphens. The user name is an optional addition allowing Company A's administrators to find and edit activations and licenses more easily should it be required in a future support inquiry.

If a correctly formatted license key is provided, the license installation continues by sending an activation request to the activation service. The activation service evaluates the request and sends back the result; if successful the result contains a license certificate otherwise an error message. The client or monolith application then reads the response result and either saves the certificate for future evaluations or displays an appropriate error message. This activation procedure is illustrated by the flowchart in figure 5.



Activation service
Client or monolith installation
Ask for user for name and license key
[cancelled]
[provided]
Lookup key in licence database
Generate unique computer fingerprint and lookup application id
Server side validation of license data associated with given license key
[found]
[not found]
Send activation request to server. Message include application id, user name, license key and fingerprint
Activation failed
[invalid]
Generate error message
[valid]
[is error msg]
Generate license certificate message (li. cert. msg.)
Send response message
Read message
[is li. cert. msg]
Save license certificate locally
Activation succeded

Figure 5. Flowchart describing the activation procedure carried out by scheme.

To make the reasoning about the different activities in figure 7 more clear, let us now define a few necessary variables. We use $*$ to denote that a variable is originating from the client side in order to distinguish corresponding variables from server and client side. For example, if k is the license key generated by the activation service (when Company A's license administrators create a license); then k^* is the corresponding license key provided by an end-user as input during the license installation procedure.

- k and k^* are the license keys stored on the server and given the end-user, respectively.
- u^*_{name} is the user name provided by the end user.
- a_{id_k} and a^*_{id} are the application identification numbers. The first saved on the server associated with license key k and the second embedded in the client or monolith application.
- f_{id} and f^*_{id} are the unique fingerprints of the end-user's computer. f_{id} stored on the server side and f^*_{id} generated by end-user's machine.

Going back to the installation procedure, after k^* and u^*_{name} been given as input, the application id a^*_{id} is looked up by the license module. a^*_{id} is a defined constant in the protected software later used to validate that k^* is associated to the right application version. In order to feature node looking, a unique fingerprint f^*_{id} of the running system configuration is generated using the proposed methods described in chapter 2.3.6. Next the license module sends an activation request to the activation service over an encrypted communication with variables k^*, u^*_{name}, a^*_{id} and f^*_{id} as parameters. k^*, a^*_{id} and f^*_{id} must be defined while u^*_{name} can be empty, as it is optional.

When receiving the activation request, the activation service searches the license database for the given key k^*, if a key k in the database matching given k^* is found the associated license information of k is gathered and calculated. The variables used by the server side verification associated with key k are[1];

- k obviusly, equal to input value of k^* since the database entry was found.
- d_{ek} the expiration date of the license with key k.
- d_{now} the current date on the activation service server.
- a_{id_k} associated with k.
- n_k the number of previous successful activation of license with key k.
- $n_{k_{max}}$ the number of allowed activations of license with key k.

If the value of k^* is found in the database a server side validation process is performed, and if it succeeds, the activation request is successful and the activation service shall generate a license certificate as result to the calling client application. Otherwise, the activation has failed and an error message is generated and returned instead. Successful validations are also saved in the database to keep a record over used activations and calculate future n_k. The server side validation procedure involves checking the values of the following variables;

[1]Other variables used by the license manager exists for maintenance purposes, but this chapter considers mainly those used to meet the security requirements from chapter 3.4.2.

- $a_{id}^{*} = a_{id_k}$
- $d_{now} < d_{ek}$
- $n_k < n_{k_{max}}$

If the returned result is a certificate, the client or monolith application saves it and continues as shown in figure 6, otherwise an error message is displayed as previously stated and the application terminates.

4.3.2 LICENSE CERTIFICATE STRUCTURE

Using the notations introduced in chapter 3.4.2, a license certificate consists of the following variables;

- Creation date d_{ck} of the certificate itself. Displayable in the protected software to meet requirement R11. It is also checked in the local certificate validation procedure in an effort to detect if an adversary is manipulating the system time the program fetch from Windows, see chapter 4.3.4 for more details.
- Expiration date d_{ek} of the license associated with license key k. Obviously used by both server side validation and local certificate validation.
- Application id a_{id_k} of the application. Also used by both server side validation and local certificate validation.
- The fingerprint identifier f_{id_k} given as parameter in the activation request. Used in the local validation procedure to achieve node locking to the end-user's specific computer.
- Reminder date $d_{rk}$of the license associated with the key k. The date is used by the protected applicants to know when to display information dialogs about expiration dates and similar. Primarily in the certificate to meet requirement R9 and is not part in any validation procedure.
- The optional user name u_{name} given as parameter in the activation request, empty if it was not given. Only displayable as additional information about the license to meet requirement 11.
- The organization name o_{name} of the organization associated with the license used for the activation. Only displayable as additional information about the license to meet requirement 11.
- The license key k of the license associated with the product activation. Only displayable as additional information about the license to meet requirement 11. The key is part of the server side verification procedure as described in previous chapter, but is not used in the certificate validation.
- A digital signature s of the above variables. Used to meet requirement 6 and protect against illegal copying. This signature is the backbone of the protection scheme.

To create the certificate, the license service combines all the variables, except s, into a comma separated list vs_{csv}. The digital signature s is then created by signing vs_{csv}, using RSASSA-PKCS1-v1-5 (RSA Laboratories, 2012) as digital signature algorithm. Actually, more modern digital signatures such RSASSA-PSS or ECDSA (RSA Laboratories, 2012) (Johnson, et al., 2001) would be more desirable to use, but the chosen algorithm does provide enough security and has the best support from the development environment used by Company A. The final certificate c is then composed by vs_{csv} and s, $c = (vs_{csv},\ s)$.

4.3.3 LOCAL CERTIFICATE VALIDATION

The local certificate validation procedure verifies the variables and the digital signature in a license certificates. The validation is carried out according to the practices described in chapter 2.3.3. In detail, the operation is performed in the following steps given the saved certificate c as input;

1. Extract s and vs_{csv} from license certificate c. From the variables in vs_{cvs} extract d_{ck}, d_{ek}, a_{id_k} and f_{id}. See variable definitions in chapter 3.4.3. If step is successful, then continue, otherwise return an error.
2. Generate fingerprint f_{id}^* from the running system configuration.
3. Lookup constant a_{id}^* defined in the application.
4. Verify signature s using RSASSA-PKCS1-V1.5 and public key component defined as a constant in the protected application. If successful, continue, otherwise return error and exit.
5. Verify $a_{id_k} = a_{id}^*$. If true, continue, otherwise return error.
6. Verify that f_{id} matches f_{id}^*. If true, continue, otherwise return error.
7. Load current date d_{now}^*, see next chapter for how this is done, and compare it with d_{ek} according to;
    i. If current date has passed the expiration date with more than 2 months, that is if $(d_{now}^* > d_{ek})\ \wedge\ (\ |d_{ek} - d_{now}^*| > 62)$ exit with error, else continue.
    ii. If expiration date has passed current date with less than 2 months, that is if $(d_{now}^* > d_{ek})\ \wedge (|d_{ek} - d_{now}^*| < 62)$. Then show message informing that license has expired to the end-user but exit the procedure successfully to meet requirement 10.
    iii. Should neither a. nor b. hold, but $(d_{now}^* > d_{rk})$ is true, then show message informing user that license is about to expire in $|d_{ek} - d_{now}^*|$ days and exit successfully to meet requirement 9.
    iv. If none of i., ii. or iii. happened, then the certificate must be valid and the procedure should quietly exit successfully.

If all 1-7 steps above succeed, we have a valid license and the application continues as normal, se figure 4 how this is handed by the application. In contrast, should any step fail, the procedure stops with an error message, as described at every step in the list. This error message is then displayed to the end-user and the execution continues as described by figure 4.

4.3.4 Obtaining reliable dates

Since the validation is carried out locally on the end-user's system, the date d_{now}^{*} must be fetched from the end-user's system. The end-user's system is controlled by the end-user and he could try to modify the system clock into an incorrect value in an attempt to fool the copy protection scheme. To strengthen this weakness, the protection scheme addresses the issue as following.

Part of a license certificate is the date variable d_{ck}, the creation date of the certificate. During the license installation process d_{ck} is saved encrypted on the end-user's system, we call this saved date d_p. d_p is then the last date known to be correct. When d_{now}^{*} is fetched from the underlying system it is compared against d_p. If d_{now}^{*} is earlier than d_p we certainly know that d_{now}^{*} is incorrect. Otherwise it could be correct and we override d_{now}^{*}'s value to d_p.

If d_{now}^{*} is found to be incorrect or missing from the system, the protected software requires a safe method to determine d_{now}^{*} and asks the end-user for permission to contact a trusted server to get the correct date. If this call succeeds the now trusted d_{now}^{*} overrides the incorrect d_p, and the program continues as usual. If this process fails or is denied by the end-user, the certificate validation returns an error and the application denies further usage.

4.4 Alternative solution designs

Since part of the process was to investigate alternative designs, two other competing solution concepts were developed together with the selected one described above. Though not selected, these are described rather briefly in this chapter.

4.4.1 Time limited local installation scheme

The time limited local installation scheme is a variation of the basic local installation scheme described in chapter 2.3.1, but with a time limit on the distributed license files. Like in the selected design, the license manager tool is a separate desktop application used to maintain license entities and generate, in this case, license files that unlock the protected software. However, instead of being a client application connected to a centralized activation service, the entire application runs locally against a local SQL database.

When license files are used to unlock the software, the content of the file is validated in a normal license installation procedure based on the same cryptographic scheme as described in 2.3.3. The main difference is that an additional date variable is validated, a date which controls whether or not the license file is valid or not.

To handle this, the monolith and client application are extended with a module that parses, installs and validates license files. The validation of license files is performed by the applications locally at the end-user's computer without connection to any centralized activation authority; figure 6 below outlines the scheme's components, actors and their relations.

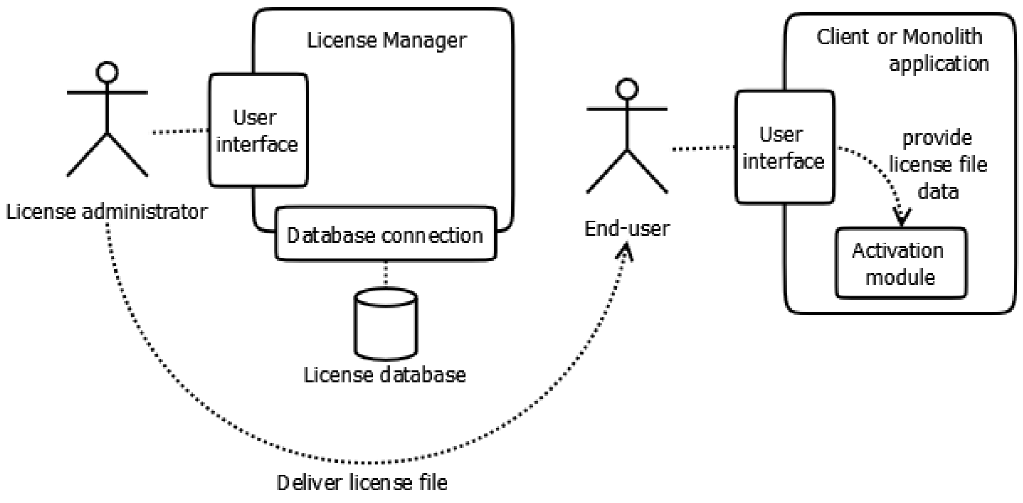


Figure 6. Conceptual overview of proposed local installation scheme.

The license installation procedure is carried out by the user and the activation module. At application startup, the activation module is responsible for checking if a license is already installed or not. If it is found, the license data associated with the saved license file is validated, otherwise the user is asked to provide a valid license file. If validation succeeds the application stores the license data securely using AES-256 or any other secure block cipher to encrypt the license data. If a saved license was not found and the end-user provides a license file, this file's license data is validated similarly, otherwise the application terminates with an appropriate error message. The process is described by the flowchart below, figure 7.

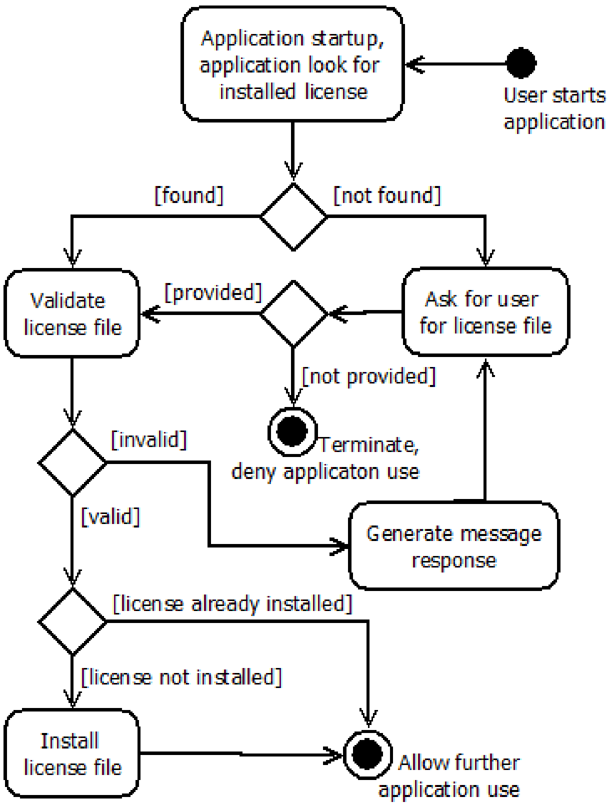

Application startup, application look for installed license
User starts application
[found]
[not found]
Validate license file
[provided]
Ask for user for license file
[not provided]
[invalid]
Terminate, deny applicaton use
[valid]
Generate message response
[license already installed]
[license not installed]
Install license file
Allow further application use

Figure 7. Flowchart over the license installation procedure.

Validation of license files is done by testing four variable values stored in the encrypted license file. These are;

- d_e the expiration date of the license.
- d_f the expiration date of the actual license file.
- a_{id} identification number of the application the license file was generated for.
- s, a digital signature created out of the three other variables.

When a license file is validated the first time, before a stored license file is present, the validation is carried out as follows;

1. Verifies signature s using the public key part embedded in the protected software.
2. Check if a_{id} match the protected application's embedded id number.
3. Date d_l is earlier than current date.
4. Date d_f is earlier than current date.

In order to make the application work after the initial license installation, future validations using the previously stored license files skip step four above. This feature gives the license files a specified time period during which they are usable.

4.4.2 Server Controlled Scheme

This server controlled network scheme originates from the network and online schemes introduced in chapter 2.4. Since the client version of the software, see chapter 3 for more details, is based on a client-server setup featuring user accounts with usernames, logins etc., it is possible to build a controlling module in the server software. A module dedicated to control client connections, i.e. application users, according to a license model. Such a scheme could outline as described in this chapter.

First off, a local installation scheme similar to the one presented in the last chapter would be used to activate the server software instead of the client applications. The primarily goal is however not to protect the server software from illegal usage or copying, but to provide the new module in the server, responsible for controlling clients connections, with necessary variables and license data. Compared to the local license installation scheme described in 4.4.1, the expiration date of the license file would therefore be dropped; other aspects remain the same however. The server software is then extended with a new module responsible for handling logged in users, a module that uses the provided license data to determine if client connections are valid or not.

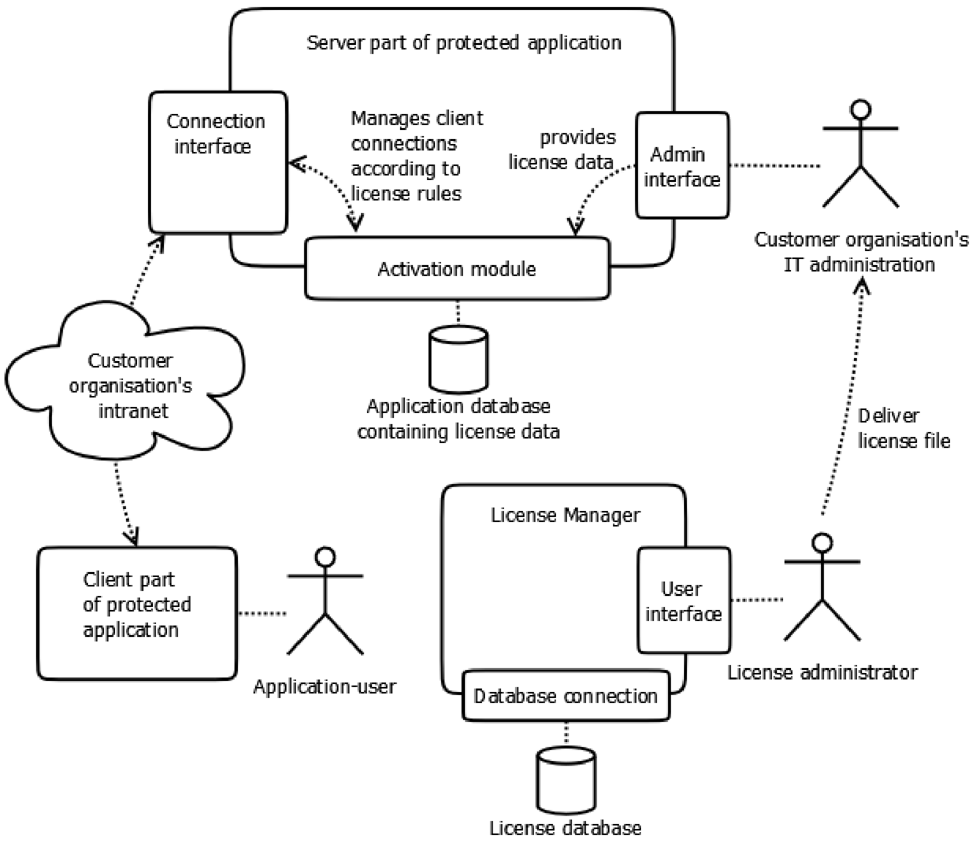

Server part of protected application
Connection interface
Manages client connections according to license rules
provides license data
Admin interface
Customer organisation's IT administration
Activation module
Customer organisation's intranet
Application database containing license data
Deliver license file
License Manager
Client part of protected application
User interface
Application-user
License administrator
Database connection
License database

Figure 8. Overview of components in server controlled protection scheme.

The scheme would still contain a separate license manager used by Company A’s administrators to control license data and generate license files, in this case targeting the installation of the server part of the protected software. All in all, the components involved and their connections are illustrated in figure 8 above.

Kept on a conceptual level, the license file should contain the necessary license data; expiration dates, application identifiers, reminder dates and the number of allowed installations. This data is during license installation, on the server, saved into the application’s already existing database. Since the server normally runs on the customer’s site, the data must be saved encrypted using a secret key preferably embedded in the server software itself. In all essential regards, the rest of the license installation procedure will outline as in the local protection scheme from last chapter but targeting the server instead of the client software.

Since the installation is carried out towards the server part of the protected software, it is the IT administrator that will do the installation. Thus is the license files distributed to the IT department or equal of the customer organizations rather than directly to end-users.

After a successful license installation and while the license file is valid in regard to expiration dates, the server would keep track over logged in accounts and save them along the register license data in encrypted database entries. The new copy protection module would use this data to verify users at every login to assure no more users than allowed are ever registered.

5 IMPLEMENTATION AND SYSTEM DESIGN

While the last chapter described how the solution works on a rather conceptual level, this section is dedicated to the software design and implementation details. The major software component's software architecture is described here, considering specific implementation details such as programming languages, third party libraries and used technologies.

5.1 SYSTEM OVERVIEW

Figure 9 below illustrates the software components introduced in chapter 4.2 in a component diagram. One can see that *ActivationService*, *ActivationModule* and *LicenseManager* are connected via the two interfaces *IActivationService* and *ILicenseAdminService*. Unsurprisingly, these correspond to the two web interface from figure 3. Furthermore, a third program is introduced, *InvokeMailService*. It is a small windows service exclusively created to initialize mailings to customers and stakeholders at Company A (Microsoft, 2014). How the larger components are designed and implemented is described in the following sub-chapters.

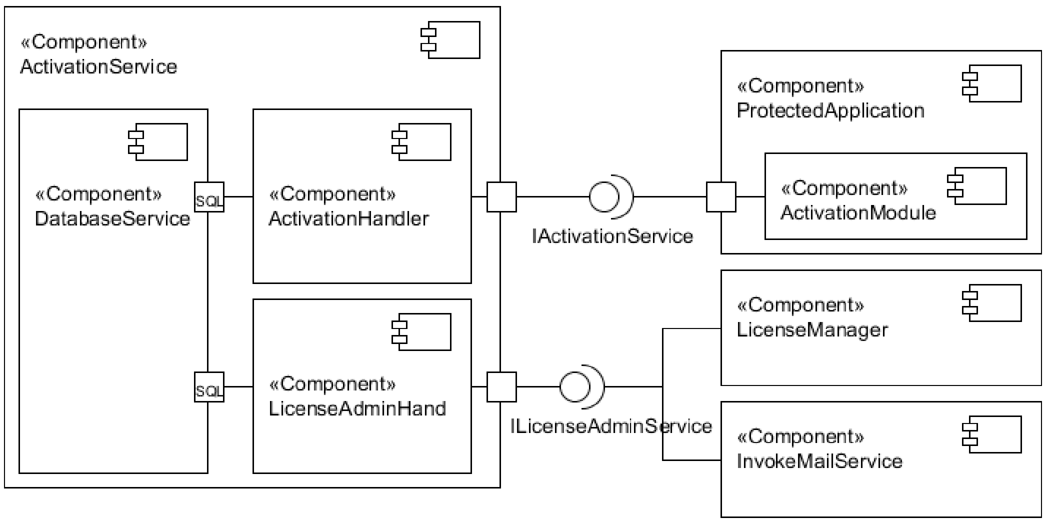


Figure 9. A component diagram over the entire system.

5.2 THE ACTIVATION SERVICE AS A WCF SERVICE

The activation service follows a service-oriented architecture (SOA) in order to provide an accessible and usable web interface which the license manager, the protected applications and possibly future applications can connect to. Above all else, the SOA design offers a clear decoupling between client and services components by using standardized technologies. The main advantage is that client applications can be developed using different languages but still consume the web service without considering implementation specific details.

The service is implemented in .NET 4.5 and is consequently written in the C#. More importantly, it is based on the WCF framework (Windows Communication Foundation) which is Microsoft current technology for creating service-oriented systems and applications. All WCF services feature so called *endpoints* though which all communication occurs. Loosely speaking, a WCF endpoint consists of an address on which it is found, a binding specifying how communication is performed and a service contract containing a set of operations supported by the service. WCF bindings are highly configurable regarding which underlying technologies is to be used to communicate with the service; for example HTTPS as transport protocol. The endpoints are themselves specified in Web Service Description Language (WSDL); refer to the specification for further details (W3C, 2014). A service contract is in this case a set of methods, i.e. services, which the consuming client applications can call; they assemble the functionality provided by the service.

In our solution, *endpoints* were configured to use Simple Object Access Protocol (SOAP) specified messages over HTTPS since it had good support from Delphi, the language the protected applications are developed in, see chapter 3.1. The endpoint binding can however be configured to use JSON and other technologies easily. For more information about WCF, please refer to Microsoft’s documentation (Microsoft, 2014).

To summarize, *ActivationService* from figure 9 has two endpoints *IActivatonService* and *ILicenseAdminService* at two different addresses. How to connect to them is described in WSDL and they feature one service contract each, *ILicenseAdminContact* and *IActivationContract*. Each contract has a set of methods used to manipulate or gather license data; appendix 1 lists these operations in detail. Mainly, *IActivationContract* specify the operations used for activating products while *ILicenseAdminContact* specify the necessary operations used to meet Company A’s license administration needs. Finally, the messages transmitted over the network are XML based and specified according using the SOAP standard, transferred over the HTTPS.

5.3 Activation Service Software Architecture

Internally, the *ActivationService* consists of three subcomponents, the *ActivationHandler*, *LicenseAdminHandler* and *DatabaseService*. These components are by themselves rather small, consisting of only one or a few classes but with very distinct tasks. *LicenseAdminHandler* and *ActivationHandler* are responsible for providing all necessary functionality in order to support the implementation of *IActivationContract* and *ILicenseAdminContact*.

The communication over *ILicenseAdminContact* applies a Data Transfer Object pattern, from now on referred to as DTO pattern (Microsoft, 2014). This means that most data carried over the network are represented by fields in transfer objects; figure 10 shows the different DTO classes used.

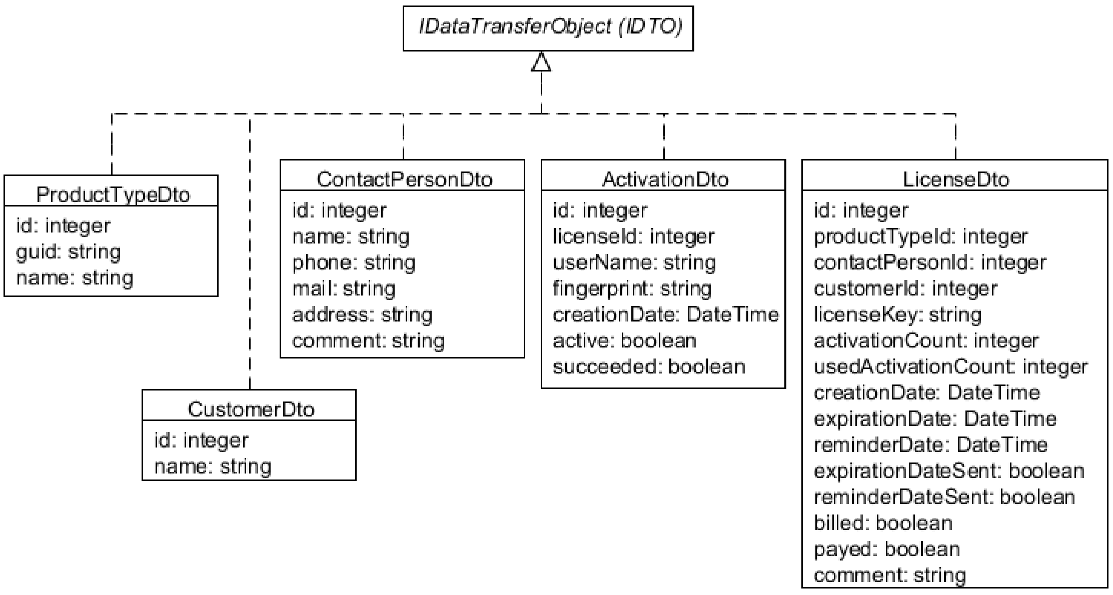


Figure 10. The different Data Transfer Objects which carry data.

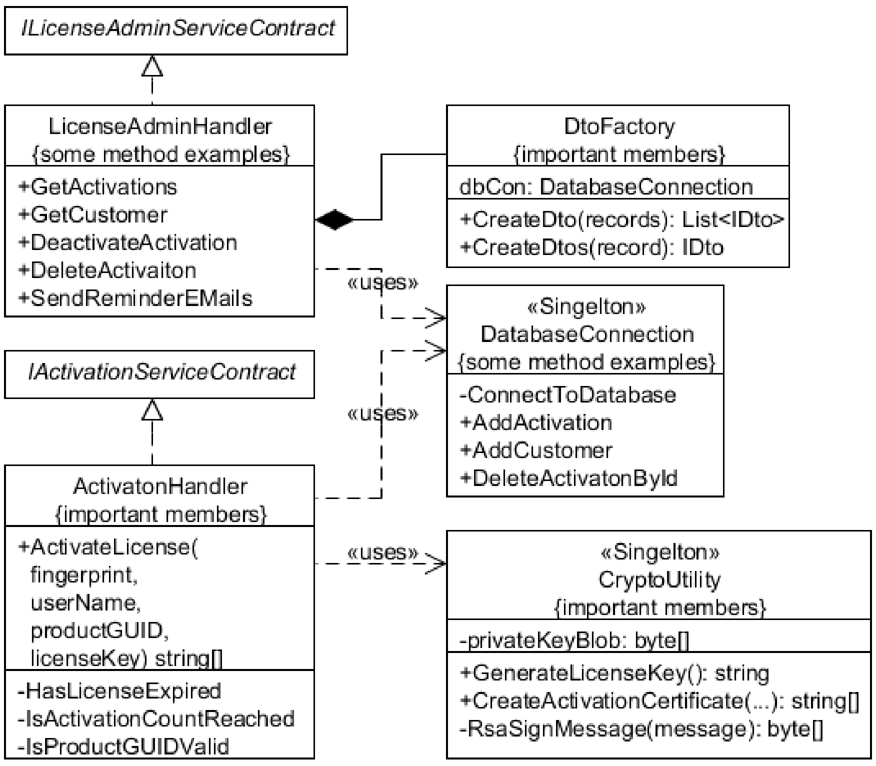


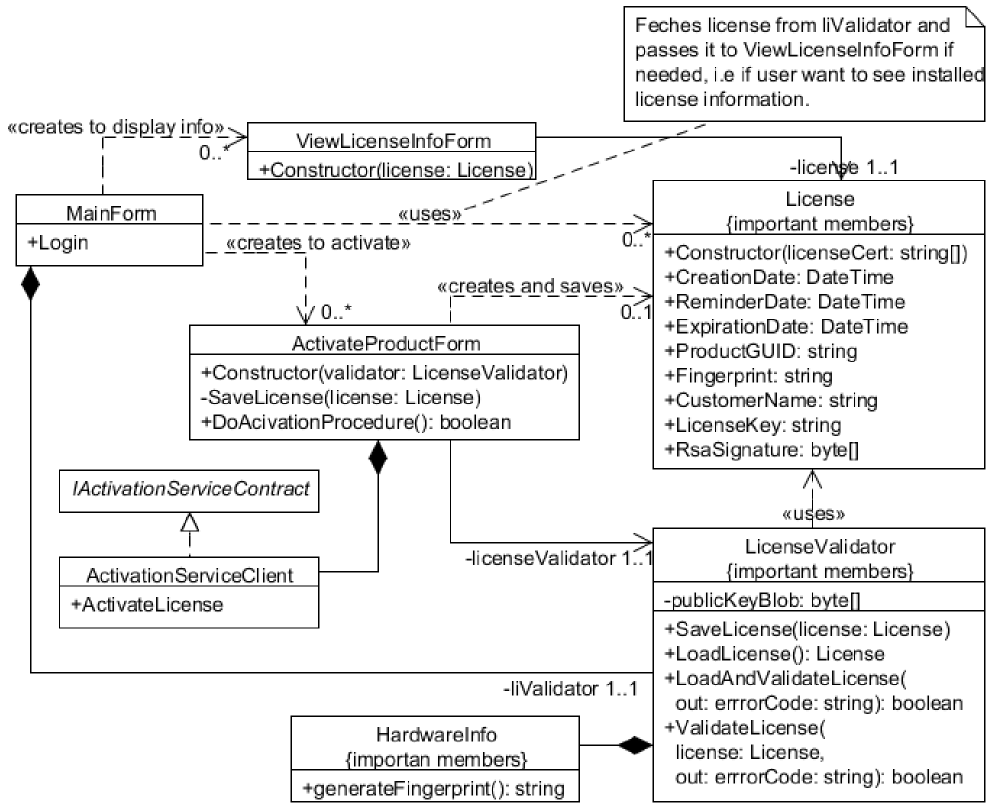
Figure 11. A simplified class diagram of the activation services' key classes.

Essentially, there is one DTO type for each license entity in the domain model, see chapter 3.2 where the domain model is described. It should be stressed that the other endpoint, *IActivationServcie*, does not use DTOs at all, only simple data types. This is mainly due to environment incompatibilities between used technologies. However, the fact that *IActivationContract* only specifies few methods makes the advantages of using DTOs negligible.

Leaving the transfer objects, the class diagram in figure 11 describes the most crucial classes. Component *ActivationHandler* from figure 10 consist of the class *ActivationHandler* and *CryptoUtility*. *CryptoUtility* features all necessary cryptographic related tasks; mainly the creation of license keys and activation certificates as described in chapter 4.3.1. Component *LicenseAdminHandler* contains the class *LicenseAdminHandler* and *DtoFactory*. *DtoFactory* construct the five DTO types in figure 10 by taking primitive data returned from the *DatabaseConnection* which, as suggested by its name, follows the factory pattern. The *DatabaseService* component is the *DatabaseConnection* class with belonging SQL server. Mainly, *DatabaseConnection* provides an abstracted interface to access the database data. Both *DatabaseConnection* and *CryptoUtility*, particular *DatabaseConnection*, must not exist as multiple instances at the same time and implement the Singleton pattern to address this issue.

5.4 Activation Module Software Architecture

As discussed in chapter 3.1, the protected monolith and client applications are built using Delphi. The activation module component *ActivationModule* is therefore also implemented in Delphi. Figure 12 below summarize the activation module's important classes and members.



Feches license from liValidator and passes it to ViewLicenseInfoForm if needed, i.e if user want to see installed license information.
«creates to display info»
0..*
ViewLicenseInfoForm
+Constructor(license: License)
-license 1..1
MainForm
+Login
«uses»
0..*
«creates to activate»
License
{important members}
+Constructor(licenseCert: string[])
+CreationDate: DateTime
+ReminderDate: DateTime
+ExpirationDate: DateTime
+ProductGUID: string
+Fingerprint: string
+CustomerName: string
+LicenseKey: string
+RsaSignature: byte[]
«creates and saves»
0..1
0..*
ActivateProductForm
+Constructor(validator: LicenseValidator)
-SaveLicense(license: License)
+DoAcivationProcedure(): boolean
IActivationServiceContract
«uses»
ActivationServiceClient
+ActivateLicense
-licenseValidator 1..1
LicenseValidator
{important members}
-publicKeyBlob: byte[]
+SaveLicense(license: License)
+LoadLicense(): License
+LoadAndValidateLicense(out: errrorCode: string): boolean
+ValidateLicense(license: License, out: errrorCode: string): boolean
-liValidator 1..1
HardwareInfo
{importan members}
+generateFingerprint(): string

Figure 12. Class diagram of the activation module.

The license module has two GUI components implemented as forms, *ViewLicenseInfoForm* and *ActivateProductForm*. Please refer to the documentation for more information about Delphi libraries' what a form is (Embarcadero, 2014). Their two forms' names summarize their functionality rather well, *ViewLicenseInfoForm* shows license information about any installed license on the end-users system while *ActivateProductForm* guides the user though the license installation procedure. *MainForm* is basically the protected applications main controller, which is of interest since it initializes the license checks as described in figure 4 from chapter 4.3. The other classes are *License*, *LicenseValidator*, *HardwareInfo* and *ActivationServiceClient.*

License is basically a container class of all relevant license data used by the activation module, it is the software component holding the license certificate described in chapter 4.3.1 for example. *LicenseValidator* is responsible for loading any saved license certificates, i.e. any *License*, and evaluate if it is valid or not. In the case of a failed validation, *LicenseValidator* should output the corresponding error code such that the caller can show appropriate information to the end-user.

All cryptographic operations carried out by the activation module are indeed done by *LicenseValidator*, most importantly is the validation of RSA signatures. These cryptographic operations rely on CryptoAPI which is part of Windows and compatible with .NET's *System.Security.Cryptography* package (Microsoft, 2014) used by the activation service. Finally, *HardwareInfo* is a helper class responsible for the generation of unique fingerprints and is owned by the *LicenseValidator*.

ActivaiotnServiceClient implements client side version of the service contract, *IActivationClientContract*, and is responsible for all network communication with the activation service. These two classes are generated by the Delphi IDE which has the necessary tools and features to easily consume a WSDL services and auto create the necessary communication code.

5.6 LICENSE MANAGER SOFTWARE ARCHITECTURE

Since the license manger is completely developed from the beginning, there were no real constrictions in which platform to use. By recommendation of Company A's developers, C# with .NET was chosen in combination with Windows Forms as GUI technology (Microsoft, 2014).

With these technology choices as basis, the license manager design was divided into two layers, a GUI layer and model layer. Since Windows Forms were used, complete separation between GUI and controller components was not possible like in for example the MVC or MVVC based designs. Classes that would correspond to the model where however clearly separated from the combined GUI and controller, meaning that the application's business logic is independent from the GUI.

The class diagram in figure 13 below summarizes the most important classes in the model package while figure 14 describes the various GUI classes with controller functionality, mostly windows forms. Every DTO type from figure 10 has a domain object (or business object if you like) with additional operations and temporary variables. Together they model the application's domain logic according to the license model from chapter 3.2.

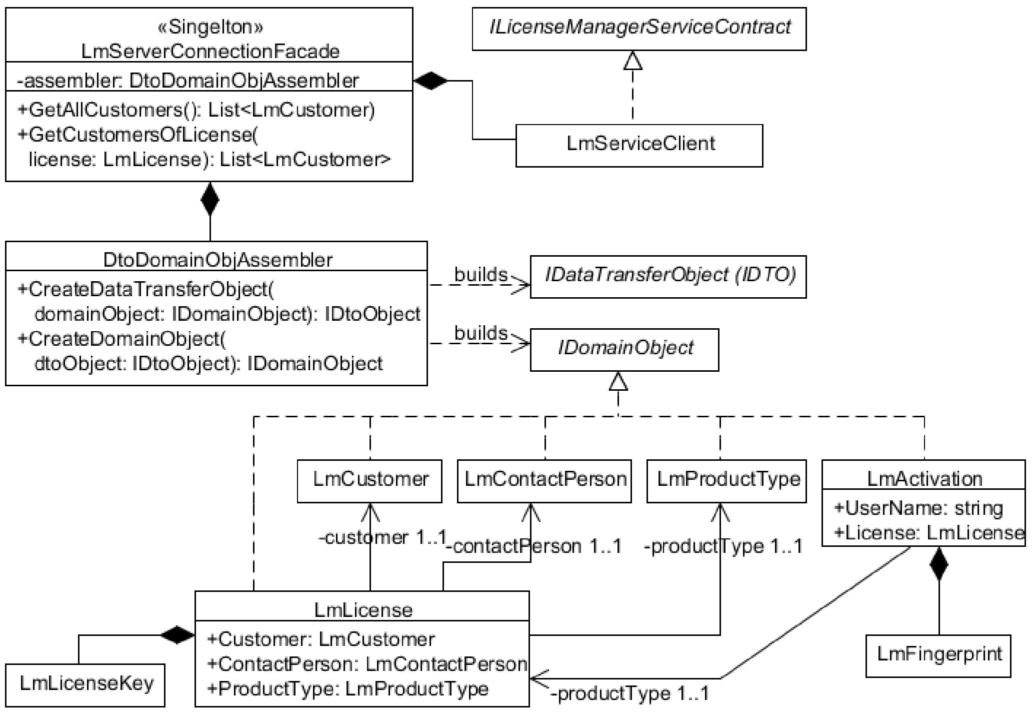


Figure 13. The most important classes in the model package of the license manager.

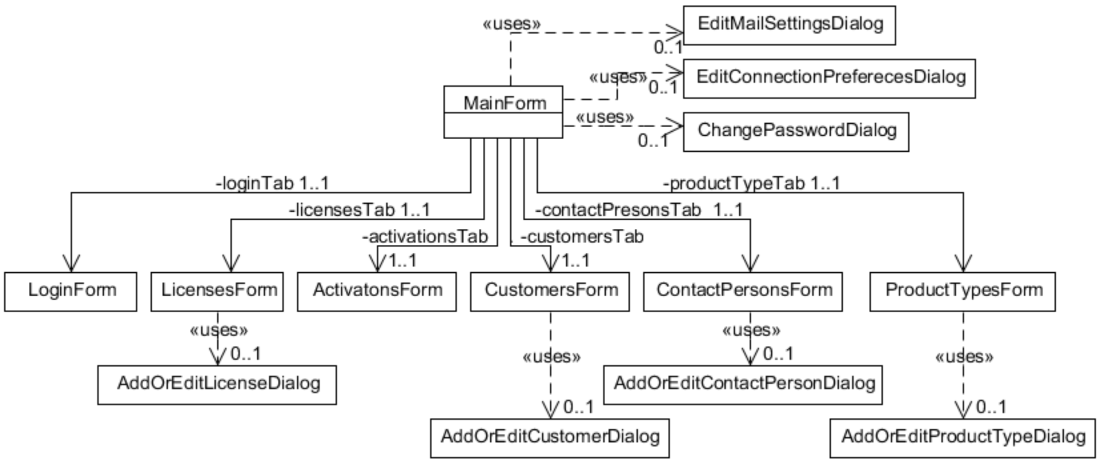


Figure 14. The controlling GUI classes of the license manager.

DtoDomainObjAssembler implements the assembler pattern originally proposed by Fowler (Fowler, 2003) and recommended by Microsoft in usage with the DTO pattern (Microsoft, 2014). Basically it features an abstraction between the DTOs and domain objects such that they are completely independent from each other. The *LmServerConnectionFacade* class implements the facade and singleton pattern while giving an interface towards network code. Its purpose is to provide a bridge between the model package and controlling GUI objects through which they can fetch domain objects and simpler data. To accomplish this, the facade owns an instance of *LmServiceClient* which implements *ILicenseManagerContract* to feature the client functionality to consume, i.e. connect to, the activation service’s web interface. The data, usually DTOs, gathered from this object are passed to *DtoDomainObjAssembler* to create the domain objects used by the higher level classes. In a similar way, data transfer objects are created by *DtoDomainObjAssembler* from domain object when preparing request to the web service.

The GUI package consists of a couple of form classes fulfilling different purposes. The *MainForm* is the main window in which the entire application is hosted. Mainly, it controls six tabs in which six other major forms reside; these are the *LoginForm*, *LicenceForm*, *ActivationForm*, *CustomersForm*, *ContactPersonsForm* and *ProductTypesForm*. They can be seen as six different pages; each with the functionality to view, delete, modify and create a specific domain object, except from the *LoginForm* which obviously handles user authentication. Each of these six forms uses the *LmServerConnectionFacade* to get the needed domain objects from the activation service.

In addition to these pages forms, a set of dialog forms implemented as modular windows exists. These are used to gather input from the end-user for the purpose of modifying settings or domain object attributes.

6 DISCUSSION

This chapter starts by comparing the selected solution with the two alternative designs form a security and end-user perspective, motivating the choice of protection scheme. It continues by discussing the selected solution design's security properties in depth, motivating the choice to omitting protection against illegal modification and reverse engineering. Finally, we end with a brief discussion of potential improvements and future paths of development.

6.1 SECURITY REQUIREMENTS EVALUATION

This section is dedicated to compare the three proposed solution design introduced in chapter 4 against the security requirements in chapter 3.4.1. This is done by analyzing how each protection scheme meets each security requirements, respectively.

The selected solution design, described in chapters 4.2 and 4.3, certainly meets requirement R1, R2, R3 and R4 very well. As described, the scheme relies on two validation procedures, one server side and one client side. During the server side validation, license keys are accepted or denied. To be accepted, the number of previous activations using the same key is controlled, thus is R3 meet. R1, R2 and R4 are rather obviously met during the local validation procedure of the server side generated license certificates, see chapter 4.3.3 for the exact validation procedure.

Requirement R6 translates to stopping an adversary from being able to create his own certificates. Since the application requires a valid RSA signature signed by the activation service's private key, this is not possible unless the RSA scheme is compromised. Mainly because getting hold of the private RSA key with only the public key component available is considered infeasible if RSA is used with appropriate parameters, such as good key length, padding scheme etc. Thus is R6 is fulfilled.

R5 is also met since no unique or special information, such as encryption keys, are required for individual end-users or customer organizations. Only the activation module, with its RSA public key component, application id and fingerprint generation algorithm is required. Importantly, these are the same for any delivery of the same program version.

All network communication between the activation service and protected application can easily use SSL or TSL, guaranteeing confidentiality and integrity for the license certificate transitions over Internet. The license certificates are also saved encrypted, should anyone unauthorized get hold of it. All in all, R7 is therefore met.

6.1.1 TIME LIMITED LOCAL INSTALLATION SCHEME

In this scenario, license files are generated in advanced and delivered with the distributed programs files, see chapter 4.4.1 for details. If the local license file validation fails at startup, the application terminates, thus is business value protected and R1 met. Similarly do expiration date checks against current date satisfy R2 while application id checks satisfy requirement R4. The verification of the digital signature in the license file makes it impossible to create new valid license files given any information the protected applications have access to, thus guaranteeing R6. Since license file content is encrypted using AES or another block cipher for storage, R7 is secured.

The scheme is not without problems however. First off, there is nothing stopping anyone from illegally installing several copies using the same license file during the allowed time period, therefore somewhat failing to meet R3. The only protection gained is a time limiting restriction over how long time period an adversary can redistribute the license without even altering the distributed executables.

Furthermore, as with most local installation schemes, finding reliable dates to compare expiration dates against, in in our case also license file expiration date, is a problem. One cannot entirely use the scheme described in chapter 4.3.4 since it requires a reliable first start date from a trustworthy third party. In the online activation scheme, this date is acquired from the activation service.

6.1.2 Server Controlled Scheme Evaluation

Using this scheme, the protected software's server code would enjoy the same protection as the client side code would from the local installation scenario, but without the time limitation on license files. Due only to the applied local installation scheme on the server, R1, R2 and R4 are met for the server side code, and thus the client applications since they are useless without the server. Any number of servers can however be installed, meaning that the client applications are not protected from illegal usage or copying indirectly by the server side application's when connecting to different servers.

End-user's accounts will however be validated on login against the server. The license files installed are saved encrypted in the server's database and include subscription periods and allowed user count data. Since the server obviously saves records of existing user accounts in the database, R3 can be secured using simple verification checks at login.

None the less, the scheme suffers the same potential weakness regarding fetching secure dates from underlying operating system as the local installation scheme. Much more significant are however the security issues with the monolith version. Since the monolith version lacks a server, it gain no more security than a plain local installation as described in chapter 2.3.1, thus failing to meet requirement R3.

Requirements R5-R7 are however easily met since license information is stored encrypted in the application's database. The same information also contains a digital signature from the license file.

6.2 Usability Requirements Evaluation

This section discusses the usability features of the three proposed solution designs from chapter 4 against the requirements in chapter 3.4.2. This is done by analyzing how each protection scheme meets respective requirement.

Regarding internet connection requirements, the online activation scheme requires the end-user to be connected while completing the license installation procedure the first time, i.e. when the activation procedure is carried out as described in chapter 4.3.1. R8 is therefore fulfilled, but with bigger usability impact compared to the other two proposed schemes.

Furthermore, the user must remember the delivered license key to be able to activate any more licenses in the future. Though configurable, it is intended that every customer organization is given one key allowing as many activations as paid user licenses. Thus is it not that much information to keep track of. Furthermore, since the key is only a lookup number used by the activation service, it is rather short and can be saved written or digitally. The license manager also keeps track of distributed keys making it easy to help a customer who lost it.

Requirements R9, R10, R11 and R12 are easily implemented since the necessary information is saved in saved license certificate. Regarding the extra two months usage offered after the real license expiration, it is also just a matter of tweaking the date validation checks such that the requirement is fulfilled, as described in chapter 4.3.

6.2.1 Time Limited Local Installation Scheme

The main advantage of the local installation scheme is that end-users do not need an internet connection except when downloading the license file. This does however not add any inconvenience since the programs files already are delivered digitally, thus is R8 met without problems at all. R9, R10, R11 and R12 are again, as with the selected activation scheme, just a straight forward matter of implementing the desired features since the information to display is present in the saved license file.

However, being limited to install the software during a certain timeframe must be considered a rather harsh constraint. It is not very satisfying from the copyright holders perspective either since every reinstallation of the software within the subscription period would likely require new license files, thus creating a support inquiry between Company A and the customer organization for each such event. This does contradict one of the purposes of the license solution, to reduce maintenance. Furthermore; high security in a local installation based scheme requires license files instead of shorter more flexible text strings which reduce usability further.

6.2.2 Server Controlled Scheme

Regarding requirement 8, internet access is not required with this scheme, only network access to the server software which is a requirement anyway. For installing the server service, one must use license files in the same manner as with the time limited local installation scheme. However, compared to the local installation scheme, this is done much less frequently and by the administrators at the customer organization. Furthermore, the license files are not time limited.

The other usability requirements are as with the two other proposed solutions easily met, it is only a matter of implementing the desired functionality since all data to display is included in a license file. All in all, this scheme has the least impact on the end-user compared to the two other proposed.

6.3 Threats and Types of Adversaries

In this section we identify the types of persons likely to attempt an attack on the protected software products. The identified attackers, or adversaries, are then analyzed to determine what threats they impose. Finally, we argue that the chosen protection scheme indeed meet the required level of security.

Looking back on the problem formulation and motivation for developing the copy-protection, we can identify two types of adversaries. The first is the typical end-user, experience with using computers but without any expertise on how to analyze and modify the program executables. It is believed that this is the most common person to illegally copy Company A’s software. The typical scenario would be a mid or small sized organization that buys a couple of licenses and the following years only buys one license while still using several. Without any protection, Company A is unable to stop this misuse.

The second type of adversary is a more capable person hired specifically to crack the program. As of today, there are no signs that such adversaries exist, but with a potentially success in China this threat could grow significantly (Buisiness Software Allicance, BSA, 2012) (Yang, et al., 2013). This kind of adversary would have the necessary knowledge and experience to use an assembler debugger to analyze the machine code and alter it in a way that removes its protection mechanism.

For the average user how lacks the means of modifying the program files, the chosen protection scheme is secure. As discussed in chapter 2.3.3 and 7.1.1 the digital signature in combination with the uniquely generated fingerprint protects the application from illegal copying and usage. The programs are therefore fully protected against the most common attack. Against the professional pirate, experienced with assembler programming and reverse engineering tools, the protection is however rather weak. The methods used to stop him from succeeding are obfuscation, temper-proofing, anti-debugging or even self-modifying protection strategies. As discussed, these strategies make it much harder to analyze and modify the executables, but in the end they would only delay a skilled attacker. Inevitably he would be able to deobfuscate the program and remove the protection given enough resources and time. These techniques are however omitted in the implemented prototype, making him even more likely to succeed.

The problem is however not as worrying as one might think. The important point is that some larger customers use Company A’s software solutions together with their own subcontractors. As stated in the introduction, it is mainly the subcontractors’ misuse that has caused issues. The sold software is also an information system, as described in chapter 3.1, and as such rather useless on its own. It is required that the customer organization use it together with Company A’s proposed way of working with project management. Because of this, the threat of a professional pirate is rather small. He would certainly be too expensive to hire for the typical small or midsized business just to avoid a few license fees. Most importantly however, a business trying to copy the software with the goal of reselling it would not only need the cracked software, but also acquire knowledge and expertise about Company A’s processes and models. With this background, we believe that the skilled hacker does not impose a very big threat.

Furthermore, as suggested by Nill et al, it is not advisable to invest in a too complex protection strategy unless motivating financial losses are identified. If the risk of infringement is overall low, but the strategic importance of sold products high, it is instead better to apply a light protection scheme while observing the development closely (Nill, et al., 2009). This recommendation in combination with Company A's limited size and resource motivates why an investment in expensive strategies is not advisable. At least as long no financial losses can be derived from the professional type of attacker. It is instead a more sensible solution to develop a solution aiming to stop the general user who indeed has been identified to bypass the license agreement. Among the proposed solutions, the selected design was identified to meet these recommendations best while still leaving the door open for future improvements.

6.4 Project Goals Feedback

In this section we look back to the questions we set out to answer from chapter 1.4 and compare the results presented in previous chapters against these objectives. We show where these questions have been answered in the report, concluding that the project goals have indeed been achieved.

Q1. What copy-protection schemes and strategies are known and used as of today? And how do these work?

As stated in the method chapter, a literature review was conducted in an effort to answer this question. The results from this literature review are presented in chapter 2, the Theoretical Background, which can be seen as the proposed answer to Q1. For even more reading and in depth studies, refer to the referenced documentation, articles and journal papers from chapter 2.

Q2. How do the license model used by Company A look today? And how may it change in the future?

The license model used by Company A is clearly presented in chapter 3.2 and 3.3. Furthermore, the protected software programs are described in all necessary detail in chapter 3.1. As stated in these chapters, the distribution methods or licensing agreements has stayed the same for a long period and the marketing responsible at Company A do not believe in any changes over the foreseeable future.

Q3. What are the desired security properties of the software protection?

The desired security properties of the copy-protection features are summarized by the security requirements presented in chapter 3.4.1. How the developed prototype solution meets these requirements is in turn evaluated in the chapter 6.1.1 arguing that they are indeed met.

Q4. What are the desired end-user requirements regarding usability and availability?

The desired end-user requirements, i.e. requirements guarding the end-users interests, are presented in chapter 3.4.2. Likewise is the evaluation concluding that they are met discussed in chapter 6.1.2.

Q5. What high-level functional, non-functional and other requirements targeting the license management tool exist?

The desired functionality of the license manager tool is discussed and summarized by the requirements Q16 to Q23 in chapter 3.4.3.

Q6. How shall the license management tool and software protection be designed?

Q6 is answered in chapter 4 where the solution designs are described, in particular the implemented protection scheme's design. In chapter 5 we describe the software architecture further for both the license manager and the other system components. Finally, chapter 6.2 and 6.3 motivates why the described protection scheme and solution design were selected.

6.5 Feedback from Company A

Overall Company A' is satisfied with the results presented in this report, different stakeholders has expressed that the project meets their initial expectations. While some desired features are still missing, they do see a direct use of the system at its current state if some further testing and deployment issues were resolved. Specifically, personnel with regular contact with customers have identified several cases where they believe the current protection would give value, forcing users to buy licenses they actually use.

Furthermore, the report serves as a good knowledge base future development initiatives considering copy-protection features can consult. This is mainly because the report collects possible improvements and copy-protection techniques into one place, tailored for Company A's needs.

Company A also sees opportunities to use the software solution for other purposes than administrating and protecting sold licenses. Being a web-service, the activation service can for example provide live data about Company A's customer base for marketing purposes. One example could be plotting current customers on a map for Company A's website. The activation service could also be extended with features improving product distribution as described in next chapter.

Finally it should be said that the solution, begin a prototype, is not ready for use directly. However, it is likely that efforts will be made to fully implement and deploy a protection scheme based on the prototype within the coming year. Something one must see as a success given the project's scope.

6.6 Future work

This last section discusses aspects which could be improved together with some ideas of possible improvements. First off, requirement R20 regarding exporting license information from the license manager to MS Excel, or more precisely to the .XLSX-file format were never met. The feature was omitted in order to concentrate on more important aspects.

Furthermore, to actually deploy the solution, testing need further attention. The developed prototype solution is only a prototype in the sense that much required testing has not been completed. To guarantee a good service, the solution should undergo everything from manual system and acceptance testing down to rigorous unit testing. Currently, no sufficient unit test suite exists. Some aspects are however particularly hard to test and need extra care, for example the fingerprint generation algorithm which need testing on a wide range of computer systems due to its hardware dependency.

As a general improvement to existing distribution methods, one could develop a website and extend the activation service with the protected program files such that they can be downloaded. This would not only be convenient improvement and addition to the current distribution methods, it would also open up doors for a range of protection schemes relying on customizing the sent program files individually to the downloading client, possible increase security and improve the service experience simultaneously.

Another idea is to extend the activation service with the functionality to let end-users deactivate or unregister activations. This might be convenient in order to, for example, reinstall the software on another computer without contacting Company A. Currently, this is not considered a big problem since almost all customers contact Company A to get hold of the program files when reinstalling anyway. However, in some scenarios it might be appreciable with an unregister feature. As of now, the functionality does exist in the activation service, but is disabled. It should however be noted that the unregister feature is accompanied by some security concerns; it is possible to save the license certificate manually, unregister the installed license and then reinstall the saved license certificate manually if the user knows where it is saved.

Finally, regarding stronger protection features, it is possible to improve the protection scheme with various obfuscation and anti-debugging techniques as discussed in chapter 6.3. However, it should be stressed once again that given Company A’s limited size and resources, expensive and complex setups should generally be avoided in favor for developing and marketing the actual software product.

BIBLIOGRAPHY

Anckaert, B., Sutter, B. D. & Bosschere, K. D., 2004. *Software piracy prevention through diversity.* New York, Proceedings of the 4th ACM workshop on Digital rights management .

Boneh, D., Lynn, B. & Shacham, H., 2004. Short Signatures from the Weil Pairing. *Journal of cryptology,* 17(4), pp. 297-319.

Buisiness Software Allicance, BSA, 2012. *SHADOW MARKET - 2011 BSA Global Software Privacy Study, Ninth Edition,* s.l.: Buiness Software Alliance.

Choudhary, V., 2007. Comparison of Software Quality. *Journal of Managment Information Systems,* 2(141-165), p. 24.

Collberg, C. S. & Thomborson, C., 2002. Watermarking, Temper-Proofing, and obfuscation - Tools for Software Protection. *Transactions on Software Engineering,* 28(8), pp. 735-746.

Conner, K. R. & Rumelt, R. P., 1991. Software Piracy: An Analysis of Protection Strategies. *Management Science,* Volume 37, pp. 125-139.

Djekic, P. & Loebbecke, C., 2007. Preventing application software piracy: An empirical investigation of technical copy protections. *The Journal of Strategic Information Systems,* 16(2), pp. 173-186.

Edlira Martiria, A. B., 2012. Monotone digital signatures: an application in software copy. *Procedia Technology,* Volume 1, p. 275–279.

Embarcadero, 2014. *Delphi XE5 | Develop Apps for Android, iOS, Windows and OS X.* [Online]
Available at: http://www.embarcadero.com/products/delphi
[Accessed 18 02 2014].

Embarcadero, 2014. *Vcl.Forms-TForm - XE2 API Documentation.* [Online]
Available at: http://docwiki.embarcadero.com/Libraries/XE2/en/Vcl.Forms.TForm
[Accessed 10 02 02].

Ferrie, P., 2011. *The "Ultimate" Anti-Debugging Reference.* s.l.:s.n.

FIPS, 2001. *ADVANCED ENCRYPTION STANDARD (AES) (FIPS 197),* s.l.: Federal Information Processing Standards Publications (FIPS PUBS).

FIPS, 2013. *Digital Signature Standard (DSS),* Gaithersburg: FEDERAL INFORMATION PROCESSING STANDARDS PUBLICATION (FIPS PUB).

Fowler, M., 2003. *Patterns of enterprise application architecture.* Boston: s.n.

Free Software Fundation, 2013. *Categories of free and non free software.* [Online]
Available at: http://www.gnu.org/philosophy/free-sw.html
[Accessed 19 June 2013].

Hachez, G., 2003. *A Comparative Study of Software Protection Tools Suited for E-Commerce with Contributions to Software Watermarking and Smart Cards,* s.l.: s.n.

IDC Analye the future, 2012. *2012 Key Trends in Software Pricing & Licensing Servey, Seventh Edition,* s.l.: IDC Analye the future.

Jan M. Memon, A. K. A. B. A. S., 2007. *A Study of Software Protection Techniques.* Hyderabad, Pakistan, Springer Netherlands.

Jiutao, T. & Guoyuan, L., 2010. *Research of Software Protection.* Xuzhou, China, International Conference on Educational and Network Technology (ICENT 2010).

Johnson, D., Menezes, A. & Vanstone, S., 2001. The Elliptic Curve Digital Signature Algorithm (ECDSA). *International Journal of Information Security,* 1(1), pp. 36-63.

Kisserli, N. & Preneel, B., 2011. A taxonomy of self-modifying code for obfuscation. *Computers & Security,* 30(8), pp. 679-691.

Kügler, R., 2009. *Scalable Software Protection and Flexible Licensing.* Karlsruhe, Germany: WIBU-SYSTEMS AG.

Licenturion, 2001. *Inside Windows Product Activation - A Fully Licensed Paper.* [Online]
Available at: http://www.licenturion.com/xp/fully-licensed-wpa.txt
[Accessed 12 July 2013].

LimeLM, 2013. *What is hardware-locking licensing and why use LimeLm.* [Online]
Available at: http://wyday.com/limelm/features/why/
[Accessed 2 08 2013].

Microsoft, 2014. *Data Transfer Object.* [Online]
Available at: http://msdn.microsoft.com/en-us/library/ff649585.aspx
[Accessed 10 02 2014].

Microsoft, 2014. *Services (Windows).* [Online]
Available at: http://msdn.microsoft.com/en-us/library/windows/desktop/ms685141%28v=vs.85%29.aspx
[Accessed 10 02 2014].

Microsoft, 2014. *System.Security.Cryptography Namespace ().* [Online]
Available at: http://msdn.microsoft.com/en-us/library/system.security.cryptography%28v=vs.110%29.aspx
[Accessed 10 02 2014].

Microsoft, 2014. *What Is Windows Communication Foundation.* [Online]
Available at: http://msdn.microsoft.com/en-us/library/ms731082%28v=vs.110%29.aspx
[Accessed 10 02 2014].

Microsoft, 2014. *Windows Forms.* [Online]
Available at: http://msdn.microsoft.com/en-us/library/dd30h2yb%28v=vs.110%29.aspx
[Accessed 10 02 2014].

Nill, A., J, C. & II, S., 2009. Global software piracy: Trends and strategic considerations. *Business Horizons,* 52(3), pp. 289-298.

Peyravian, M., Roginsky, A. & Zunic, N., 2003. Methods for preventing unauthorized software distribution. *Computers & Security,* 22(4), pp. 316-321.

Png, I., 2010. *On the Reliability of Software Piracy Statistics.* s.l., 43rd Hawaii International Conference on System Science (HICSS).

Rawlinson, D. R. & Lupton, R. A., 2007. Cross-National Attitudes and Perceptions Concerning Software Piracy: A Comparative Study of Students from the United States and China. *Journal of Education for Business,* 37(2), pp. 87-93.

RSA Laboratories, 2012. *PKCS #1 v2.2: RSA Cryptography Standard,* s.l.: EMC Corporation.

SafeNet, 2010. *Software Protection and Licensing Guide.* s.l.:SafeNet.

Sanaei, M. G., Zamani, H., Abarghouei, B. E. & Hakimi, A. G., 2013. Online Modules: Novel Model in Serial-Based Method of Software Copy Protection. *International Journal of Computer Science and Telecommunications,* 4(3), pp. 1-9.

Sharma, V. K., Rizvi, S. A. M. & Hussain, S. Z., 2010. Distributed Software and License Key Management "An Initiative to Stop Software Piracy". *Ubiquitous Computing Security Systems ,* 5(4), pp. 1771-1777.

Shields, T., 2008. *Anti-Debugging - A Developers View,* s.l.: Veracode.

Tan, G., Chen, Y. & Jakubowski, M. H., 2007. Delayed and Controlled Failures in Tamper-Resistant Software - Lecture Notes in Computer Science. In: J. L. Camenisch, C. S. Collberg, N. F. Johnson & P. Sallee, eds. *Information Hiding.* s.l.:Springer Berlin Heidelberg, pp. 216-231.

W3C, 2014. *Web Service Description Language (WSDL) Version 2.0 Part 0: Primer.* [Online]
Available at: http://www.w3.org/TR/wsdl20-primer/
[Accessed 10 02 2014].

Wroblewski, G., 2002. *General Method of Program Code Obfuscation,* Wroclaw: s.n.

Yang, Z., Zhou, J., Qin, H. & Koong, K. S., 2013. Quantitative analysis of global software piracy: 2003 through 2010. *Business and System Research,* 7(1), pp. 81-100.

Yuschuk, O., 2014. *OllyDbg.* [Online]
Available at: http://www.ollydbg.de/
[Accessed 7 2 2014].

APPENDIX 1 – SERVICE CONTACTS

This chapter lists the methods specified by the two service endpoints featured by the activation service. The two contracts, or protocols as they can be seen, are the *ILicenseManagerContract* and the *IActivaionServiceContract* introduced in chapter 5.1.

ILicenseManagerContract

```
bool AddLicense(string password, LicenseDto license);

bool UpdateLicense(string password, LicenseDto license);

bool DeleteLicense(string password, LicenseDto license);

List<LicenseDto> GetLicenses(string password);

bool AddProductType(string password, ProductTypeDto productType);

bool UpdateProductType(string password, ProductTypeDto productType);

bool DeleteProductType(string password, ProductTypeDto productType);

List<ProductTypeDto> GetProductTypes(string password);

bool DeleteActivation(string password, ActivationDto activation);

bool UpdateActivation(string password, ActivationDto activation);

List<ActivationDto> GetActivationsByLicenseId(string password, int licenseId);

List<ActivationDto> GetActivations(string password);

bool AddCustomer(string password, CustomerDto customer);

bool UpdateCustomer(string password, CustomerDto customer);

bool DeleteCustomer(string password, CustomerDto customer);

List<CustomerDto> GetCustomers(string password);

bool AddContactPerson(string password, ContactPersonDto contactPerson);

bool UpdateContactPerson(string password, ContactPersonDto contactPerson);

bool DeleteContactPerson(string password, ContactPersonDto contactPerson);

List<ContactPersonDto> GetContactPersons(string password);
```

```
bool AuthenticateAdmin(string password);

bool UpdateAdminPassword(string password, string newPassword);

string GetAdminMail(string password);

bool UpdateAdminMail(string password, string newAdminMail);

string GetServiceEmail(string password);

bool UpdateServiceEmail(string password, string newServiceEmail);

string GetServiceUser(string password);

bool UpdateServiceUser(string password, string newServiceUser);

string GetServiceEmailPassword(string password);

bool UpdateServiceEmailPassword(string password, string newServiceEmailPassword);

string GetReminderMailSubject(string password);

bool UpdateReminderMailSubject(string password, string newReminderMailSubject);

string GetReminderMailBody(string password);

bool UpdateReminderMailBody(string password, string newReminderMailBody);

string GetExpirationMailSubject(string password);

bool UpdateExpirationMailSubject(string password, string newExpirationMailSubject);

string GetExpirationMailBody(string password);

bool UpdateExpirationMailBody(string password, string newExpirationMailBody);

string GetMailHostAddress(string password);

bool UpdateMailHostAddress(string password, string newHostAddress);

string GetMailHostPort(string password);

bool UpdateMailHostPort(string password, string newHostPort);

void SendReminderEmails();

void SendExpirationEmails();
```

IActivaionServiceContract

string[] RegisterLicense(string fingerprint, string userName, string productTypeName, string licenseKey);

string[] UnregisterLicense(string fingerprint, string licenseKey);