

*Master Thesis*  
*Software Engineering*  
*Thesis no: MSE-2005-10*  
*August 2005*



# **Securing Customer Data in a Distributed System**

**Johan Gjertz**

School of Engineering  
Blekinge Institute of Technology  
Box 520  
SE – 372 25 Ronneby  
Sweden

This thesis is submitted to the School of Engineering at Blekinge Institute of Technology in partial fulfillment of the requirements for the degree of Master of Science in Software Engineering. The thesis is equivalent to 20 weeks of full time studies.

**Contact Information:**

Author:

Johan Gjertz

E-mail: [johan.gjertz@gmail.com](mailto:johan.gjertz@gmail.com)

External advisors:

Jan Gerle & Daniel Starke

Buchner & Partner GmbH

Address: Lise-Meitner-Str 1-7, 24223 Ralsdorf, Germany

Phone: + 49 (0)4307 8119 60

University advisor:

Håkan Grahn

Department of Systems and Software Engineering

School of Engineering  
Blekinge Institute of Technology  
Box 520  
SE – 372 25 Ronneby  
Sweden

Internet : [www.bth.se/tek](http://www.bth.se/tek)  
Phone : +46 457 38 50 00  
Fax : + 46 457 271 25

## **ABSTRACT**

This work presents a security analysis of a distributed software system. Relevant threats have been identified and a set of possible countermeasures are presented. The different countermeasures have been compared against each other by looking at performance, scalability, flexibility, usability and cost considerations. A prototype system has been implemented as a proof-of-concept with database encryption, logging and access control.

**Keywords:** Database encryption, performance, VPN, Security threats.

# CONTENTS

<b>ABSTRACT</b> .....	<b>I</b>
<b>CONTENTS</b> .....	<b>II</b>
<b>1 INTRODUCTION</b> .....	<b>1</b>
<b>2 BACKGROUND</b> .....	<b>3</b>
2.1 SYSTEM DESCRIPTION .....	3
2.2 LAWS .....	3
2.3 REQUIREMENTS .....	3
2.4 DEFINITIONS.....	4
<b>3 SECURITY THREATS AND VULNERABILITIES</b> .....	<b>6</b>
3.1 DATABASE SERVER .....	6
3.1.1 <i>Unauthorized Server Access</i> .....	6
3.1.2 <i>Poor Configuration Management</i> .....	6
3.2 COMMUNICATION .....	7
3.2.1 <i>Information Gathering</i> .....	7
3.2.2 <i>Denial of Service Attacks</i> .....	7
3.2.3 <i>Spoofing</i> .....	7
3.2.4 <i>Sequence Number Guessing</i> .....	8
3.3 CLIENT APPLICATION.....	8
3.3.1 <i>SQL Injection</i> .....	9
3.3.2 <i>Password Cracking</i> .....	9
3.3.3 <i>Poor Configuration Management</i> .....	9
3.3.4 <i>Session Management</i> .....	9
3.3.5 <i>Buffer Overflows</i> .....	9
<b>4 CRYPTOGRAPHY</b> .....	<b>10</b>
4.1 TECHNIQUES.....	10
4.1.1 <i>Symmetric and asymmetric cryptography</i> .....	10
4.1.2 <i>Vulnerabilities</i> .....	10
4.1.3 <i>Symmetric cryptography</i> .....	11
4.1.4 <i>Digital Certificates</i> .....	13
4.2 CRYPTOGRAPHIC PROTOCOLS .....	13
4.2.1 <i>PPTP – Point-to-Point Tunnelling Protocol</i> .....	14
4.2.2 <i>L2TP – Layer 2 Tunnelling Protocol</i> .....	14
4.2.3 <i>IPSec – Internet Protocol Security</i> .....	16
4.2.4 <i>MPLS – Multiple Protocol Label Switching</i> .....	16
4.2.5 <i>SSL – Secure Socket Layer</i> .....	17
4.2.6 <i>Firewalls</i> .....	18
<b>5 SUGGESTED SECURITY COUNTERMEASURES</b> .....	<b>20</b>
5.1 DATABASE SERVER .....	20
5.1.1 <i>Logging</i> .....	20
5.1.2 <i>Access control</i> .....	22
5.1.3 <i>Configuration management</i> .....	25
5.2 DATABASE ENCRYPTION .....	25
5.2.1 <i>EFS</i> .....	26
5.2.2 <i>Column level encryption</i> .....	26
5.3 COMMUNICATION .....	34
5.3.1 <i>Comparisons</i> .....	35
5.4 CLIENT APPLICATION.....	36
5.4.1 <i>SQL injection</i> .....	36
5.4.2 <i>Access control</i> .....	36
5.4.3 <i>Password safety</i> .....	36

5.4.4	<i>Session control</i> .....	37
5.4.5	<i>Configuration management</i> .....	37
5.4.6	<i>Buffer Overflows</i> .....	37
<b>6</b>	<b>PROTOTYPE SYSTEM</b> .....	<b>38</b>
6.1	DESCRIPTION.....	38
6.2	TESTS.....	39
6.2.1	<i>Test Results</i> .....	39
<b>7</b>	<b>CONCLUSIONS</b> .....	<b>40</b>
	<b>ACKNOWLEDGEMENTS</b> .....	<b>41</b>
	<b>REFERENCES</b> .....	<b>42</b>
	<b>APPENDIX A TEST DATA</b> .....	<b>45</b>
	TEST 1 .....	45
	TEST 2 .....	45
	<b>APPENDIX B GLOSSARY</b> .....	<b>47</b>

# 1 INTRODUCTION

Internet is one of the most hostile environments that a software system can operate in, in terms of security threats. When handling personal data there are privacy laws and regulations, i.e., for integrity and confidentiality purpose, that need to be followed. So when having a distributed system operating on the Internet that handles sensitive personal data, it is important to evaluate the threats and security risks that are present, to be able to take appropriate countermeasures against them.

Buchner, a company located in northern Germany, is developing a software system for healthcare that will handle patient information. The system will be distributed over the Internet. In this system there are several vital aspects of the security that need to be evaluated. The aim is therefore to identify and study the security threats and law requirements present in a distributed system on the internet and evaluate different security solutions for them.

Questions that might be important in this situation are: How should the authentication be realized to be secure enough to be trusted? What access should different users have? What kind of encryption is needed? Should the data be stored encrypted? If so, how will that affect the performance? How should the passwords be handled? What impact on the usability will the security impose on the system?

The objectives are to identify the security requirements, according to the German law, that need to be met for securing patient information, to identify the security threats present in a distributed system in general, and identify which countermeasures that needs to be made to ensure the handling of sensitive patient data will follow German laws. Further we need to compare different existing solutions that could be implemented to secure the system, so that the patient data will be secure, and evaluate advantages and disadvantages between different solutions in terms of performance, usability, cost, scalability, and flexibility.

The research questions we are asking in this thesis are: which security threats exist for a distributed system on the Internet, and how serious are they? Which different countermeasures can be taken to deal with the security threats? Which countermeasures should be used?

This study is not an aim to find all existing threats for the system, i.e., to make a 100 % secure system. Identification of all security threats is a complex task which is depending not only on the system itself (meaning the software), but also its environment (operating systems and other software and hardware needed for running the system) and the users operating the system have to be taken into account. Security is also not a static process that is done at one point in time. There are all the time coming up reports on new attacks, for example CERT/CC [3] is an organization which receives reports on security attacks/vulnerabilities (on networks). Last year, for example, they received almost 3800 reports on new attacks.

The aim of this study is to look at known vulnerabilities from the perspective of the software, and evaluate them, to see how they can be considered to be threats to the system. Identification of these vulnerabilities is basically done by looking at already known attacks and analyzing how they could threaten the security of the system. Not known or not yet invented attacks are very hard to safeguard against.

The work presented is an industry based evaluation project. It is conducted as a literature survey where laws and known security problems are investigated and analyzed. There is also an investigation of security technologies. The literature study is followed by an empirical analysis of different solutions.

The rest of this thesis is organized as follows. The next chapter gives a presentation of the system with known prerequisites that will give the reader a better understanding

of the following analysis. The relevant law is also presented. The law requirements posed on the system are described and presented.

The third chapter identifies three security aspects of the system and analyses the security threats connected to those.

In the fourth chapter cryptography technologies and security protocols using cryptography are investigated.

The fifth chapter presents a set of security countermeasures and solutions for the law requirements posed on the system.

In the sixth chapter a prototype system is presented that implements a set of proposed countermeasures.

In the final chapter, chapter seven, presents a summary of the work and discusses conclusions that can be drawn.

## **2 BACKGROUND**

### **2.1 System description**

The system that is being developed is called BostonCare; it will be used for patient administration in healthcare organizations. This means that the system is handling sensitive patient information. It is a dynamic program with high level design possibilities, to supports modification of usage, therefore it is not at this point defined exactly how the system will be used in practice, and the exact information it will handle.

The system will be made up by a client application (windows forms) and a central database. The client application is developed with Visual Basic.NET in the Microsoft Visual Studio 2003 development environment, with the .NET framework version 1.1. The database is Microsoft SQL Server 2000, during development. Later when version 5 of MySQL is released, which has support for stored procedures; it might be an alternative option, to save licensing costs.

The same database server will be used by all clients in the same organization. One organization can be distributed over a wide area, with client applications on several different locations. The central database will be hosted by a third party, called an Application Service Provider (ASP). This means that the administrator of the database will be someone outside the organization and can therefore not be trusted.

The end-users of the system will be the healthcare personnel, doctors and nurses for example. The different users will have different level of access.

The end-users of the system may not have unique login accounts on the computers, where BostonCare is installed.

### **2.2 Laws**

In Germany there are data protection laws which restrict the handling of personal information. There is one federation law and also state laws. Public healthcare organizations in Germany need to follow the state law where they operate; therefore the data protection law in Schleswig-Holstein [29] has been used to elicit the data protection/security requirements for the system.

### **2.3 Requirements**

This section presents the requirements that the BostonCare application should implement for achieving the required level of security that is enforced by the data protection act in the state of Schleswig-Holstein. The requirements elicited are only the parts of the law that are relevant for the system. The law defines for example how disclosure of data to other organizations can be done. This is not relevant for the system, and it will not be part of the requirements.

- The sensitive data in the central database should be stored encrypted. (§5.1.1), (§6.3)
- The system need to have authentication mechanisms. (§5.1.2)
- The system should have data access control. Only authorised personnel may have access to the system. Also users at different wards should not be able to access each others patient information. (§6.1)

- The communication between the client and the central database should be encrypted. (§6.3)
- The system should keep a log of the following: Automatically collected personal data should be logged. Who collected it, when it was collected and in what manner. (§6.4)
- Modification to personal data should be logged. Who made the modification, when it was modified and in what manner. (§6.4)
- Automatic disclosure (revelation) of personal data should be logged. Who made the disclosure, when it was disclosed and in what manner. (§5.1.3), (§8.3)
- A log entry should exist exactly one year, thereafter it should be erased. (§6.4)
- There should be integrity support mechanisms; making sure no unwanted data changes are made. (§28.1)
- The system should support erasure/modification and blocking of stored data in the database.
- Data should be blocked automatically after a time period, that has been defined as maximum usage need, according to the client organization. (§28.3.1)
- Blocked data should not be possible to access from the client software.
- The collected data may not be used for other purposes than what it was collected for (§13).

To follow the law for the Protection of Personal Data in Schleswig-Holstein all these requirements needs to be supported by BostonCare. However some of these requirements are trivial to implement, for example the support for data erasure or blocking. These requirements will not be further investigated in this work.

## 2.4 Definitions

There are some terms that are important to understand in this context. The law defines these terms as described below.

- **Blocking** - a way to remove data from the system without erasing it (archiving). It is still present in the database, but it is not possible to process it via the client application. Or “Blocking means preventing the further processing of stored data“- according to the State Data Protection Act Schleswig-Holstein.
- **Confidentiality** - confidential information is information that has to be protected against unauthorised disclosure.
- **Availability** - a system must be available to users when they need it.
- **Integrity** - making sure data is complete and not altered in unwanted ways.

- Authorization - the granting of a right or a privilege, which enables a subject to have legitimate access to a system or a system's object.
- Authentication - a mechanism that determines if a user is who he claims to be.

## 3 SECURITY THREATS AND VULNERABILITIES

To make sure the system supports the law requirements the security has to be strong, the most important security attributes are the integrity and the confidentiality. When defining what security countermeasures the system should implement, it is important to know what threats exist, and how relevant they are. The research question that needs to be investigated is:

Which security threats exist for a distributed system on the Internet, and how serious are they?

To answer this question we first need to look at the existing vulnerabilities that are present in the system. The vulnerabilities should then be assessed to find out how big threat they are to the system. To do this, the system is divided up into the different assets that it is made up from.

- Database Server
- Client-Database Server communication
- Client application

For every asset of the system, the vulnerabilities are listed separately. All of the vulnerabilities listed below are ways an attacker/intruder can gain access to the system in an unwanted way and thereby compromises the data confidentiality, the integrity, or the availability of the system. According to Schneier [22], when assessing the threat risks, it is suitable to base the risk on real world examples or experiences from similar systems. You need to figure out who the attacker is and what his goals are.

There are two possible types of attackers of this system. There are internal attackers, i.e. a person in the organization who tries to use the system in unauthorized way. Reasons for this can be bribery or threats by somebody that wants to have access to information. There are external attackers, people who do not have access to the system.

There are passive and active attacks. Passive can be defined as: “attempts to learn or make use of information from the system but does not affect system resources”, this threatens the confidentiality. Active attack: “attempts to alter system resources or affect their operation”. These types of attacks threaten the integrity or the availability.

### 3.1 Database Server

The server holds the data used in the application. This data will partly be sensitive personal information. By law, this information shall be secured, especially in terms of confidentiality and integrity. The Database Server will be located at an Application Service Provider (ASP). Therefore the administrator of the Database Server may not be authorized to have access to the information in the database.

#### 3.1.1 Unauthorized Server Access

The access to the server needs to be restricted in several ways. If there is no access control when connecting to the database, there is a risk of unauthorized access. First the access needs to be restricted from unauthorized people, but also from the end-users and the administrators that already have access, but which should be restricted. The administrators of the database server will naturally have full access to the database, but according to the law; they are not allowed access to some of the data.

#### 3.1.2 Poor Configuration Management

If the configuration of the server is not taken care of, “the surface area” of possible attacks will be increased, which will result in a lower security.

## 3.2 Communication

The communication is the part that makes sure that the database server and the client application can communicate. This part is flexible and can be realized in different ways. At some installations, the communication will only go over a LAN (local area network), but for some, the access will be remote, over the Internet. Therefore it is important to protect the communication, and to make sure attackers can not compromise the security. For this reason it is important to find out which security threats that exist on the Internet. These threats also exist on a LAN, but the risks are generally lower, where the risk of external attackers is much lower.

### 3.2.1 Information Gathering

The first phase of an attack is to gather information. For example what kind of software is installed and the type of configuration that is used. The information helps the attacker to analyse the system and find its vulnerabilities. This attack is then followed by further, more advanced attacks.

Information gathering can be made in many different ways. One approach described by [21] is to use a port scanner, a software that asks a host which ports are open, which can for example reveal that there is a SQL Server installed if the TCP port 1433 is listening for incoming connections (which is the default port). The SQLPing tool can be used to send UDP broadcasts to whole subnets, to find out: the SQL server address, the SQL Server name, the Instance name, cluster status, version (only with SQLPing versions prior to 1.3), and the Netlib details(connection details).

Network eavesdropping means that a third party that could listen to the communication. When the user credentials are sent, the attacker could collect them, and use them later to gain access to the system. In a LAN, this is easily done by setting the network card in promiscuous mode, which makes it possible to receive all traffic on the network. The network traffic can then be monitored with a packet sniffer. Over the internet, the risk of this threat is very small according to an article by Tippett [32]. This is based on the fact that serious attempts to sniff the traffic on an Internet pipe failed; only the IP-headers were possible to log. This was done several years ago, and since then, the Internet pipes are much faster, which would make it even harder to do this now, much harder according to the article.

### 3.2.2 Denial of Service Attacks

If an attacker wants to reduce the availability of your system, it is possible to do a DoS – Denial of Service or a DDoS – Distributed Denial of Service attack. This type of attack is described by [19], it is done by sending more connection requests to the server than it can handle. If the attacker sends TCP SYN packages with a faked IP address (connection request package for a TCP/IP connection) the server answers with the SYN-ACK message and saves the connection in a queue and waits for the ACK from the client. If the client does not answer (which it can't, if the IP is faked), the server's queue will eventually become full and no more connections to the server can be established. The pending connections will of course time-out and eventually making the server available again.

### 3.2.3 Spoofing

#### 3.2.3.1 IP spoofing

Spoofing, described by [19], is a way for an attacker to pretend to be someone else on the network, via a fake IP address, and thereby get access to restricted information. This is done by changing the source address stored in the header of an IP package. This can be done because normally the source address is never authenticated, when a package is received.

It is possible for operators of networks to make sure that an IP package sent out is coming from the right IP area (for example if you are having Internet access from home, there is normally only one possible source address, making it impossible to do IP spoofing). This countermeasure is of course impossible to implement, because an attack can come from any network connected to the Internet.

### 3.2.3.2 DNS spoofing

When translating an URL into an IP address a DNS – Domain Name System is used. This can be taken advantage of. An attacker can map an URL to the false IP, if he has control of the DNS server. In this way someone who want to go to [www.google.com](http://www.google.com) for instance, might get the wrong IP address when he looks it up, and then comes to another site. This type of attack is also sometimes called phishing.

Another threat is if the DNS is used to authorize a user. In that case a remote client connects to an application server, the server quires the DNS with the client IP. The returned name is checked against a list with trusted domains to authorise the client. There is nothing that prevents an administrator to map an IP back to an FQDN - Fully Qualified Domain Name, which he is not authorized; therefore it is possible for an attacker to do a reverse mapping of an IP address, that map to an URL that the server trusts, to exploit this weakness.

DNSSEC is a security extension of the DNS created by IETF [9], which with the help of cryptography makes sure that data integrity and authentication is supported to all parts of the DNS system. The drawback with this extension is the performance overhead for all DNS operations that the encryption introduces.

DNS spoofing is not relevant for the BostonCare, because all communication will be done with IP addresses, thereby the use of a DNS will never be needed.

If an attacker uses IP Spoofing, it can be possible to create a successful DoS attack. This is of course not good, but the damage is not considered to be high in the BostonCare system and there is also no good reason for an attacker to try this either. The motive for doing this type of attack is if a competitor wants the customers to change to their system. If it is done, the customer will think the system is slow and unreliable, and maybe starts to consider switching to another provider. This kind to attacks is more likely to happen to web sites.

### 3.2.4 Sequence Number Guessing

This attack [19], which is sometimes also called session hijacking, can be done if there is a trust relationship between host A and host B. Meaning there is no authentication needed for creating a connection. For an attacker that wants to connect to host A while pretending to be host B does this attack in five steps:

First the attacker gathers information from A, to be able to predict the initial sequence numbers chosen by A. This can be done by sniffing the network or establish a few TCP connections to A directly.

Second, the attacker launches a DoS attack against B.

The attacker sends a SYN message to A (with B's IP as source address), to initiate a TCP connection handshake.

A sends a SYN-ACK message back to B with the initial sequence number(Y). Because of the DoS attack on B, B can't answer.

The attacker sends a fake ACK message back to A with a guessed sequence number to establish the TCP connection.

Now the connection is established and A thinks that B has connected.

## 3.3 Client Application

This component is the part that the normal end-users will be in contact with. It will be installed on the workstations at the client's workplace.

### 3.3.1 SQL Injection

With SQL injection, attackers can execute malicious SQL-queries in the database. There is a risk of unauthorized erasure, modification or insertion of data.

In the client application, this type of attack is only possible to do via the input fields. The input in the client application where the risk of this threat is highest is where the login is performed, after that the user needs to be authorized to use the system, and then you can consider the risk of this threat low.

Other ways to do this attack is to use other programs, such as the Microsoft Query Analyser, which gives direct access to the database and the possibility do any type of query. However to be able to connect to the database via another program, you still need to be authenticated.

Here are a few examples what a SQL injection attack can look like:

If there is an input field for a search an attacker can input the following string:

```
Zz' UNION SELECT 1,(SELECT @@version), SUSER_SNAME(),1 --
```

This input will, if it would succeed, return the SQL Server version and service pack status, the operating system version and the login used to access the database.

If the user account, used to connect to the database was an administrator (SA), the attacker will have the possibility to execute any command he wants.

### 3.3.2 Password Cracking

The user needs to be authenticated by logging in to the system at application start-up. When this is done, the user will have access to the system. If an attacker cracks a login and the password, he will have compromised the security. This can be done in several ways, different social engineering tricks for example or by doing multiple login attempts (dictionary attack) to brute force the password.

### 3.3.3 Poor Configuration Management

In the client application it is also important to have control over the configuration of the computer that the application is hosted on. If viruses or other malicious programs are installed, on the same computer, there is a risk that an attacker can read what the user types on the keyboard, or make screenshots of what the user is looking at in the application.

### 3.3.4 Session Management

It is possible that users forget to logout or close the application, before leaving the computer. If there is no session management, this introduces a risk that an attacker can get access in this way.

### 3.3.5 Buffer Overflows

Buffer overflows are a well known and commonly exploited vulnerability in software. In 1999, more than 50% of all security vulnerabilities reported to CERT/CC [3] was buffer overflow bugs [33].

A buffer overflow attack is done by sending more data, to a program input, than what is allocated by the program. This attack can be done both on the statically allocated memory (the heap) and the dynamically (the stack). If this is done, the program writes outside the memory that was allocated and thereby writes over other information. A sophisticated attack done on the stack memory can make it possible for an attacker to get full control over the program, by overwriting the return address of a function, for example.

## 4 CRYPTOGRAPHY

This section presents cryptographic techniques and security protocols that are commonly used to secure software systems.

### 4.1 Techniques

Cryptography is an essential technology when there is need for protecting information. It is not panacea, cryptography has its weaknesses, which needs to be understood and dealt with. Cryptography used in the right way can be a very powerful protection and provide confidentiality, authentication, integrity, and non-repudiation. But used in the wrong way, it can be useless.

Encryption is to “hide” the data and decryption is to make it readable again.

Plaintext -> encryption (algorithm and key) -> cipher text  
Cipher text -> decryption (algorithm and key) -> Plaintext

To encrypt data you need a key and an algorithm, together they are called a crypto system. The level of security provided by a specific encryption depends both on the key and the algorithm. The longer the key and the “stronger” the algorithm is, the more secure the encryption is, assuming the key is inaccessible.

#### 4.1.1 Symmetric and asymmetric cryptography

There are two types of encryption methods [33], symmetrical and asymmetric. Symmetric cryptography uses the same key for encryption and decryption, whereas asymmetric cryptography has one for encryption, the public key, and one for decryption, a private key. Asymmetric cryptography is practically easier, easier to trust and distribute. If you need to have a secure communication with someone over the Internet for example, it is possible to use asymmetric cryptography, you only distribute the public key. The only thing an attacker can do is to encrypt data; he can never decrypt your or your partners’ data, in this way the integrity can not break. When using symmetric cryptography you need to set up a secure communication where you first need to distribute the key. The advantage of the symmetric cryptography is that it is much faster than asymmetric, 1000 times or so. Therefore a combination of the two techniques is often used, for example in SSL and IPSec.

#### 4.1.2 Vulnerabilities

There are different ways to break the security of a cipher text. The encryption algorithm may have flaws; this is true if there are patterns in the cipher text. A crypto analysis can then use this pattern to either extract the plaintext or the key. All algorithms used today on the internet are considered resistant against crypto analysis according to [28].

Another threat is brute force attacks. An attacker that has a cipher text can test all possible keys, until a readable message appears. This attack is possible to do on the DES algorithm, which only uses a 56 bit key. A 56 bit key only has around  $10^{16}$  possibilities and that is already considered to a weak key. Michael Wiener developed in 1995 a special hardware dedicated to brute force a DES encryption and find the key. His hardware could find the key in 3.5 hours. The machine costed 1 million US\$ to develop. It is likely that better hardware is possible to develop with a higher budget [28]. Also now ten years later hardware is much faster, as a result of Moore's law about "computer evolution" a computer's computing power doubles every 18 months. Even taking this in consideration, brute forcing a 128 bit key, which has  $10^{38}$  possible keys, is considered almost impossible. This text is cited from the NIST's fact sheet [16] on the AES Algorithm:

"Assuming that one could build a machine that could recover a DES key in a second (i.e., could test  $2^{55}$  keys per second) it would take that machine approximately 149 thousand-billion (149 trillion) years to crack a 128-bit AES key. To put that into perspective, the universe is believed to be less than 20 billion years old."

The biggest threat to the encryption is the retrieval of the key. Key retrieval can be done in many ways; the software can be flawed in some way making it possible to retrieve the key from where it is stored. A recent report on this type of problem is for example the Hyper-Threading security flaw on Intel processors that makes it possible to steal RSA private keys [20]. Other threats are bribery, theft or social engineering [33].

### 4.1.3 Symmetric cryptography

There exist two main types of symmetric algorithms; stream and block ciphers. Stream ciphers are the fast type, and they encrypt a single byte at the time. Block ciphers work on blocks of data, and encrypt one block at the time. When encrypting more than a single block, there are different modes of operation. The mode is an important part of the encryption, and a strong algorithm can be made weak, with a bad mode of operation. An example of this can be seen in figure 3.

#### 4.1.3.1 Modes of operation

The easiest mode is ECB – electronic code book (figure 1). As described in [25], the plaintext is divided up in blocks which are encrypted separately. This mode is sometimes not recommended since blocks with the same data will have the same encryption. That will decrease the confidentiality, because it is possible to see patterns in the cipher. All blocks that have the same data, will look the same. This makes it possible to tamper with the cipher; which means that the integrity can be questioned.

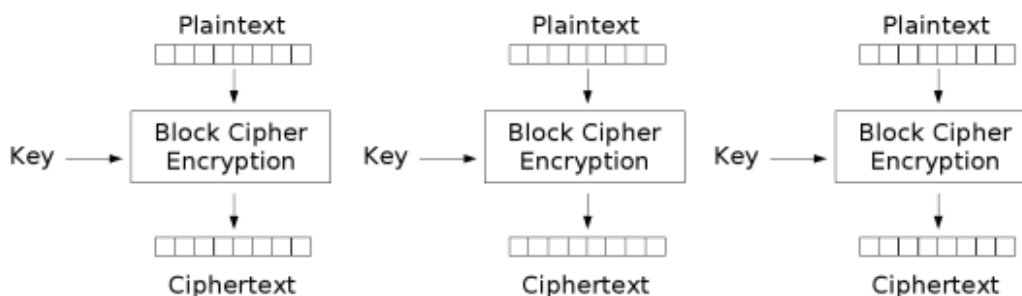


Figure 1. Electronic codebook (ECB) encryption mode.

CBC – Cipher block changing encryption (figure 2) works by XORing (XOR is a bitwise operator, which takes two bits as input and returns true if they are not equal) the encrypted result from the previous block together with the next block of plaintext before encrypting it [25]. This results in a more secure encryption than ECB, because the risk that two equal plaintexts will have the same cipher text is minimal. This means that every block encryption is depending on the previous block. This helps the integrity, it is impossible to change block order without discovery. The confidentiality is also higher, patterns are also harder too see, because two equal plaintext blocks will not likely have the same cipher.

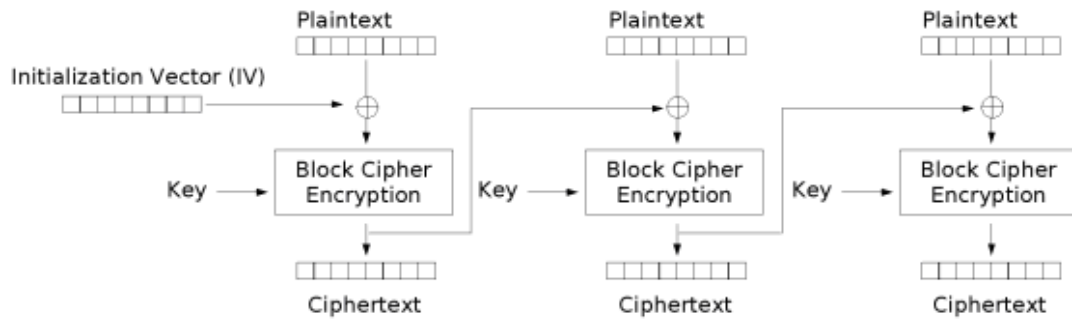


Figure 2. Cipher Block Changing (CBC) encryption mode.

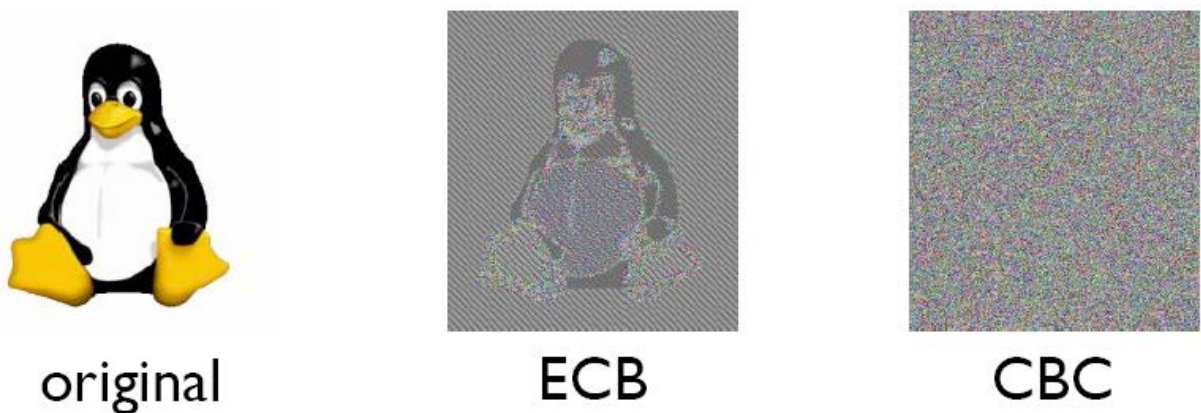


Figure 3. The figure shows a picture in three formats, no encryption, with ECB and with CBC encryption mode.

The bitmap in figure 3 shows that it is very obvious that the ECB encryption is not the right mode for this type of encryption.

Counter mode, Cipher feedback (CFB), and output feedback (OFB) are three different cipher modes that turn a block cipher into a stream cipher [25]. They all mitigate the risk introduced when a plaintext block reappears in a message, or across multiple messages. The counter mode does this by mixing the plaintext with a sequence number, which is not allowed to be repeated if the security is to be maintained. The cipher feedback (CFB) and output feedback (OFB) modes XOR keystream blocks with the plaintext blocks to get the cipher text. Just as with other stream ciphers, flipping a bit in the cipher text produces a flipped bit in the plaintext at the same location.

#### 4.1.3.2 Algorithms

DES – Data Encryption Standard was introduced in the 70’s by the NIST (U.S. National Institute of Standards and Technology). It has been and still is widely used. It uses a key length up to 56 bits and uses 64 bit block sizes. It has support for CBC mode. DES can be used in stream cipher mode (CFB) and in OFB. Today the basic version of the DES is not considered secure [19], but there are variants of this algorithm; Triple DES, DESX, GDES, and RDES that are cryptographically stronger.

Triple DES is widely used, by DPAPI in Windows and with SSL for instance. It uses DES to encrypt the data three times in a row. There are two or three keys used. This technique makes the algorithm not very fast (approximately three times slower than DES), and for a real time system a faster algorithm is needed.

IDEA – The International Data Encryption Algorithm, it was developed by Xuejia Lai and James Massey and made public in 1990 [13]. It uses a 128 bit key and a 64 bit block size. It is used in Pretty Good Privacy (PGP) V2.0. It is a patented algorithm, which may only be used freely for private use.

SAFER [19] – Developed by James Massey after the development of IDEA and first presented in [12]. There are two variants, SAFER-64 with a 64 bit key and SAFER-128 with a 128 bit key. SAFER-64 uses 6 rounds and SAFER-128 recommends 10.

Blowfish – This algorithm was developed by Bruce Schneier and first presented in [24]. It uses like the DES algorithm also 64-bit blocks. It consists of two parts: a key-expansion part and an encryption part. The key expansion converts a variable-length key of maximum 56 bytes (448 bits) into a set of sub keys. It uses 16 rounds, where every round consists of a key-dependent permutation, and a key- and data-dependent substitution. All operations are XORs and additions on 32-bit words. Blowfish is according to Schneier faster compared to both IDEA and DES.

AES – Advanced Encryption Standard, is a NIST [16] standard that is supposed to replace the DES in the future. It is originated from the Rijndael proposal made by Joan Daemen and Vincent Rijmen. It is one of the fastest algorithms that exist. It uses a block size of 128 bits, and keys with 128, 192 or 256 bits. This algorithm is not patented.

#### 4.1.4 Digital Certificates

Digital certificates are normally used for authentication and key exchange, when communicating over the internet. A certificate exists of a public key, identity information and one or more digital signatures. The signature(s) is there to show that someone else can confirm that the identity information is correct. The signature normally comes from a trusted third party, a Certificate Authority (CA). Digital certificates help protect against man-in-the-middle attacks. In this type of attack the attacker acts as a proxy between the client and the server, to be able to read and modify the communication. However if the signature is self-made, which they can be, there is no protection against this attack. A very common standard is the X.509 certificate format. X.509 is used by IPsec and SSL for example.

## 4.2 Cryptographic Protocols

To secure a communication over a public net like the Internet, encryption is needed. A VPN is a „Virtual Private Network”; communication in a secure tunnel over an un-trusted net. It is a way of communicating over the internet, but with a much lower security risk.

When the communication takes place, the tunnel first has to be established. The tunnel is established as a session, at start there is a negotiation between the two endpoints, the server host and the client host, to agree to the tunnel. During the negotiation the session specific parameters are decided, like encryption details, address assignment and other relevant parameters. For this propose a tunnel management protocol is used. The management protocol also takes care of maintenance and termination of the tunnel. Once the tunnel is created, data can be transferred. This is done with the help of a tunnel data transfer protocol.

A VPN protocol has support for the following things:

- Establishment, maintenance and termination of tunnels.

- Address management. It must make sure that the private addresses of the clients are hidden from the outside.
- User authentication. The VPN must verify the identity of all connected hosts, and make sure only authorised hosts can have access. This prevents attackers from doing information gathering attacks, as mentioned earlier. There should also be logging support, to be able to audit who was connecting, and when.
- Negotiation of how to realize the cryptographic procedures, key exchange, encryption procedures and signature procedures.
- Encryption/decryption of the data-packages. This prevents eavesdropping and also makes sure on one tampers with the packages.

The five most common protocols are:

- PPTP, Point-to-Point Tunnelling Protocol
- L2TP, Layer 2 Tunnelling Protocol
- IPSec tunnel mode, Internet Protocol Security
- SSL/TLS, Secure Socket Layer
- MPLS, Multiple Protocol Label Switching

Table 1. The different protocols exist on different levels of the OSI model, which can be seen in this table.

OSI Model	TCP/IP Protocol Stack	VPN Protocols
7. Application Layer	Application (HTTP, FTP, POP)	
6. Presentation Layer		
5. Session Layer		
4. Transport Layer	TCP, UDP	SSL
3. Network Layer	IP	IPSec
2. Data link Layer	Network card	L2TP, PPTP, MPLS
1. Physical Layer		

#### 4.2.1 PPTP – Point-to-Point Tunnelling Protocol

The PPTP protocol [19] works on the second layer in the OSI model and supports sending and encrypting multi-protocol transmissions. It was developed by the PPTP form, including Microsoft Corporation, Ascend Communications, Primary Access, ECI Telematics, and 3Com, and is documented in RFC 2637 at the Internet Engineering Task Force (IETF).

The underlying protocol used can only be IP. The payload is encapsulated and sent in a GRE encapsulated PPP– Point to Point Protocol package and can be encrypted and compressed. When setting up a VPN in Windows, PPTP normally uses the MPPE protocol (Microsoft Point-to-Point Encryption) to handle the encryption [19]. The tunnel maintenance is done over a TCP connection.

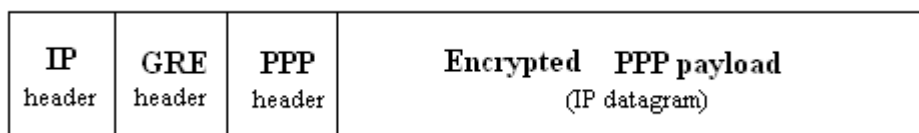


Figure 4. The PPTP encapsulation scheme.

#### 4.2.2 L2TP – Layer 2 Tunnelling Protocol

The L2TP protocol [19] also works on the second layer in the OSI model and supports sending and encrypting multi-protocol transmissions. The L2TP is an IETF standard that is a combination of the best features of the L2F and the PPTP protocol and is documented in RFC 2661. It was derived from the L2F – Layer 2 Forwarding

implemented primarily in Cisco products. It is also an extension of the PPP, which is an important component of VPN architecture.

The data can be sent in any medium that supports point-to-point datagram delivery, such as IP, X.25, Frame Relay, or ATM, as opposed to only IP which the PPTP protocol only supports. For sending the payload L2TP uses UDP to send L2TP encapsulated PPP packages. The PPP can as with the PPTP encrypt and compress the payload. Normally L2TP is used together with the IPsec protocol to provide encryption. The tunnel maintenance is done over UDP, with a set of L2TP messages.

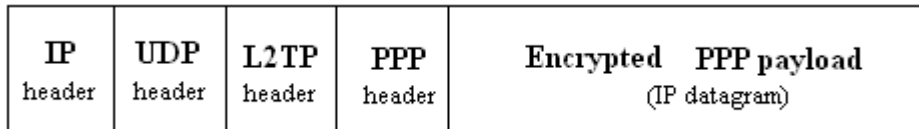


Figure 5. The L2TP encapsulation scheme.

### Comparison between PPTP and L2TP

There are some differences between the two protocols [11].

1. The L2TP have header compression support. This results in a four byte header, instead of a six byte header as in the PPTP.
2. There is no tunnel authentication support in the PPTP protocol. But this can be solved by using IPsec or SSL as extra layer.
3. The PPTP does not support multiple tunnel usage between two endpoints as opposed to L2TP where you can have different tunnels for different qualities of services.

A drawback with both the L2TP and the PPTP protocol is that there is no single standard for encryption and authentication. This means that two different products can be incompatible because of this reason, even if they both use the same protocol (PPTP to PPTP, or L2TP to L2TP). But this problem is actually of no concern in this case, because there will only be known nodes connected to the VPN, which of course will be configured with the same protocols.

### Authentication

When setting up the remote connection, it is very important to make sure who you are communicating with. It does not help to have a secure channel if the host on the other end is the attacker, then the whole point of the secure channel is gone. To mitigate this problem, authentication is needed. When using the PPP protocol, which is used by both PPTP and L2TP, there are a number of different ways to do this. When using L2TP/IPsec, this authentication does not need to be very secure, because it occurs after the IPsec already has established a secure channel.

For Windows Server 2003 and Windows XP there is support for the following PPP authentication protocols:

- Password Authentication Protocol (PAP)
- Challenge-Handshake Authentication Protocol (CHAP)
- Microsoft Challenge-Handshake Authentication Protocol (MS-CHAP)
- MS-CHAP version 2 (MS-CHAP v2)
- Extensible Authentication Protocol (EAP/TLS)

Microsoft suggests that the EAP/TLS protocol should be used together with digital certificates or smartcards to create the most secure authentication scheme [35]. All the other protocols have known weaknesses that could be exploited by an attacker.

### 4.2.3 IPSec – Internet Protocol Security

IPSec is a technology standard (IETF) to support end-to-end security over a network. IPSec has two modes; tunnel mode and transport mode. In a VPN solution the tunnel mode is used, which is the most secure version, it protects both the header and the payload, the transport mode only protect the payload. The IPSec protocol works on the third layer in the OSI model and supports only sending and encrypting IP traffic [31]. It can be used both with IPv4 and IPv6.

It is build up of two parts, the AH (Authentication header) and the ESP (Encapsulating Security Payload). The AH (Authentication header) provides integrity through authentication for IP Packages. Depending on which crypto algorithm that is used and how keying is performed the AH may also provide non repudiation of origin services (which prevents the sender from denying that he sent the information). It is also possible to offer anti replay service at the discretion of the receiver to help counter denial of service attacks.

The ESP (Encapsulating Security Payload) provides the data confidentiality and partial traffic flow confidentiality when used in tunnel mode. This can be achieved by encryption from almost any available symmetric encryption standard. The minimum encryption used is the DES algorithm. ESP can also provides some authentication, however the authentication done by the AH is stronger. The ESP authentication does not protect lower level IP headers. If only upper layer protocols need to be authentication then ESP authentication should be used, to save space. If for example a NAT is used, the AH is not needed and can not be used. The NAT makes sure the source address will be hidden, making it unnecessary to verify it. It is also not possible to traverse a NAT when the AH is used.

The IPSec tunnel consists of a client and a server. The client sends the whole IP package encrypted to the server, which decrypts the package and forwards it to the recipient. This results in a good protection against “man in the middle attacks” by hiding IP addresses externally. IPSec is controlled by a security policy. This policy is used as a preference when the client and the server agree on the encryption and the authentication mechanisms they should use.

As mentioned above, when setting up a remote access VPN with IPSec, it should be used in combination with the L2TP protocol. The reason for this is that there is not full protection from traffic control when only using the IPSec protocol [19]. There is no standard way to do IP address assignment and name server assignment. The IPSec tunnel is not represented as a logical interface over which packets can be forwarded and received, routes cannot be assigned to use the IPSec tunnel and routing protocols do not operate over IPSec tunnels [35]. Therefore it is advised to use IPSec as an encryption scheme in combination with L2TP or PPTP, to increase the protection. This combination is described in RFC 3193, of the IETF.

IPSec provides support for the Internet Key Exchange (IKE) a protocol for automatic key management. IKE provides negotiation services and key derivation services. IKE can be used with X.509 certificates for automatic authentication [31].

### 4.2.4 MPLS – Multiple Protocol Label Switching

MPLS is an alternative to the layer 2 protocols. This protocol logically separates the traffic of the VPN from other network traffic without the need for a layer 2 tunnelling protocol. This means that there is no need for a firewall against attackers because it is not possible to access the VPN from the internet. MPLS can be applied with any of the layer three protocols (therefore it is called Multiple). It can be applied and used on top of IP, Frame Relay, ATM, or Ethernet. One advantage of MPLS is the scalability of the protocol, it is possible to support multiple VPN:s for multiple customers without using too many virtual circuits, and this is the main issue when having scalability problems [4]. MPLS also has a good support for quality of service. This is because there is support for much larger package sizes, as compared to the ATM counterpart.

## 4.2.5 SSL – Secure Socket Layer

Using a SSL, version 3 (also called TLS – Transport Layer Security) VPN instead of the above solutions is an option. SSL is a standard way to create a secure tunnel over the internet. It is defined in RFC 2246 from IETF. It operates on the transport layer over TCP/IP, to secure a connection on the application level. Any application that uses TCP can therefore be supported by SSL encrypted communication. When SSL is used for a VPN, it fills the same purpose as any other VPN, to establish a site-to-site tunnel. SSL also has built in support for data compression. It is possible to use SSL with or without authentication. The authentication is done with RSA public keys and X.509v3 digital certificates. SSL can be used without authentication, and then the Diffie-Hellman [28] protocol is used for key exchange. The encryption can be done with several different algorithms with varying key lengths. Typical algorithms include Data Encryption Standard (DES), Triple DES (3-DES), RC2, RC4, and Advanced Encryption Standard (AES).

The SSL protocol

First the SSL Handshake protocol is used to establish the connection.

1. When a connection is created, the “cipher suit” is agreed upon (the strongest encryption support from both parties). The cipher suit consists of a symmetric encryption algorithm, a message digest method (hash method) and an authentication method.

The client sends a ClientHello message with its SSL version and the cipher suit and compression it supports. The server replies with a ServerHello message with information on the cipher suit and compression it has chosen together with a session ID for the connection. If there is no common cipher suit, the server sends a “handshake failure” message.

2. Server authentication

If the server is using authentication, which is normal, it is done at this point. The server sends its signed X.509v3 certificate. If the client needs to be authenticated, which is optional, the server sends a certificate request message. The client replies with its signed X.509v3 client certificate or sends a no certificate alert. At this point the server can choose to send a handshake failure or continue.

3. The symmetric encryption key is shared.

This procedure is depending on the encryption algorithm selected, but typically the client generates a pre master secret using the selected hash method. The secret is put in a digital envelope by encrypting it with the server’s public key and then sent in a “ClientKeyExchange” message. Now both ends can generate the real encryption key, based on this pre secret. From the real key four different keys are constructed:

The Client Write Key and the Client Write MAC Key, which are used for outgoing data from the client. Then there are the Server Write Key and the Server Write MAC Key, used in the opposite direction [36].

4. Client authentication

If using client authentication, the client must prove that it knows the correct RSA private key by sending a “CertificateVerify” message. This message has the pre secret that has already been sent once, but it is manipulated to protect the integrity, so that on third party can tamper with it without being discovered. The secret is then signed with the client’s private key before it is sent. The server then checks the validity of the message by comparing it with the client certificate.

5. Start communicate

Now both parties send a “ChangeCipherSpec” message to confirm that they are ready to use the symmetric encryption that they have agreed upon.

## 6. Integrity check

Both parties send computed MD5 and SHA hash values of the conversation so far. This confirms that all messages have been received and that no attacker has tampered with the communication.

After this, communication can start. This is done with the SSL Record Protocol. The data is fragmented into blocks with 16,383 bytes as max size. From the blocks SSL Records are created and transferred. A record includes a content type, which describes the higher protocol layer that needs to be used to process the data after it has passed the SSL protocol (HTTPS, NMTPS or SMTPS for example), the protocol version number, normally SSL 3.0. The data length, the data-block (normally compressed and encrypted) and a MAC. The MAC is used to authenticate the sender. How to verify the MAC is defined in the cipher specification. By default the MAC is constructed similar to HMAC specified in RFC 2104 [19] (used by the IPsec AH to authenticate the payload). A hash value is generated from concatenating the Client Write MAC key, the record length and a sequence number. The MAC gives protection against replay attacks and gives authentication support.

## SSL Weaknesses

There have been some studies to find weaknesses of the SSL protocol [19]. The protocol is excellent against passive attacks (eavesdropping etc), But with a sophisticated active attack it is possible to get the server's private key. Thereby being able to read the ClientKeyExchange message to get the pre secret and then create the keys used for the encryption. If this is done, the attacker will be able to read the whole conversation. But to succeed, the attacker has to make between 300.000 and 2 million connections to the server, before the administrator would become suspicious. This is however not very realistic.

## 4.2.6 Firewalls

If you have a computer or a network connected to the internet, a firewall is the basic protection that is used to hide and protect yourself against attackers. This is an important part of a VPN. It is a filter that stops unwanted traffic and lets authenticated traffic through. It stops attackers to use or exploit services running on your system and doing information gathering or DoS attacks, as mentioned in chapter three. It can stop both in and out going transmissions. There are three main types of firewalls, depending on which OSI level they filter at, the IP-package level, the TCP session level or the Application level. Basically you can say that the higher the filter works, the more fine grained the protection is and the more it will affect the performance.

A common type of firewall works as a gateway for the whole network and forwards all incoming traffic to the right host on the network. It can filter the traffic based on ports, addresses and protocol used. This type supports easy auditing and control mechanisms, but for performance it is not an optimal solution, especially when the network grows. It is also not easy to do effective outgoing traffic filtering with this type of firewall. Installation of application level firewalls on the hosts directly is then a better alternative. It also protects against internal threats, both between the hosts themselves and from spy-ware sending sensitive information from your system to a possible attacker.

Another type of firewall can be a router; which can be seen as a firewall that filters traffic based on the source and destination addresses and ports. It is very simple and if the addresses of the communicating parties are known, and won't change, this is a good alternative to the above type.

A NAT (network address translator), or a proxy server is also a kind of firewall, the are different, but both works as a computer in the middle of a communication link, the external party never knows the real address of the host in the network. An attack

therefore, first needs to penetrate the NAT or the proxy server before he can access the actual host, which makes these types of firewalls safer. However these types of firewalls are bad for the performance both latency and throughput than the routers or host based firewalls.

## 5 SUGGESTED SECURITY COUNTERMEASURES

This section presents countermeasures needed to mitigate the threats and vulnerabilities that were presented in chapter three. There are also solutions presented for some of the law requirements. For every identified asset, the countermeasures are separately presented. There are different solutions and design decisions that are presented and discussed.

### 5.1 Database Server

#### 5.1.1 Logging

To follow the law requirements, all views, updates and adding of sensitive data needs to be logged. This provides integrity to the system by the possibility to keep track of system usage. According to the law, it should be possible to audit who did what at what time. The log should be readable, meaning that there should be a program available that can show the collected data, in an understandable way.

This can be done in different ways. For Microsoft SQL Server, there are several [14] third party products available, which work by reading the Server transaction log. Some of these products are compliant with the American act on electronic access to patient information (HIPA).

##### 5.1.1.1 Own implementation

Another solution is to manually implement the logging and store the log data in the database. This can be done in different ways. One way, would be to implement it in the client application, together with the crypto module. Every time encryption or decryption has to take place, you automatically know that sensitive data is being accessed, updated or inserted. For this reason, it would be practical to have the control of the logging at the same place. The only problem is that this would require extra calls to the database. When the database is remote, this may be a bad idea performance wise.

A better way would be to use additional SQL statements every time a database command is made. To do this, all calls to the database should be made via stored procedures. Use of stored procedures is a good idea, for other reasons also. They provide protection against SQL injections; they make it possible to increase the access control (no users need to have direct access to the tables) and also increase the performance of the database.

In the stored procedures, to do the logging, there only needs to be an extra SQL statement that inserts a new record to a log table. The log table should store data like: username, a timestamp, and which data was accessed or modified [29], (§5.1.3 & §8.3). The viewing of the logged data is easily accessed with the SQL Query Analyzer, by using normal SQL queries. An example of how this could look like (the timestamp and the current user are automatically added to the new log row):

```
CREATE PROCEDURE SP_INSERT_PATIENT
@name          varchar(50),
@lastname      varchar(50),
@birthdate     varchar(50),
@phone         varchar(50)
AS
INSERT INTO PATIENT (name, lastname, birthdate, phone)
VALUES (@name, @lastname, @birthdate, @phone)
INSERT INTO PATIENT_INSERT_LOG (name, birthdate)
```

```
VALUES (@name, @birthdate)
GO
```

There is however a limitation when doing own logging. It is not possible to log failed login attempts. For this you need to use built-in logging.

#### 5.1.1.2 Triggers

The logging could also be done with triggers. However the problem when using triggers is that you don't have direct access to the data that is modified and triggers can not be used for viewing of data.

A trigger is a stored procedure that is invoked when a table or view is being modified (INSERT, UPDATE and DELETE). A trigger can process Transact-SQL statements and will be part of the current transaction; if there will be a server error the trigger is also rolled back. The triggers can then be used for adding integrity to the database.

#### 5.1.1.3 Built-in logging

In Microsoft SQL Server there are several possibilities to log database activities [27]. The audit log is the simplest logging, which can only log login activities. It has four different settings:

- No (no logging)
- Failure (log only failed login attempts)
- Success (log only successful logins)
- All (log both failed and successful logins)

This is however too simple. A more advanced logging option would be to use full C2 auditing, which meets the C2 evaluation requirements from the Department of Defence Trusted Computer Evaluation Criteria.

This type of log is built-in to the database server. It uses a default profile to decide what to log. When having the C2 log on, it is wise to use a separate disk to store the log on, it helps performance and if the disk is full and it is not possible to add more to the log, the server halts all execution, until it is possible to write to the log again.

The log files are stored as trace files and can be viewed by the SQL Profiler tool. The disadvantage with the C2 logging is that the logging is too extensive, which is bad both performance wise and for readability of the log. Therefore it is better to use trace-logging, if you plan to the built-in logging. It works in the same way, except that you define what to log manually with a set of system Stored Procedures:

- `sp_trace_create` Creates the trace definition
- `sp_trace_setevent` Defines which events will be traced and which data columns will be stored in the Trace file
- `sp_trace_setfilter` Creates the filter definition for the trace
- `sp_trace_setstatus` Starts, stops, or deletes the trace definition

The creation of the traces can be done directly as SQL scripts or with the SQL Profiler, which provides a GUI. The trace-log works on the Server level, so if you for example have two databases on the same server, you need to create two different trace-logs, to separate the auditing of the different databases. To activate the trace-log, the trace script needs to be run every time the server starts. This can however be atomised. The problem with the trace logging is that it is only possible to log predefined data values. You can not log the data that is viewed for example, only the command text, the user, the time of the request, and similar data. Trace-logging has an advantage that it is separated from the rest of the database. There is no need to go into all of the Stored Procedures and manually add logging statements.

#### 5.1.1.4 MySQL

For MySQL there exist several logs. The binary log has the best performance and generates only around 1% extra overhead. However it is only useful for recovery and analysis, it does not log any read queries.

The query log is therefore the only available built-in log useful and is able to support the logging requirements. It has however not been tested, and it is likely not a good option because there is no documentation mentioning any limitation of the logging [15].

#### 5.1.1.5 Discussion

A third party product is a very good option, for example the Lumigent software which is compliant with HIIPA, but works only if the database server would be a Microsoft SQL server. For this reason and because of the licensing costs, this option should be avoided.

The best solution is to use the same type of logging, independent of which database server that is used. Therefore the use of the built-in logging support should also be avoided, if the software should support both database servers.

Therefore the choice of logging should be an own implementation, preferable in the database. The main advantage with this solution is that it is easy to log exactly what is needed and to store it in a user friendly way. It is also the best option in flexibility, the possibility to switch to MySQL would be easier. The disadvantages with this is that it will add extra development costs and the performance will probably suffer more than when only using the transaction log in combination with a third party auditing tool, for example.

To log logins and failed logins a good idea is to use the application log when using Microsoft SQL Server. For MySQL it is possible to use the Query log for this.

It can be a good idea to support different levels of logging. In some situations, when there is an emergency (when access restrictions are set aside) for example. It can be needed to do extra logging. This can added by modifying the stored procedures and add a log level parameter. Here is a modified version of the previous stored procedure:

```
CREATE PROCEDURE SP_INSERT_PATIENT
@name varchar(50),
@lastname          varchar(50),
@birthdate         varchar(50),
@phone varchar(50),
@log               int
AS
INSERT INTO PATIENT (name, lastname, birthdate, phone)
VALUES (@name, @lastname, @birthdate, @phone)
If( @log = 1)
begin
INSERT INTO PATIENT_INSERT_LOG (name, birthdate)
VALUES (@name, @birthdate)
End
GO
```

#### 5.1.2 Access control

Access control is how to achieve confidentiality, availability and integrity. It is basically what security is all about. Mechanisms to let authorized people have system access and to deny system access for all others. There need to be access control mechanisms in the database server.

To support this, the first policy should be to use authentication of the user when setting up a database connection. All users should have personal user accounts, this to

be able to log who does what and to make it possible to apply different access control to different users.

The second policy to support the access control is to authorize all access using least privilege. The users can only access what they are explicitly allowed to. This is to protect sensitive resources from being accessed by underprivileged users. For example it is not wise to use the administrator (SA) account to connect an application to a database if the only database access needed is to read data. If the account would be compromised, an attacker would have full access to the database, which would be much worse than if he would only have read privileges on a limited number of tables.

#### **5.1.2.1 Authentication**

For the database authentication there are three different possibilities. The first solution is to authenticate the user, when using Microsoft SQL Server, with Windows Integrated Security. This type of login has advantages over the other option, SQL Security authentication. Windows Integrated Security uses the Windows 2000 security system which provides features such as password expiration, account-lockout (after several failed login attempts), secure validation, encryption of passwords and auditing. Therefore this solution should be used if possible, but there are several requirements that have to be fulfilled before this authentication can be used. Active Directory, Kerberos and delegation support needs to be installed and the database needs to be on a trusted or on the same domain as the client computer. It also requires all users to have unique Windows accounts. This may not be the case, and even if it were, the system would be dependent on Microsoft SQL Server. For these reason this solution should be avoided.

The second solution is to use SQL Security and let users have a unique database user accounts. This is a feature that is supported in both Microsoft SQL Server and MySQL. If this is done, the user credentials needs to be sent to the database server. This security vulnerability of course needs to be mitigated, which can easily be done by having an encrypted communication such as SSL. An advantage with this type of login is that there is good support for logging. If every user has an own login, the username is easy to log. There is no need for extra parameter sending.

The third solution is to use SQL Security, but to let all users have the same database account. The drawback will be that user actions can not be logged as easily and the users still need authentication in some way at the client application. Another disadvantage with this solution is that the user credentials needed for the database connection needs to be stored on all client workstations, this will be a security risk. Also if the credentials are compromised and need to be renewed, all workstations need to be updated with the new password.

From these three alternatives, one solution needs to be selected. When looking at the BostonCare software the first alternative can be ruled out directly. It is not feasible to assume that the database will be on a trusted or the same domain as the clients are. Even if that was true, there is no support for this authentication in MySQL, which might be used in the future. Therefore only solution two and three are possible. And when comparing these, it is logical to use the second alternative. Both because of the ease of logging and because of the security risk introduced when storing the credentials on the client workstations.

When using the SQL Security, there are no built-in features for safeguarding against password vulnerabilities. This needs to be addressed. There has to be a check that makes sure that passwords are strong and that they are changed regularly. There has to be a lockout feature installed, to prevent dictionary attacks.

### 5.1.2.2 Authorization

Authorization can be controlled in two ways, either by controlling the user or the objects of concern (patient records or functionality). User based control is known as capabilities and object based control is known as access control lists or ACL's [1].

Both types of control are useful. Several different users will access the system, doctors, nurses and administrators. They all have their associated roles with different assignments. Capabilities can be defined and appointed to different roles, for the different types of users. Access control lists are useful for defining who have access to which patient records. Together these two controls can provide support for the law requirements on the patient privacy.

To support the policy of using least privileges, there has to be capability restriction on what the users can do. If a doctor for example is not able to plan and insert new appointments for the patients, access to the stored procedures for this should also be revoked.

The access control in Microsoft SQL Server works like Windows to control the access control. There exist Users and Roles.

#### User

A user is a normal user account which is used for authenticate the user. It is possible to set the access control directly for the users, but it should be avoided. When having a complex access structure, or when several users have the same access rights, it makes more sense to use roles, and add users to the roles. It gives also a better overview, when using roles for defining access control you have the benefit of giving the roles appropriate names, describing what the role gives access or restrict access to.

#### Roles

A role is comparable to a group in Windows; it can be seen as a container that holds users. There are two types of roles, server- and database-roles. The server-roles are used to configure and control the SQL Server. The database-roles are used to grant permissions or set restrictions on the database objects (tables, views, procedures etc). The permissions that exists are SELECT, UPDATE, INSERT, DELETE and EXEC. For columns, you can only set SELECT and UPDATE. When a user has two conflicting roles, one that is restricting access to a specific object and one that is giving access to it, the restricting role has precedence.

To realize the second policy all users will be created without any rights at all. To control the access rights of the users, roles will be created with specific access rights. The users will then be added to roles when there is a need for access.

MySQL also has access control support. It is similar to SQL Server, with possibility to control access down to column level [15].

#### Row level access control

The ACL's (object based control) as mentioned earlier are logically used for controlling which user has access to which patient. Controlling this, the other way around is a bad idea, because there are much more patients than users.

A simpler version, but with less control granularity, can however be used, where every patient is connected to a sub-division, and only users in the sub-division has access to these patients. Use of access control lists for the patients are however recommended both by [1] and [7] when dealing with access control mechanisms in information systems for healthcare. In this way, it is possible to exactly control which users have access to which patient. To support this, there has to be row level control. This is however a feature that is missing in both Database systems, it is not possible to set table row access.

This feature can be implemented in different ways, depending on how the access control should be. You can let the users have separate databases. This will however mean that they won't be able to share any data at all, so when there is an emergency for example, this solution is not very smart. The only situation when this type of control should be used is when several different organizations, which do not share patients, share the same database server. For implementing ACL's another solution is needed.

For every table that needs row level access control, there should be an extra column (foreign key) that connects the row with an ACL table, in the ACL table all users that have access to the row are added. Then to access the tables, views should be used that filters the table with the ACL table. The good thing with this solution is that there only needs to be one view per table, no matter how many users there are. In an emergency, when the users need to have access to normally restricted records, another view without any restrictions, can be used.

To secure the access control further, no user will have direct access to the tables. The only access that should be granted is access to stored procedures, not even the views. This makes sure all commands will be logged, if the stored procedures have logging, which should be the case. It also makes sure that no one can modify the log tables. Here is an example of a view can look:

```
CREATE VIEW dbo.patientview
AS
SELECT  dbo.patientview.*
FROM    dbo.patient, dbo.patient_acl
WHERE
(
dbo.patient_acl.user = SUSER_SNAME() AND
dbo.patient_acl.id = dbo.patient.acl_id
)
```

(`user_sname()` is a function in Microsoft SQL Server that returns the name of the current user)

In the case of when a patient is moved and gets a new doctor, the ACL just needs to be changed to the new doctor, to change the access control of the patient.

In some databases, working with views can be very slow, when there many records in a table. This is because views work by making a copy in the memory of the real table, before the query is executed. In Microsoft SQL Server, this is not the case. A test done with a table holding 200 000 records showed that queries on a view were having the same performance as queries on the real table.

### 5.1.3 Configuration management

When the users log in on the client application, they use the same login credentials as they use to connect to the database. This makes it important to check that all new user accounts in the database are strong. There should also not exist any guest account on the database and any other accounts that are not being used. An anti virus program and a spy ware detection program should be used. Frequent security updates should be done, so that no known vulnerabilities exist. Harden weak default settings and disable unused communication protocols and services. A firewall should also be used.

## 5.2 Database Encryption

For the database server the most relevant threat is the persons who have direct access to it, the administrator (not the administrator of the client workstations), hackers or persons who break in and steal the physical disks of data. Loss of confidentiality is

the biggest threat, which needs to be mitigated. The countermeasure that needs to be implemented for these types of threats is encryption of the database.

There are different ways to do this. There are many COTS software solutions that can do this for you, for example XP\_CRYPT [37] or Encryptionizer [6]. The Encryptionizer only supports Microsoft SQL Server, with encryption with the standard algorithms. You can choose to encrypt the whole database, which is done on the file system level, and/or you can choose to encrypt data on column level. The column level encryption can be used as built in SQL functions for use in views, triggers or stored procedures etc. This makes the implementation very easy going. But all performance penalty that the encryption adds, will be on the server. The column level encryption supports encrypted access for users, groups, or roles. It is even possible to protect the data against the administrator. There is also support for encrypting log files. The Encryptionizer is possible to download and test free of charge. The other tool XP\_CRYPT has more or less the same features, and works in the same way, but also supports ORACLE databases.

### 5.2.1 EFS

The use of an Encrypted File System (EFS) is a possible way to encrypt the data in the database. The EFS encrypts the database file outside the database, on the file system level, making the encryption transparent to the database. It is an easy method to use, if the file system supports it, it is automatically done with the right configuration. Another benefit of using this technique compared to other encryption solution is that EFS would probably add least performance overhead. But it is not necessarily so, when using EFS, all data has to be encrypted; other techniques can offer a finer grained encryption. If for example only one column in one table needs to be encrypted, in a database where there are a hundred tables, the encryption of all data can give unnecessary performance loss.

The reason for the good performance with this technique is that when the encrypted database files needs to be accessed by the SQL Server, the operating system first decrypts it. So once the server is up and running there will probably not be a much extra performance penalty.

The main problem with this technique is that the SQL Server has to run as the same user as the user that has encrypted the file to be able to access it. Therefore this type of encryption is not sufficient for the purpose of hiding the data from the administrators. It is a good solution if you only need to protect the data against physical theft of the media that the data is stored on. Or an attacker who has got access to the system, but doesn't have the login for the right user account that stores the encryption key for the database file.

EFS could be used as a countermeasure, but not as a single solution for the database encryption. It can be used to add an extra layer of security. But it should only be used, if the security requirements are very high.

### 5.2.2 Column level encryption

The other solution which is more advanced is to encrypt the column values on the tables directly. Encrypt data before insertion and decryption the data after retrieval from the database. This can be done directly in the database or at the logical/business layer of the application. This method of encryption will very likely have a higher performance penalty than the above technique. It will also be a penalty in terms of usability. But when choosing which technique to use, it is much more important to look at the threat mitigation than the performance and the usability issues. It does not make sense to choose a technique that is not adequate for your purpose.

If the encryption/decryption is placed in the logical layer (which mean that it will be on the client application), there will be a lower performance impact on the database server, than if it would be in the database. This makes the application much more scaleable. Of course the client will get a lower performance, but this will be very

limited. For the security, it is also preferable, while the key and the plaintext never need to leave the client workstation. The limitation of the usability is something that is needed to be in mind. If data is stored encrypted, ad hoc reports and correctness checks of the data directly in the database would be impossible. These are things that normally are done, at least during development.

With Microsoft SQL Server 2000 there is not very good support for doing the encryption at the database level with build in encryption. If you want to do this, one of the above COTS should be used. In MySQL however there is support for the AES encryption directly on the table columns.

When encrypting the data columns, both the data types and the sizes need to be considered. It is possible to save the encrypted values in binary format or in string format. The block size, or the encryption mode, of the encryption algorithm and the string encoding decide how much extra space the encryption generates.

### 5.2.2.1 Algorithms

The choice of encryption algorithm for the database should be symmetric. Use of an asymmetric algorithm is not a good idea, because of the performance; as mentioned earlier (they also produce much more cipher data). The algorithms considered are implemented in the System.Security.Cryptography namespace (in the .NET framework version 1.1), which are shown in table 2.

Table 2. The encryption algorithms in the .NET framework.

	Rijndael	RC2	TripleDES
Block sizes	128, 192, 256	64	64
Key sizes	128, 192, 256	40, 128 (variable 4)	128, 192

The RC2 and TripleDES implementations are only wrapped unmanaged implementations from the CryptoAPI, whereas the Rijndael (also known as AES, the Advanced Encryption Standard) is the only fully managed code implementation. Therefore use of Rijndael is a good choice, because if an implementation is made in managed code, there are build in security mechanisms to protect against buffer overflows for example, which is a source of security vulnerabilities [30].

### 5.2.2.2 Data Size

With this encryption method there is a possible need for extra storage space as opposed to EFS. Normally the encryption algorithms works with blocks of data, and if the data to be encrypted is smaller than the block size, extra data (padding, for filling the block) will be needed, and if the last block is full and no padding is needed, an extra block will be added to the cipher data. You could use a stream cipher, or use OFB, CFB or counter mode, instead of a block encryption, but this is generally not a good idea if you want to use the same initialisation vector [25]. In .NET these modes are also not supported. Then, however the encrypted size would be the same as the plaintext size.

There is also more data added if you want to save the encrypted data as string format instead of in binary format. When using Base64 encoding, around 33 % extra data will be needed, according to [8]. Base 64 encoding is a way to represent binary data in ASCII string format; this is not possible with other string encoding schemas, because the binary data can contain unprintable characters.

When encrypting and storing small values, the size of the blocks will have a large impact on the size of the encrypted data. If for example the database is storing a normal integer value encrypted, it will be much larger than the normal size, which is normally 32 bits. With encryption this will have the size of the block. This means 4 times the space when having a 128 bit block size. If you then on top of this use a string format to store the encrypted value, the size will be 192 bits. If you instead would use

a 64 bit block size and save the value in binary format, the size would be 64 bits, or 1/3 of the size. The reason for using a string format instead of binary is the usability. During product development and maintenance it is easier to work with unencrypted data. To convert from unencrypted data in string format (which most data is stored as) to encrypted binary data, you need to change a lot in the database, the table declarations and also all stored procedures which include these columns needs to be edited. Use of base64 encoding over binary data is also recommended by [33], chap 14. Figure 7 shows how the encryption/decryption procedure works.

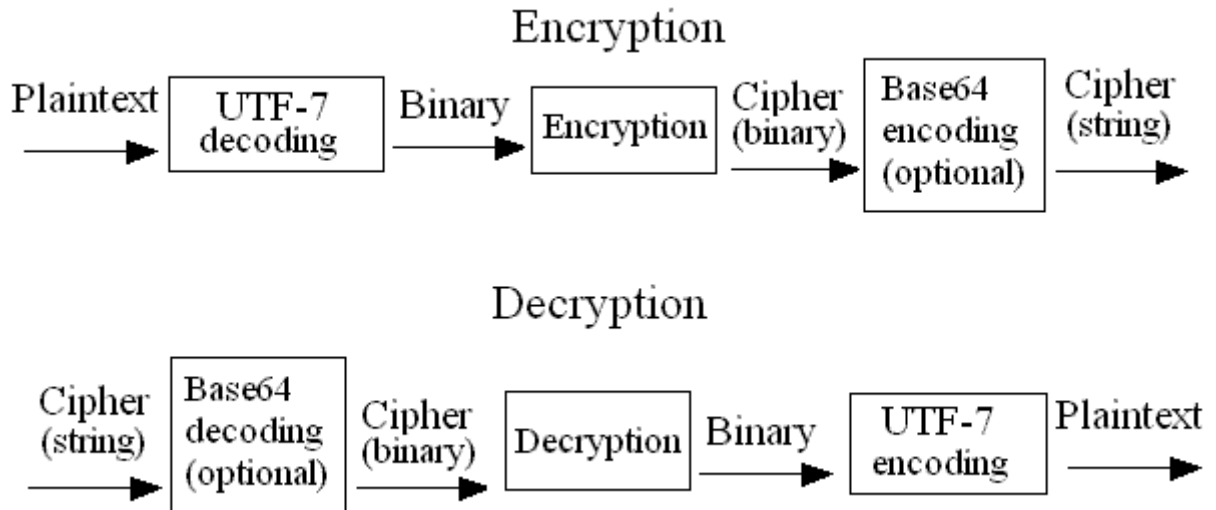


Figure 7. The encryption and decryption procedure.

The string encoding is also affecting the size. (The encoding schema used when encoding the plaintext string to its byte representation, and when decrypting from the decrypted byte format to the decrypted string format.) When using UTF-16 encoding, every character allocates two bytes. With UTF-8 encoding most characters (the ASCII characters) only takes one byte, but some take up to four, making UTF-8 a better choice. But the UTF-7 and ASCII encoding only allocates one byte per character, which makes them better. ASCII can however not be used, while it only supports English text characters. Therefore UTF-7 should be used.

The extra addition of data will also have an impact on the querying performance, because of the encryption (which is not the case with EFS). This will result in more computation for the SQL Server. And the data communication will also have a longer delay, because of the extra data.

The performance is of course dependent also on the encryption algorithm (and the implementation) and the encryption key size. Also which columns that are using encryption will of course affect. It is often not needed to encrypt all data fields. Only fields that are considered to be sensitive, that store patient information that is not publicly available, should be encrypted [29], (§11).

### 5.2.2.3 Compression

Before encryption is done, it can be a good idea to consider compressing the plaintext. Once data is encrypted it is not possible to compress it anymore. Encryption works by removing all patterns in the data, and compression can only be done if the data has patterns [28]. If the plaintext is large, it may be useful to compress it before the encryption takes place. It would speed up the performance of the communication to the database, when it is remote, by sending less data. This may be a good idea to decrease the performance degradation made by the encryption. However in BostonCare, columns in a database will probably not be large enough for gaining any noticeable difference, when using compression.

#### 5.2.2.4 Trailing nulls or spaces

Dependent on the specific database server, encrypted values could get problems with trailing space or null characters removal (which can be a part of the cipher text, if the cipher is not base64 encoded). In Microsoft SQL Server for instance, trailing nulls can be removed when saving data in a binary format (The ANSI\_PADDING configuration value defines this for the database, when it is on, there is on removal.) [2]. In MySQL there is the same risk but with trailing spaces when using a VARCHAR or CHAR data type, and there is no way to turn this feature off [15]. If this would happen, that a space or a null character would be removed from the cipher text, the decryption would fail. So therefore it is suggested to use the BLOB data type for encrypted columns.

#### 5.2.2.5 IV – Initialization vector

A limitation with this approach is that it is possible to interpret the encrypted data in the database. If the same value is encrypted in two different rows with the same key, they will have the same encrypted value as well. This gives an attacker the possibility gather information, even if he is not able to decrypt the data. To prevent this problem, it is possible to use an initialization vector, initiated with random values. An initialization vector is used by all encryption modes, except the ECB. The initialization vector works in the same way as CBC encryption, by XORing the encrypted result from the previous block together with the next block of plaintext before encrypting it. This results in a more secure encryption, in form of randomness, because the risk that two equal plaintexts will have the same cipher text is minimal. The initialization vector is used to XOR the first block, thereby the name. All of the algorithms, in the .NET framework listed above support usage of an initialization vector.

If an initialization vector is used for encryption, it will also be needed for the decryption. This means that for every value that is encrypted, a unique initialization vector is needed (there is no need for the IV to be secret however), to manage all of these; there will be an extra performance overhead and extra complexity added. To avoid this, there is a simpler alternative. Instead of using unique initialization vectors and keeping track of them, it is possible to use a salt value added to the beginning of the plaintext before the encryption and then use the same initialization vector all the time. The only thing to remember then is to know how many bytes the salt value has, which would be constant, to discard before reading the value. This solution will of course add some extra overhead, but much less than when using initialization vectors. If for example two random bytes are added, the extra overhead would be very small, and there would be  $2^{16} = 65536$  possible values, which would make the risk that two equal plaintexts would have the same encrypted value very small.

When salting the plaintext before the encryption, the security will increase, but in some cases this might not be very good for the performance. If a database search is made on such a column, this would mean that all rows first need to be decrypted before the search is made. This means that all rows have to be returned to the client application, where they then need to be decrypted and then the search can be made. If there are too many rows in the database, this problem could result in a very high performance penalty.

A way to solve this is to not use the salt, on those columns, where searching is needed. Thereby the search string can be encrypted and all values will have the same encryption as well. This makes it is possible to do the search in the database without first decrypt the column. There are however problems with this kind of searches. When doing the search on an encrypted field, it is only possible to find values that are exactly like the search string, including upper/lower case letters, as opposed to searching on unencrypted fields. This can however be taken care of, all data in the database could be stored as lowercase letters, and then when a search is done, make the search string to

lower case before it is encrypted. The security offered when not using salt, will be lower, but according to the law, the security will be high enough [29], (§11).

Other limitations that can be a problem is aggregate functions (functions that perform a calculation on a set of values and return a single value) on the encrypted columns will not work (AVG, MAX, MIN and SUM, for example). It is also not possible to return a sorted result set based on an encrypted column, directly in the database. This has to be made in the client application, after decryption. However these limitations should not introduce any real problems. Sorting for example is already implemented on record sets in the .NET framework, and the performance is not a problem. The aggregate functions can also be used in the client application.

#### 5.2.2.6 Sizes

The first thing to consider when defining the size of a column is if the column should be encrypted with a salt. Then the size will be the size needed for the actual data plus the salt, then rounded up to the next block size. If the value should be stored as string, then the size needs to be 33% bigger. For example (in bytes):

Datasize: 20

Saltsize: 2

Blocksize: 16

Total column size (binary): 32 bytes

Total column size (string): 48 bytes

#### 5.2.2.7 Performance tests

When choosing which algorithm to use, it is appropriate to test the performance of the different algorithm implementations. If the first choice algorithm is much slower than the others are, a re-evaluation can be appropriate. There are three test sets done in total. The first test set shows the encryption/decryption performance of the different algorithms. The second and the third test sets show the difference between several encryption settings and between encryption and no encryption. In appendix A more data from the tests can be seen, with the actual response times from the tests.

The performance values are displayed as percent. For every test, (every stack or column with the same colour) the algorithm with the best performance was given 100% and the others have percent values comparable to the time relation from the best. Therefore the different tests in the same test set are not comparable.

The tests have been performed on a single computer with the settings shown in table 3. Table 4 shows the algorithms used and their settings.

Table 3. The test environment.

Operating System	Microsoft WindowsXP professional, SP2
Programming Framework	Microsoft.NET version 1.1
Database Server	Microsoft SQL Server 2000 Developer Edition
Hardware	AMD Athlon 1.2 GHz with 512 MB RAM.

Table 4. Algorithms and settings for the tests.

	Rijndael	RC2	TripleDES
Block sizes	128	64	64
Key sizes	128, 256	128	128
Encryption mode	CBC	CBC	CBC

#### Test Set 1

This test shows the time it takes to encrypt and decrypt data with three different algorithms. Three different sizes of data that are tested are 4 bytes, 30 bytes and 10 Kilobytes. The first two are the most relevant sizes, because the size of the data used in

the real application would likely have sizes in that range. To be able to get measurable results, the 4 byte and the 30 byte tests were run 100 000 times and the 10 Kilobyte test was run 1000 times. This makes the accuracy of the tests more precise.

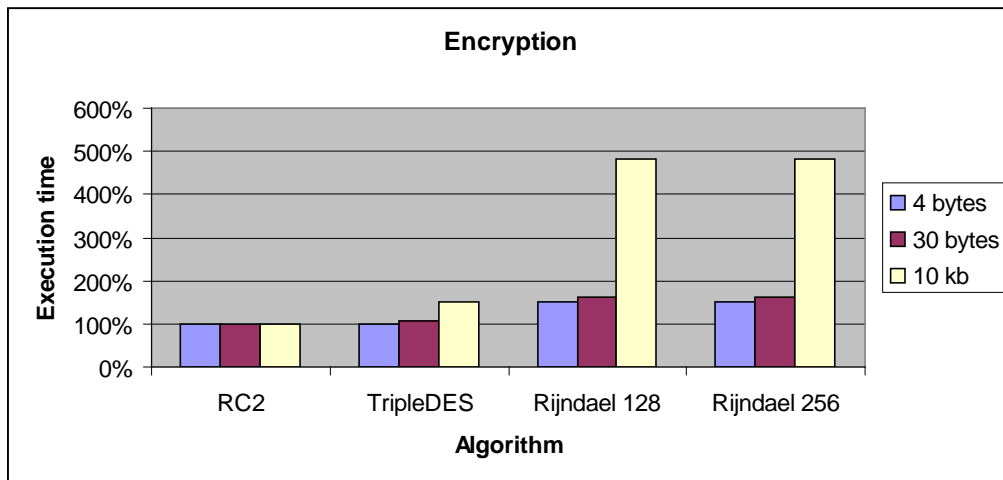


Figure 8. Test results from the encryption test.

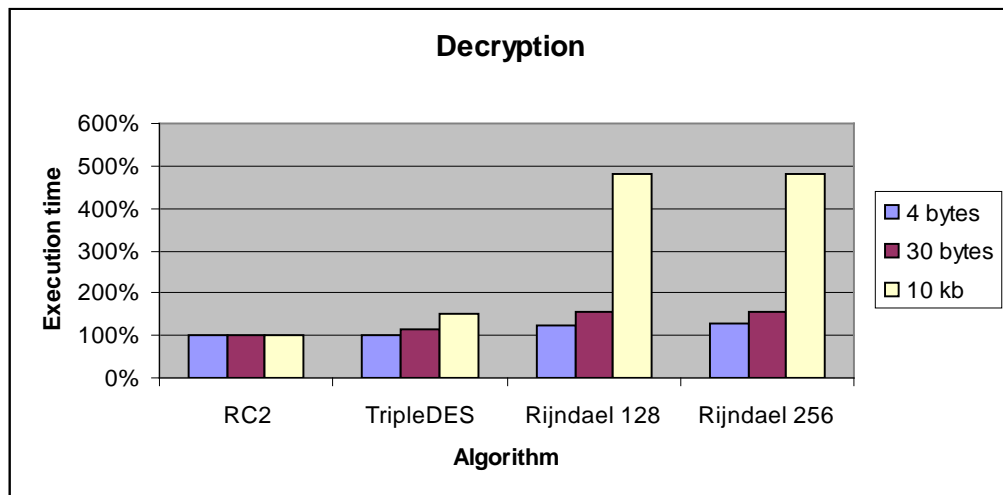


Figure 9. Test results from the decryption test.

Figure 8 shows the performance differences between the algorithms, when doing encryption. Figure 9 shows the performance differences between the algorithms, when doing decryption. As can be seen, both figures are almost identical, the encryption and the decryption have the same performance for the same algorithm. The first two algorithms, Rijndael 128 and Rijndael 256, refer to the same algorithm, but with different key sizes.

As can be seen in, Rijndael is the slowest algorithm and RC2 is the fastest. The larger the data, the more the performance differs, especially for Rijndael. Therefore if large data columns, like CAT scans, need to be encrypted, the TripleDES algorithm should be preferred over Rijndael. Another observation is that the key size is not affecting the performance at all.

#### Test Set 2

The second test set compares the algorithms together with a database to simulate a real application. It also shows the performance overhead created by using encryption compared to no encryption. The size of the encryption key is 128 bit for all algorithms.

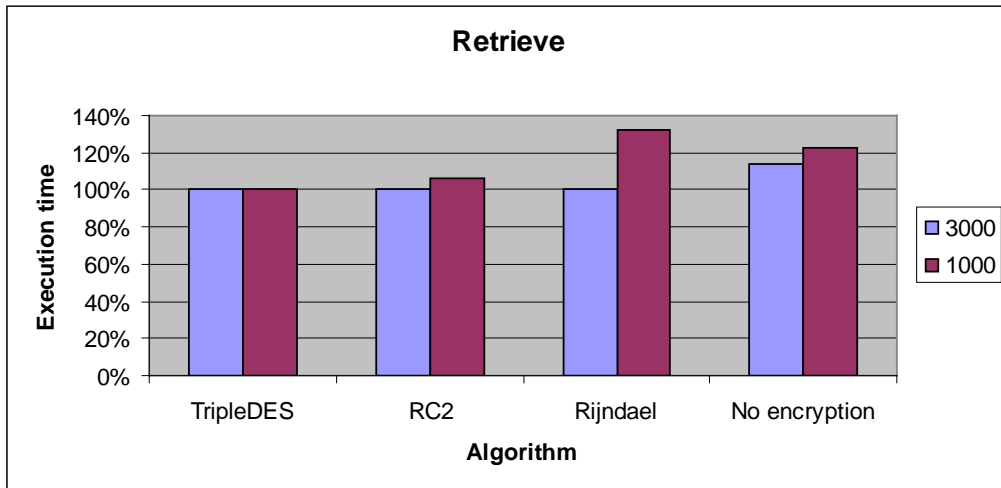


Figure 10. Test results from the retrieve test.

Figure 10 shows only the performance of querying the database and retrieve the result, without decrypting and displaying it. This is done to see how the increases of the data sizes degrade the performance. It is interesting to see that all encrypted retrievals are faster than the one without any encryption for the 3000 rows, even that more data is retrieved. A reason for this can be that the encrypted data is stored in a binary format as opposed to string format which the no encrypted table has. The Rijndael encryption is however slower for the 1000 rows retrieval. A reason for this can be that the Rijndael algorithm uses a larger block size, which of course can affect the performance negatively. A thing that is worth is that if the database server would be remote, the unencrypted retrieval would likely be faster, because less data needs to be transmitted.

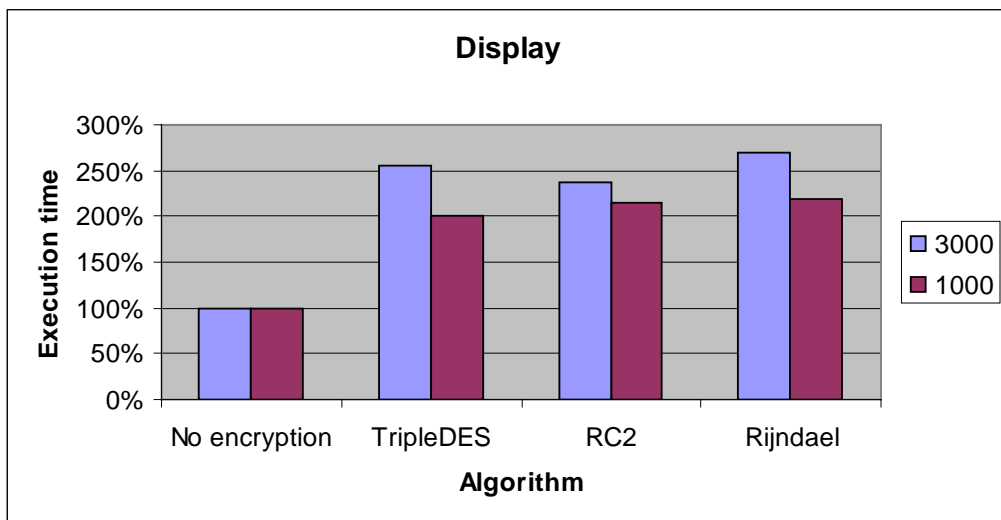


Figure 11. Test results from the display test.

Figure 11 shows the performance of decrypting and display the result in a Data Grid. The first test shows the performance when retrieving 3000 rows from total the total 18000 rows that exist in the table. The second test shows the performance, when retrieving 1000 rows. The setting without any encryption is as expected fastest. Then the different algorithms perform roughly as in the previous test set, where RC2 is the fastest, TripleDES second and Rijndael slowest. The TripleDES is however faster than the RC2 when querying 1000 rows.

### 5.2.2.8 The key management

The key management is an important part of the encryption. The level of security provided by the database encryption is very dependent on the key management. The purpose of the database encryption is to hide the data against the database administrators and for a possible attacker who gets access to the database server. For this reason there is only need for one encryption key. It is of course possible to use different keys for different levels of security, to use the encryption as access control. A database encryption scheme proposed by Samba Sesay et, al. [26] suggests this. In their solution there are several different keys used to strengthen the access control, one key for classified data, used for encrypted data that all authorised users should be able to access, and also private keys for every user, used for private encryption. The access control is however easier to achieve in other ways, with a less complex access control added in the database without the extra trouble of using different keys, and which will also result in a lower performance penalty. If using encryption for the access control, it is also not possible to read unauthorized information in the case of an emergency.

The safest place to save the encryption key and do the encryption/decryption is on the client application. This ensures that an attacker has to get access to both the client computer and the database server, to be able to read the data in the database.

### 5.2.2.9 Key Distribution

The key needs to be distributed on all clients and stored encrypted. When having only one key, it is possible for all users at all workstations where the client application is installed to encrypt and decrypt all data in the database. If there would be different keys for every user or every workstation, it would not be possible to read all the data inserted by another user, which is possible when using access control.

If there would be several organizations that would use the same database server, they would use different keys, making it impossible for them to read each others data, even if they would use the same database server. This protects the organizations from each other. A hacker from one organization could otherwise with some work get access to the others data, and be able to decrypt it, if they would use the same key.

The key distribution is done manually. To distribute the key, over the client workstations, a program is being used for both the generation and the encryption of the key. A password is first entered, then the program checks to make sure it is strong (20 characters long and check that all different types of characters are used, at least 2 numbers, 2 uppercase letters, 2 lowercase letters and 2 special characters). Then the password is hashed using SHA1 [17] (the hashed value is the key that will be used for the database encryption). Finally the hash value is encrypted, using DPAPI [5], and stored on disk. This method of key generation is of course not optimal, ideal would be to generate the key at random. This is however very hard to achieve. The Windows cryptoAPI has a random key generator, but according to [33], it has not been tested by the security community enough to be trusted, and other solutions should be used instead.

Therefore this approach is a good solution. It certifies a high level of security, if you assume that the different types of characters are used equally much, the entropy of each character would be around 6.6 bits, when there are 100 possible characters for use ( $2^{6.6} = 100$ ). That means that a 20-characters long password would produce around 128-bit entropy ( $6.6 * 20 = 132$ ) with a total random password, which is the same size as the encryption key. In practice the entropy is less, to create a password, people normally tend to have some structure. This can however easily be prevented to some extent by enforcing the use of all different types of characters, then the entropy will be reasonably high.

The usability problem that exists when having a 20-characters long password is of course much worse, than having a normal 8 characters long password. To try to remember such a password is next to impossible, especially if it is used very seldom,

only at the installation of new client applications. It is vital that this password is not forgotten, if it would be, it would be impossible to recover the encrypted data stored in the database (if all the keys at the client workstations are lost). Therefore this password needs to be written down and saved somewhere safe. This is sometime seen as a very bad idea, but according to Johansson, a security expert at Microsoft, there is nothing wrong with writing down your passwords [10]. If the password is written down there is no problem that it is 20 characters long, if it would be only 8 characters, it still would be needed to be saved somewhere. The other usability aspect is that it is troublesome to type in a 20 characters long password is also not very bad in this case. The password is only used at the installation, therefore the extra work can be considered to be ok, it is not worse than typing in a serial number which user already are used to.

The program helps the distribution of the database key and makes sure that the key is never stored unencrypted on disk. Using this program and the secret password is also a better alternative compared to having a backup secret key stored on disk. It helps the client administrator (not the same person as the database administrator) to recover the key in an easy way if it is lost.

To protect the stored key from people accessing the client workstations the DPAPI is used. With DPAPI you can encrypt data and as local machine store, making it possible for everybody that has access to the machine to decrypt the data. Or you can encrypt data as a user store, which only makes it possible for the current user to access the data. But for administration simplicity, it is preferable to use local machine store, and to strengthen the security, the file where the encrypted data is stored, you can use access control support from the operating system, to make sure only authorised users have access to the file. To make sure other applications running on the same user account cannot decrypt the data, an extra secret can be used. Prompting for passwords are always bad for the usability, so therefore there has to be a trade-off between the usability and the security, and it is fare to draw the line here if only authorised persons have access to the client computers.

## 5.3 Communication

The connection between the clients and the server can be realized in three different ways:

- Access over Internet
- Access via a leased line.
- Access via direct-dial with a modem.

By using the Internet, the cost will likely be lower than the other options. Calling over the conventional telephone network, with a modem, or leasing a line can be costly. Internet is more commonly used. It gives a better quality of service, because transmissions over the Internet have the possibility to travel over several routes. If the current route goes down, there will be other alternatives. But the problem with communicating over the Internet introduces security threats, as mentioned above. The communication is essential, therefore countermeasures should be implemented to protect against these threats. Otherwise it will be just a matter time, before the first successful attack breaks the system. The obvious countermeasure is to use encryption, which can protect against sequence number guessing attacks, eavesdropping and data tampering. The question is which protocol that should be used.

One protection could be to use a SSL encrypted database connection which is supported by both Microsoft SQL Server (Net Libraries) and MySQL, when doing remote connections. This is a very good countermeasure against eavesdropping and sequence number guessing, but it is limited against other types of attacks like information gathering, DoS and spoofing attacks. First of all, there is no authentication of the server, only the client can be authenticated, when using the Net Libraries

(Microsoft SQL Server). Both the client workstation and the Database server need to be directly accessible from the Internet. The communication will not be threatened, but there exists many other threats. Therefore stronger countermeasures are needed.

The best protection is to set up a VPN, which can give good protection against all of the existing threats. The only vulnerability existing when using a VPN is that there is no protection against internal threats, attackers on the LAN. Therefore if the VPN is set up between more hosts than the clients and the server of the system, there can be untrusted hosts on the LAN network. If that would be the case, the SSL encrypted database connection could be added, to countermeasure against eavesdroppers picking up login credentials sent during client authentication. However this should be avoided, because it will decrease the connection performance.

### 5.3.1 Comparisons

When implementing the VPN, it is good to know that there are different types of VPN:s, with different properties. There are many commercial solutions available for setting up a VPN.

PPTP is widely used, but the security is questionable. There have been a lot of flaws found in the Windows implementation of PPTP in a research article published 1998 by Schneier and Mudge [23], they did a security analysis showing several major problems. Since then there has been updates made by Microsoft, which addresses the major weaknesses, but offline password guessing attacks are still possible (when using MS-CHAP v2, as authentication method), with tools like L0ptrcrack according to [23]. It is however possible to use other authentication methods, such as EAP-TLS as mentioned earlier, which is considered to be more secure [35].

Even if PPTP is considered to be secure at the moment, it is better to use an open standard that has been developed in public. A public solution is likely to have gone through much more security analyses than a proprietary solution, according to Schneier [22]. A company can never use the same amount of resources that exist in the public research society.

The MPLS solution is a good option; it has very good performance and availability. It is very resistant against DoS attacks. However it is only considered to be a “trusted VPN” according to [34]. Which is not the same as a secure VPN; a trusted VPN only has dedicated circuits. To be secure, the provider of the circuits needs to be trusted, which may not be the case. Therefore if MPLS should be used, the database connection should use SSL encryption on top of the MPLS. MPLS is very suited for large VPN: s with many nodes and high transfer load. Therefore the MPLS solution probably is the most expensive alternative.

The most suitable solution for this VPN is the L2TP/IPSec combination, which is a public standard as mentioned above. IPSec is also the standard choice when implementing a VPN by companies like SUN, Microsoft and IBM, according to [11]. Depending on the provider, the encryption can be implemented in the hardware. If that is the case, it will be very fast, making it ok to use the double encryption (both for the database connection and for setting up the VPN). But if the VPN solution does the encryption in the software, it might slow down the communication. In that case, it is reasonable to skip the encryption for the VPN, because it would not add much extra security. When using Windows Server 2003, this solution is already available, this makes it a very suitable option.

Another option would be set up as a SSL VPN. There is however some differences compared to the above solution. On the negative side, a DoS attack would be easier. The SSL VPN has lower package drop performance, than IPSec. A faulty package is dropped earlier when using IPSec; the SSL has to process a package more before it can be dropped (it is higher up in the OSI model). There are also some advantages over the L2TP/IPSec solution. The configuration is generally much easier. There are for example no problems with NAT traversal, if IPSec is used with AH authentication; it

is not possible to traverse a NAT. There is an open-source SSL VPN called OpenSSL [18], which is free to use. The main advantages of OpenSSL are the price and the portability. OpenSSL runs on all major operating systems. So if the risk of a DoS attack is small, a SSL VPN can be a good choice.

## **5.4 Client Application**

In the client application there are several countermeasures that should be implemented.

### **5.4.1 SQL injection**

For protection against the SQL injections, there are several prevention mechanisms that can be done. According to [21], use of stored procedures should be used for all commands to the database; this will be used, also for other reason like performance and better access control.

### **5.4.2 Access control**

There has to be access control mechanisms also in the client application. As in the database, this should be achieved with authentication before any system usage.

The access control should also be set to authorize all access using least privilege, as in the database. The users should only be able to access what they are explicitly allowed to.

#### **5.4.2.1 Authentication**

For the authentication at the client application, the same credentials as used in the database are sufficient. There are other possibilities, smart cards or biometrical identification for example. One of these alternatives could of course be added, to give extra strength to the authentication. But, the organizations where BostonCare will be installed are considered to be relatively safe and an intruder will not likely get access to the system. Therefore considering both the cost and the difficulty of the implementation and the management of an extra system, would be to worth the added security. Strong passwords that are changed regularly are therefore enough.

#### **5.4.2.2 Authorization**

The authorization in the client application is the actual restriction on what a specific user can do with the system. Here there has to be capability restriction on what the users can do, in the same way as in the database. Doctors may have the capabilities to write patient diagnoses, administrators may have the possibility to create new user accounts for example. To control this, information about the users' capabilities is stored in the database, where the user accounts already are.

### **5.4.3 Password safety**

There needs to be a password policy. All users should be informed about the risks. The password needs to be kept secret, do not give it to anybody, even if they claim they are an administrator, do not store it where somebody could find it, and make sure no one looks when the password is entered. There should be a check for strong passwords, meaning at least eight characters, use of small letters, big letters, numbers and special characters. Do not show the usernames in the login GUI. If the attacker do not know the names of the users it is harder to crack the passwords. Also do not give more info than necessary at a fail login, only that it did not work. Do account lockout. A user should only have three login tries, then 30 min lockout, to prevent brute force cracking. This prevents attackers from cracking the password by doing multiple login attempts (also called dictionary attack). The negative with this feature is that an attacker can lockout the real users in this way, but this is only possible if he has their username.

#### 5.4.4 Session control

To minimize the risk that an attacker can get access to the system after an authorized user has logged in, there needs to be session management with automatic logout. If it is possible that a user can forget to logout or close the application, before leaving the computer, it will happen. In [1] they write about a situation, where a patient accesses a client application and alters the prescription data with murderous intent. This risk is easily mitigated by timer that keeps track of inactivity, after a specified time it stops the session. This feature is obviously a usability concern, if the timer is set too low, the users will be annoyed. If it is set too high, it would not increase the security. Around 15 minutes is a reasonable trade-off.

#### 5.4.5 Configuration management

To minimize the risk of possible attack surface, several things should be done. Make sure the client workstations are safe. Use anti virus program and a spy ware detection program. Make frequent software updates. Harden weak default settings and disable unused communication protocols and services. Use a firewall if connected to the internet. Force strong passwords for all user accounts on the operating system.

#### 5.4.6 Buffer Overflows

The BostonCare system is developed in .NET. One benefit from this is that all code is run in managed mode. As mentioned earlier this helps to protect against this type of threats with build-in security mechanisms.

This gives the program the possibility to safeguard against buffer overflows. However it is possible to include and run unmanaged code in a .NET application, to safeguard against buffer overflows this should be avoided as much as possible.

## 6 PROTOTYPE SYSTEM

### 6.1 Description

A prototype system was implemented with some of the suggested countermeasures. It uses database encryption, logging and access control. It is a healthcare management system, a pre-version of the BostonCare system. Development was done with Microsoft .NET and framework version 1.1. The same database server is used as in the previous tests. Figure 12 shows a screenshot of how the client application looks like.

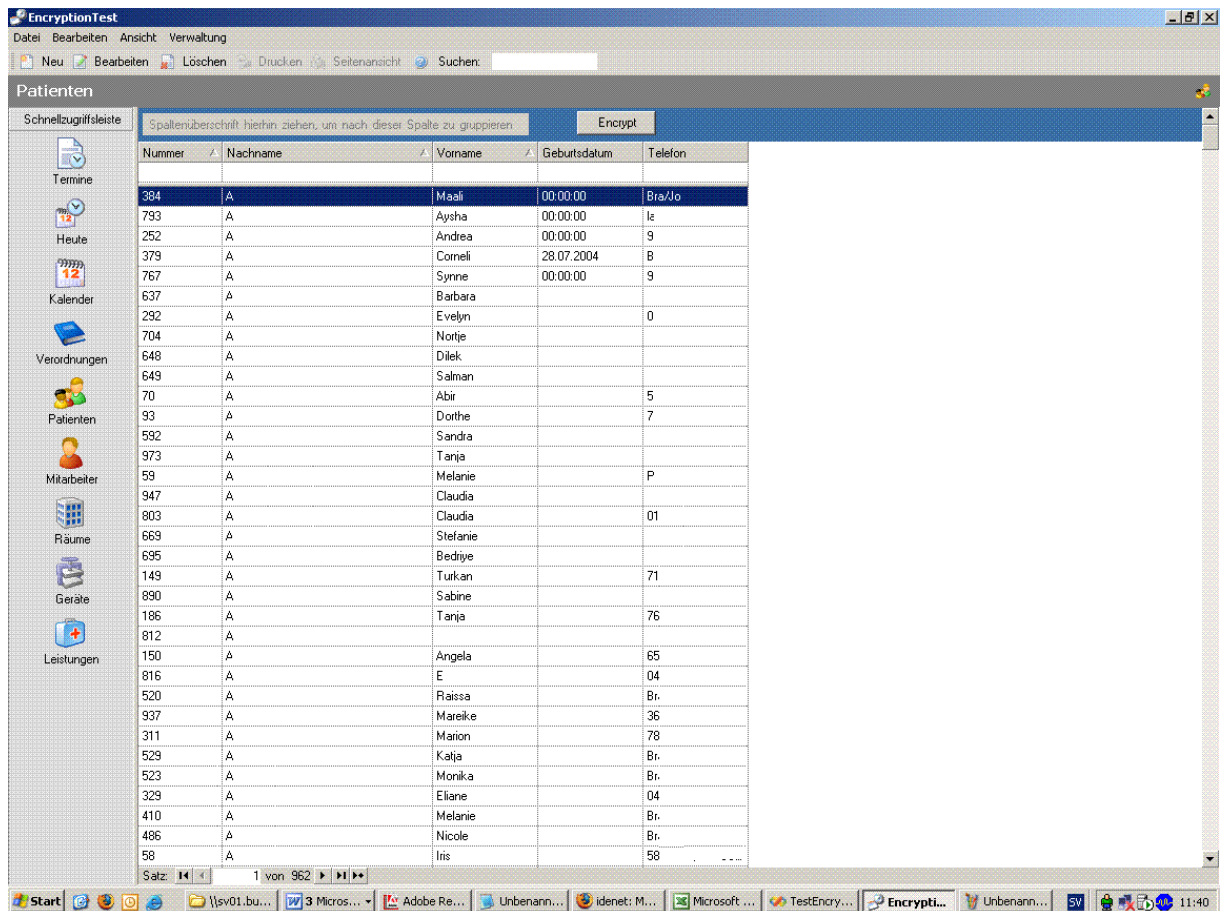


Figure 12. GUI of the prototype system.

The database encryption was implemented at the column level, as described earlier, with encryption and decryption taking part on the client side. Different settings were used and tested, which are described in the test section. The logging support was as recommended self implemented directly in the stored procedures. The access control was implemented both with authentication and authorization support. The authentication uses SQL Security with unique user accounts for all users. The user capabilities were then restricted to only the stored procedures in the database. Implementation of the ACL's was then also made, so that users only could access the patients that they were allowed to access.

This system was used to test the overall system performance and response times. This was done to make sure the performance degradation of especially the database encryption is not too high in a real application. The performance has to be acceptable

to the end-users. If the solution/algorithm chosen is too slow, it is important to find that out and re-evaluate a new faster solution.

## 6.2 Tests

The prototype test shows the difference between no encryption, Rijndael (with ECB and with CBC mode) and TripleDES algorithms. The test retrieves 10384 rows of data from the database and displays it in the application. The columns retrieved for the patients were the patient number, the name, the last name, the birth date and the phone number (which can be seen in figure 12; however some data is deleted because of confidentiality reasons). The encrypted columns were the name, the last name and the phone number. The last name was encrypted with salt and the other columns were not. All tests except specified used base64 strings when storing values in the database. All measurements are mean values from 100 tests. The test was made with the same settings as the earlier tests.

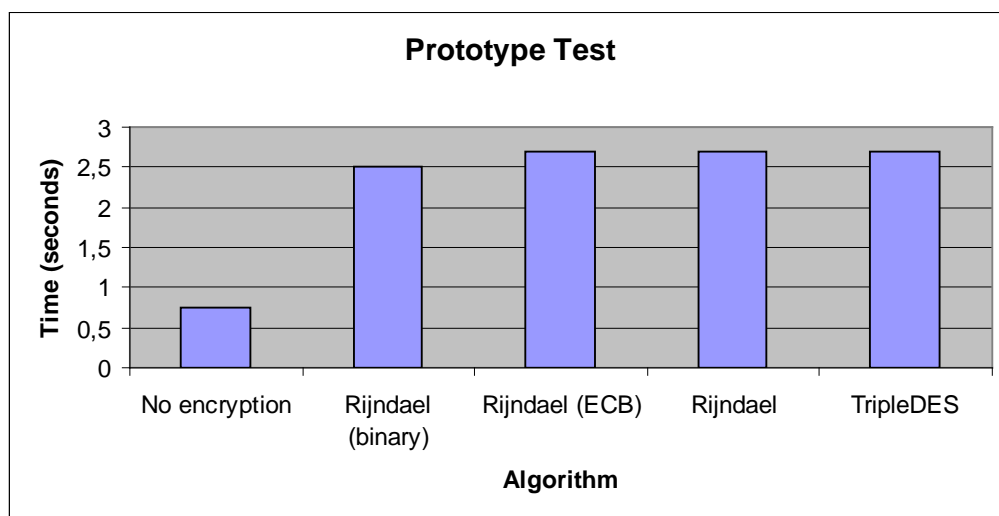


Figure 13. Test results from the prototype test.

### 6.2.1 Test Results

As figure 13 shows, there is no measurable performance difference between the different algorithms and the different algorithm settings. If the data columns that were encrypted would be larger, there might be a difference, for example if CAT scans would be stored in the database. Then the Rijndael algorithm probably would be much slower than the TripleDES. There is however a very large difference between encryption and no encryption. The difference between binary and string format is not very large. If the database would be remote, this would probably be more noticeable because of the extra data needed when using string format, however this kind of test was never done. Other tests made, are view, update and inserts of single patient records, to test performance of normal usage. These tests show no noticeable difference between encryption and no encryption.

The data retrieval that was used in the test is a very large set. For the BostonCare software, this will never be needed for normal usage. However this test shows the performance in an extreme situation. It shows that encryption is a large overhead, compared to no encryption, but the performance degradation is still reasonable.

As mentioned before, the relative difference between the different settings may vary some (the last three columns would be even slower), if the database would be remote, but the difference would probably not be very high.

## 7 CONCLUSIONS

The goal of this study was to analyse the security of BostonCare and identify security threats and law requirements present in a distributed system on the internet. Secondly, we aimed to identify security countermeasures and evaluate what best should be implemented to safeguard the system.

The work was initiated with identifying and investigating the State Act for the Protection of Personal Data in Schleswig-Holstein. A set of relevant law requirements were identified that needs to be implemented.

The law investigation was followed by a system analysis in terms of security threats. In the security analysis the system was divided up into three different assets, the database server, the communication and the client application. The assets were then separately analysed.

Security technologies have been studied to find out which possible countermeasures exist to secure the different assets against the identified threats. The focus has been on cryptography with symmetric algorithms and common secure communication protocols.

Several countermeasures have been suggested and compared against each other. Quality attributes are not uncommonly contradicting each other and therefore these countermeasures were compared in the aspect of performance, flexibility, usability, scalability, and cost aspects in regards to the level of security they gave.

The compared countermeasures included some symmetric algorithms: Rijndael, TripleDES and RC2. The studies show that Rijndael should be used, if the data to encrypt is relatively small, otherwise the TripleDES algorithm is a better alternative. A set of communication protocols were also compared, these were PPTP, L2TP, MPLS, IPSec and SSL. Suggestions are given to use a L2TP/IPSec combination, but the other alternatives are also possible to use.

A prototype system has been implemented as a proof-of-concept with some of the suggested security countermeasures. Database encryption, logging and access control was included. The system was then used to show and test the overall system performance and response times. The result from the prototype testing proved to be acceptable.

Areas of further investigation are for example encrypted communication. It could be interesting to measure the performance loss when using an encrypted communication to the database. There are different possibilities for the communication, for example hardware or software implementation of the encryption/decryption module. Different protocols may also likely have different impact on the performance, which could be tested and compared.

## **ACKNOWLEDGEMENTS**

First I would like to thank Buchner & Partner for the opportunity to work with them. I would also like to thank the following persons for their great help.

Jan Gerle, external supervisor at Buchner & Partner, for giving me advice and support during the course of this thesis.

Daniel Starke, external supervisor at Buchner & Partner, for the practical and the technical assistance he has given.

Håkan Grahn, university advisor at Blekinge Institute of Technology, for keeping me in the right direction and for assisting me in the writing of a scientific report.

## REFERENCES

1. Ross Andersson. "Security in Clinical Information Systems". Last visited 27 Jul. 05  
<http://www.cl.cam.ac.uk/users/rja14/policy11/policy11.html>
2. ANSI padding. Last visited: 27 Jul -05  
[http://msdn.microsoft.com/library/default.asp?url=/library/en-us/tsqlref/ts\\_set-set\\_2uw7.asp](http://msdn.microsoft.com/library/default.asp?url=/library/en-us/tsqlref/ts_set-set_2uw7.asp)
3. CERT/CC. Last visited 27 Jul. 05  
<http://www.cert.org/>
4. J. De Clercq and O. Paridaens. "Scalability Implications of Virtual Private Networks". IEEE Communications Magazine. May 2002. page 151-158
5. DPAPI. Last visited: 27 Jul. 05  
<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnsecure/html/windataprotection-dpapi.asp>
6. Encryptionizer. Last visited: 27 Jul. 05  
[www.netlib.com](http://www.netlib.com)
7. S. Fischer-Hübner, A. Ott, "From a Formal Privacy Model to its Implementation", Proceedings of the 21st National Information Systems Security Conference, Arlington, VA, October 5-8, 1998
8. How base64 encoding works. Last visited: 27 Jul. 05  
<http://en.wikipedia.org/wiki/Base64>
9. IETF, The Internet Engineering Task Force. Last visited 27 Jul. 05  
<http://www.ietf.org/home.html>
10. Jesper Johansson: Jot down your passwords. Last visited: 27 Jul. 05  
[http://news.com.com/Microsoft+security+guru+Jot+down+your+passwords/2100-7355\\_3-5716590.html?tag=nefd.ac](http://news.com.com/Microsoft+security+guru+Jot+down+your+passwords/2100-7355_3-5716590.html?tag=nefd.ac)
11. V. Kasacavage. *Complete Book of Remote Access : Connectivity and Security*. Auerbach Publishers 2002.
12. Massey, J. L., "SAFER K-64: A Byte-Oriented Block Ciphering Algorithm", *Fast Software Encryption* (Ross Anderson, ed.), Proceedings of the Cambridge Security Workshop, Cambridge, U.K., December 9-11, 1993, pp. 1-17.
13. Xuejia Lai and James L. Massey, A Proposal for a New Block Encryption Standard, EUROCRYPT 1990, pp389-404
14. Microsoft SQL Server log viewers. Last visited 27 Jul. 05  
<http://www.lumigent.com/>  
<http://www.sqlpower.com/>  
<http://www.logpi.com/>  
<http://www.softtreetech.com/idbaudit.htm>
15. MySQL Reference Manual. Last visited: 27 Jul. 05  
<http://netmirror.org/mirror/mysql.com/doc/mysql/en/index.html>

16. National Institute of Standards and Technology (NIST). FIPS Publication 197-1: Advanced Encryption Standard. 2001.
17. National Institute of Standards and Technology (NIST). FIPS Publication 180-1: Secure Hash Standard. April 1994.
18. OpenVPN. Last visited: 27 Jul. 05  
[www.openssl.org](http://www.openssl.org)
19. R. Oppliger. *Internet and Intranet Security 2nd ed.* 2002. ISBN 1-58053-166-0
20. Colin Percival. Hyper-Threading Considered Harmful, Last visited: 27 Jul. 05  
<http://www.daemonology.net/hyperthreading-considered-harmful/>
21. J. Scambray & S. McClure. *HACKING EXPOSED Windows Server 2003*, 2003 ISBN: 0-07-223061-4
22. B. Schneier. *Secrets & Lies Digital Security in a Networked World.* 2000 ISBN 0-47125311-1
23. B. Schneier & Mudge. "Cryptanalysis of Microsoft's PPTP Authentication Extensions (MS-CHAPv2)". CQRE '99, Springer-Verlag, 1999, pp. 192-203.
24. B. Schneier, "Description of a New Variable-Length Key, 64-Bit Block Cipher (Blowfish)," Fast Software Encryption, Cambridge Security Workshop Proceedings (December 1993), Springer-Verlag, 1994, pp. 191-204.
25. B. Schneier. *Applied Cryptography 2<sup>nd</sup> ed.* John Wiley & Sons, 1996 ISBN 0-471-11709-9
26. S. Sesay et al, "A Secure Database Encryption Scheme", American Journal of Applied Sciences, 2004. pp 327-331
27. J. Shapiro. *SQL Server 2000 : The Complete Reference.* McGraw-Hill Osborne, 2003.
28. L. D. Stein. *Web Security, A Step-by-Step Reference Guide.* 2000, ISBN 0-201-63489-9
29. The data protection law in Schleswig-Holstein. Last visited: 27 Jul. 05  
[http://www.datenschutzzentrum.de/material/recht/ldsg-neu/ldsg-n\\_e.htm](http://www.datenschutzzentrum.de/material/recht/ldsg-neu/ldsg-n_e.htm)
30. The importance of 100% managed code. Last visited: 27 Jul. 05  
[http://www.datadirect.com/developer/net/managed\\_code/index.ssp](http://www.datadirect.com/developer/net/managed_code/index.ssp)
31. J. S.Tiller. *Technical Guide to IPSec Virtual Private Networks.* Auerbach Publishers 2000.
32. P. TIPPETT. "SECURITY FOR THE CXO - THE CRYPTO MYTH". Last visited: 27 Jul. 05 <http://www.netlib.com/crypto-myth.shtml>
33. J Viega, G McGraw. *Building secure software: how to avoid security problems the right way 2001*, ISBN 0-201-72152-X
34. Virtual Private Network Consortium. Last visited 27 Jul. 05  
[www.vpnc.org](http://www.vpnc.org)

35. Virtual Private Networking with Windows Server 2003: Overview. Last visited 27 Jul. 05

<http://www.microsoft.com/windowsserver2003/techinfo/overview/vpnover.mspx>

36. What is TLS/SSL? Last visited 27 Jul. 05

<http://www.microsoft.com/technet/prodtechnol/windowsserver2003/library/TechRef/ed5ae700-e05e-45ef-b536-45795dbb99a2.mspx>

37. XP\_CRYPT. Last visited: 27 Jul -05

[www.activecrypt.com](http://www.activecrypt.com)

## APPENDIX A TEST DATA

### Test 1

Table 1 & 2 shows the result from the encryption test, table 1 shows the time in second and table 2 shows the differences between the different algorithms and data sizes.

Table 1 – Encrypt

	Rijndael	Rijndael	RC2	TripleDES
Block size	128	128	64	64
Key size	128	256	128	128
4 B * 100 000	2,1	2,1	1,4	1,4
30 B * 100 000	3,1	3,1	1,9	2,0
10 KB * 1000	5,8	5,8	1,2	1,8

Table 2 – Comparison Encrypt

	Rijndael	Rijndael	RC2	TripleDES
Block size	128	128	64	64
Key size	128	256	128	128
4 B * 100 000	150%	150%	100%	100%
30 B * 100 000	163%	163%	100%	105%
10 KB * 1000	483%	483%	100%	150%

Table 3 & 4 shows the result from the decryption test, table 3 shows the time in second and table 4 shows the differences between the different algorithms and data sizes.

Table 3 – Decrypt

	Rijndael	Rijndael	RC2	TripleDES
Block size	128	128	64	64
Key size	128	256	128	128
4 B * 100 000	2,1	2,2	1,7	1,7
30 B * 100 000	3,1	3,1	2,0	2,3
10 KB * 1000	5,8	5,8	1,2	1,8

Table 4 – Comparison Decrypt

	Rijndael	Rijndael	RC2	TripleDES
Block size	128	128	64	64
Key size	128	256	128	128
4 B * 100 000	124%	129%	100%	100%
30 B * 100 000	155%	155%	100%	115%
10 KB * 1000	483%	483%	100%	150%

### Test 2

Table 5 shows the time in seconds it takes to query a table in the database and display the decrypted result in a Data Grid. Table 6 shows the relative performance.

Table 5 – Total time

rows found/total	RC2	Rijndael	TripleDES	No encryption
3000/18000	0,9	1	0,96	0,48

1000/18000	0,29	0,31	0,27	0,18
------------	------	------	------	------

Table 6 – Total compare

rows found/total	RC2	Rijndael	TripleDES	No encryption
3000/18000	188%	208%	200%	100%
1000/18000	161%	172%	150%	100%

Table 7 & 8 shows only the time it takes to query the database and retrieve the result, without decrypting and displaying it.

Table 7 – Retrieve.

rows found/total	RC2	Rijndael	TripleDES	No encryption
3000/18000	0,14	0,14	0,14	0,16
1000/18000	0,066	0,082	0,062	0,076

Table 8 – Retrieve compare

rows found/total	RC2	Rijndael	TripleDES	No encryption
3000/18000	100%	100%	100%	114%
1000/18000	106%	132%	100%	123%

The query used to retrieve the rows was made based on the LastName field. This is how it looked:

```
SELECT *
FROM a_table_view
WHERE LastName = searchString
```

### Database tables

#### The encrypted table

```
P_ID int PRIMARY KEY,
LastName varbinary(128),
FirstName varbinary(128),
Birthdate varbinary(128),
House varbinary(128),
Room varbinary(128),
Station varbinary(128),
Meeting datetime,
suser varchar(40) default suser_sname()
```

#### The unencrypted table

```
P_ID int PRIMARY KEY
LastName varchar(40),
FirstName varchar(40),
Birthdate datetime default getdate(),
House varchar(40),
Room varchar(40),
Station varchar(40),
Meeting datetime default getdate(),
suser varchar(40) default suser_sname()
```

## APPENDIX B GLOSSARY

- AES, Advanced Encryption Standard, also called Rijndael
- AH, Authentication header, (used in IPSec)
- API, Application programming interface
- ASP, Application Service Provider
- Blowfish, an encryption algorithm
- CBC, Cipher Block Changing mode
- CERT/CC, Computer Emergency Response Team/Coordination Center
- CHAP, Challenge-Handshake Authentication Protocol, (used in PPP)
- DES, Data Encryption Standard
- DNS, Domain name system
- DPAPI, Data Protection API
- EAP/TLS, Extensible Authentication Protocol, (used in PPP)
- ECB, Electronic codebook mode
- ESP, Encapsulating Security Payload. (used in IPSec)
- FQDN, Fully qualified domain name
- HMAC, Keyed-hash message authentication code
- IDEA, The International Data Encryption Algorithm
- IETF, Internet Engineering Task Force
- IPSec, Internet Protocol Security
- IV, Initialization Vector
- Kerberos, authentication protocol
- L2TP, Layer 2 Tunnelling Protocol
- LAN, Local area network
- MAC, message authentication code
- MD5, Message-Digest algorithm 5
- MPLS, Multiple Protocol Label Switching
- MS-CHAP, Microsoft Challenge-Handshake Authentication Protocol, (used in PPP)
- NAT, network address translator
- NIST, National Institute of Standards and Technology
- OSI, Open Systems Interconnection
- PAP, Password Authentication Protocol, (used in PPP)
- PGP, Pretty Good Privacy
- PPP, Point-to-Point Protocol
- PPTP, Point-to-Point Tunnelling Protocol
- RFC, Request for Comments (see IETF)
- SAFER, an encryption algorithm
- SHA1, Secure Hash Standard.
- SSL, Secure Socket Layer
- TCP, Transmission Control Protocol
- TLS, Transport Layer Security
- UDP, User Datagram Protocol
- VPN, Virtual private network