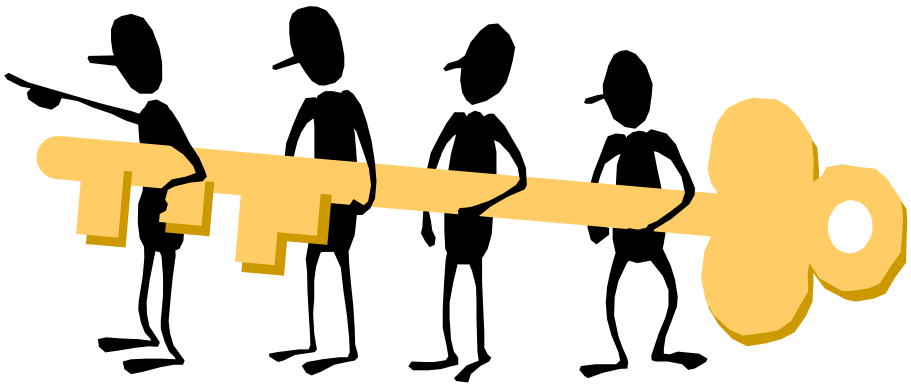
idasystems

Master Thesis

Security in Distributed Java Applications Based on Enterprise JavaBeans



Document title	Document name			Created
Security in Distributed Java Applications Based on Enterprise JavaBeans	Thesis 1.0.doc			31 January 2000
Document responsible	Document author	Version	Ref. no.	Last saved
	Hans Nilsson	1.0		9 June 2000

Ida Systems Ab, Box 576, SE-581 07 Linköping, Sweden, Tel: +46 (0)13 37 37 00, Fax: +46 (0)13 37 37 90, www.idasys.se

Abstract

The focus of this thesis was to identify what security problems exist in distributed software systems based on the Enterprise JavaBeans standard and to evaluate what services are available to solve these problems. The work was carried out at the software engineering company Ida Systems in Linköping.

The starting point for the investigation was the document and case management system PAX-NG developed at Ida Systems. We performed a survey of available security services that provide a solution for the different security problems that can be found in the software environment described above. The services were investigated from a theoretical perspective and narrowed down from six to four. Two of the services were implemented and evaluated in a test bench: WebLogic SSL and Java Cryptography Extension (JCE). The security service that was best suited for implementation in PAX-NG was the implementation of SSL (Secure Socket Layer) provided with the WebLogic application server.

The investigation was kept on a general level, so the results should apply for all distributed systems based on the Enterprise JavaBeans standard using the WebLogic application server.

Contents

1 Introduction

This chapter gives an introduction to the thesis and to the examination project that the thesis covers. It includes a project background, a general description of the purpose of the project, a presentation of the methodology and reading instructions for the thesis.

1.1 About the Thesis

This master thesis is the result of my final year examination project on the master's program "Computer Science and Engineering" at Linköping University. The project was done at the software engineering company *Ida Systems Ab* in Linköping and at the Department of Computer and Information Science, *Linköping University*. My supervisor at Ida Systems was Fredrik Öberg and my academic supervisors at the University were Professor Nahid Shahmehri and Assistant Professor Juha Takkinen. I extend my gratitude to all of them for assisting me in the writing of this thesis.

1.2 Background

This section gives a background description of the project and the host company Ida Systems.

1.2.1 Project Background

Ida Systems Ab (Ida) is a software engineering company in Linköping that develops document management systems and case management systems with workflow. They provide a software platform called PAX-Enterprise (PAX-E) which is designed to handle large volumes of information, possibly distributed geographically on different servers.

Ida is currently planning and designing the successor to PAX-E (denoted PAX-NG), which is a system based on a 3-tier architecture, just like PAX-E, with thin clients written in Java and server components also written in Java as Enterprise JavaBeans running on application servers. This final thesis evolves around PAX-NG.

1.2.2 Description of PAX-E

PAX-E is a software platform that provides services for document management and case management with workflow.

Document management systems are used for handling large volumes of documents (such as word documents, pdf documents, spreadsheets, and so on) possibly stored on multiple servers distributed geographically.

Case and workflow management systems are used for conceptualising the flow of cases through an organisation. The idea is that in every organisation the processes can be divided into several distinct sub-processes. For example in the editorial office of a news paper, the process of writing a news article can be described using the following sub-tasks: Writing an outline of the article, doing background research, writing the article, spell checking, and finally shipping the article to the editor for approval. The article can be seen as a case travelling through the different nodes of the workflow.

PAX-E is designed to work in a distributed environment. This enables a group of users to work with the same cases in a workflow and have access to the same documents even if they are located in different offices or even in different cities. To enable this, and to ensure scalability in the organisation, PAX-E is designed according to a 3-tier client/server architecture. This means that the application is divided into three separate layers: a client layer, an application layer and a data storage layer.

1.2.3 Description of PAX-NG

In PAX-E a lot of the business logic and the entire management of the GUI have been placed on the client side, making it a tedious task to port clients to new operating systems or to develop clients with customised user interfaces for specific customers. In an effort to solve these problems, Ida is planning and designing a new version of PAX called PAX-Next Generation (PAX-NG) based on so called “thin” platform-independent clients. The clients are called “thin” because all program logic has been moved to the server side. Basically the clients only contain logic necessary for communicating with the server.

This means that a PAX-NG client could, and probably will, be run as an applet in an ordinary web browser, such as Netscape or Internet Explorer. This enables users of a document and case management system to log on to their system on a web browser via an open network (such as the Internet).

The architecture of PAX-NG is still on a design level, but it is clear that the new platform will be based on Java. The clients are Java-based and the server applications are built as Enterprise JavaBeans (EJB) running on application servers.

1.2.4 Security Requirements in PAX-NG

The main difference between PAX-E and PAX-NG from a security perspective is that while PAX-E applications are run exclusively on private networks, thus limiting security issues, PAX-NG applications on the other hand will possibly be

run over a public network such as the Internet, which introduces a wide array of possible security hazards. Since the majority of Ida's customers are government agencies with high security requirements, this might be a big problem.

1.3 Purpose

The purpose of this thesis is to investigate and evaluate what security services exist to counter the security problems that are introduced in software systems running in open distributed environments. The term *security service* is used in this thesis to denote a service that provides a suite of algorithms for encryption, authentication, integrity checks, and key exchange. The focus will be on systems running in an Enterprise JavaBeans environment with the WebLogic application server used to host the JavaBeans. The work includes a recommendation on what security service would be suitable to use in PAX-NG.

The security aspect basically involves three sub-areas that are investigated [Stallings 1999]:

- **Confidentiality** (making sure that information exchanged between two parties is kept confidential and can not be intercepted by a third party).
- **Authentication** (verifying that the source of a communication really is the source that it claims to be).
- **Integrity** (making sure that information exchanged between two parties can not be altered by a third party).

1.4 Methodology

In this section the method used in the project for reaching an answer to the general question given in the former section is presented. The larger problem has been divided into several minor problem areas that can be focused on individually. The general research method has been visualised in Figure 1 as an action diagram.

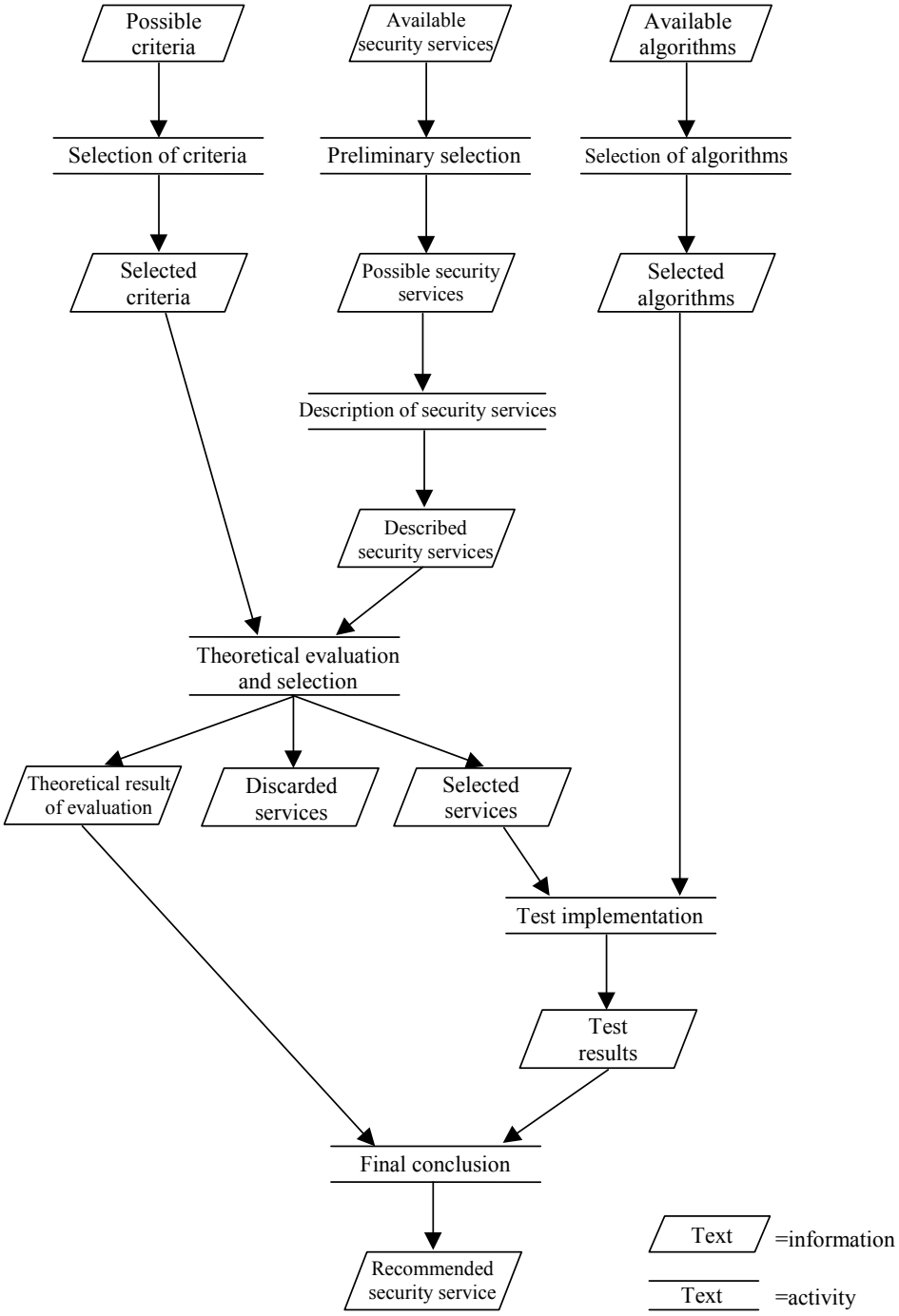


Figure 1 Process description of project method.

As seen in Figure 1, the project can be divided into three initial problem areas. One for selecting which criteria to use when selecting suitable security services, one for making a preliminary selection and writing a description of available security services and one for selecting specific algorithms that will be used in the testing of one specific security service later on. After that we perform a

theoretical evaluation of the services based on the selected criteria. This phase results in a set of selected services, a set of discarded services and a theoretical description of the services. The selected security services are implemented and tested in test benches for evaluating the performance of the selected services. Finally, the results from above are compiled and a recommendation of a security service suitable for PAX-NG is presented. Each sub-area presented above is discussed in the following paragraphs.

Selection of criteria

The overall purpose of the project is to make a survey of available security services, evaluate them and finally select the more interesting ones for a closer study. For this evaluation process to be as objective as possible, it is necessary to initially identify what criteria the services should be judged upon. These criteria will be selected after an extended study of the design specifications on PAX-NG.

Preliminary selection and description of services

After the survey of available security methods has taken place, it might be necessary to discard some services right away, before they have been formally evaluated according to the criterions mentioned above, because they seem just too inappropriate. It would be overkill to perform such a formal evaluation on all services. Descriptions of those services that were not discarded are presented.

Selection of algorithms

In the implementation phase of the project, one security service will be evaluated to study the performance of individual encryption and authentication algorithms (the service is tested in Test_Bench_JCE). This service supports a number of algorithms, and to simplify the test process a few of the more interesting algorithms are selected. To enable the selection of these algorithms we perform a literature study on the theory available on information security.

Theoretical evaluation and selection of services

The security services that passed the preliminary selection are now evaluated from a theoretical perspective based on the selected criteria. This evaluation results in two sets of selected and discarded criteria. It also results in a theoretical evaluation of the security services.

Test implementation

The security services that were selected are implemented and evaluated from a performance perspective. Performance in this case only refers to processing time. In the test phase the algorithms that were selected initially will be used in the testing of one of the selected security services. The test phase ends with an evaluation of the test results.

Final conclusion

The results from the theoretical and practical evaluation of the security services are compiled and a single service is recommended for use in PAX-NG.

1.5 Target Group

The target group for this thesis are people with at least a basic knowledge in the area of computer science and modern software development. The thesis is intended as a guidance for a person responsible for deciding what specific security service should be implemented in a large distributed software system based on the Enterprise JavaBeans standard.

1.6 Demarcation

The following areas are outside the scope of this thesis:

Digital signatures

Digital signatures are similar to message authentication, but while message authentication is used to protect two parties who exchange data from any third party fabricating or altering the data, digital signatures are used to protect the two parties from each other. Digital signatures provide a means of proving that a message transfer really took place between the two parties. No party should be able to deny it (this is often referred to as non-repudiation). Since this thesis is limited to the study of security issues in an environment where two *trusted* parties communicate over an unprotected network, the area of digital signatures will not be covered in the thesis.

Denial of service

Denial of service is an attack on security where the use of a service (such as an application server) is inhibited. This could be done by disabling the network or simply by overloading it with messages or requests for service. This threat belongs to the area of system or network administration and is not covered in the thesis.

Access control

Access control applies security policies that regulate what a specific user can and cannot do within a system. Access control ensures that users only can access resources for which they have been given permission [Monson-Haefel 1999]. This service is a part of the Enterprise JavaBeans concept. However, since it

belongs better to the area of system administration than to the area of network security it is not covered in this thesis.

1.7 Reading Instructions

These reading instructions give an overview of the structure of the thesis and describe which parts of the thesis that are essential reading for understanding the work done.

Chapter 1 is essential reading for understanding the background for and purpose with the project. This chapter also presents a problem formulation.

Chapters 2 presents background theory on distributed systems that to some extent might be familiar to the reader. Understanding the theory is essential for understanding the work done in the latter chapters of the thesis.

Chapter 3 presents background theory on security.

Chapter 4 presents the result of a survey of available security services. An overview of the services is given and the services that will be implemented in the test phase are selected. Since the selected services will be treated further in Chapter 5, it is necessary for the reader to be familiar with them.

Chapter 5 covers the test phase. The design of the test benches that were used is presented together with test results and an evaluation of the test results. The chapter is essential reading since the testing and evaluation of selected security services is an essential part of the project.

Chapter 6 presents the conclusions drawn in the research project. This is an essential part of the thesis and should be read.

Chapter 7 presents some recommendations on further work. This is intended for whoever might be interested, but is not essential reading.

Chapter 8 lists the references used in the thesis.

Appendix A gives a brief description on some of the terminology used in the thesis.

Appendix B presents a theory research on the encryption and authentication algorithms referred in the thesis. It is not essential reading, but might still be interesting for the reader.

2 Theory on Distributed Systems

This chapter gives the necessary background theory in the area of distributed systems needed to fully understand the work done in this project.

2.1 Distributed Environments

In the early years in the computer history, when big mainframe computers were used to carry out computations, computer programs used to be big and monolithic and all data processing was carried out on one computer. Later on, these monolithic programs were divided up into a client part and a server part running on separate machines according to the client/server model and even later on divided into a 3-layer architecture. These models are discussed below.

2.1.1 Traditional Client/Server Model

The traditional client/server model is a design model that became popular with the appearance of Local Area Networks (LAN) for PCs in the 1980s. The model enables a group of users working in a distributed local environment to share common resources, such as printers and databases (see Figure 2). This new model was a big step forward since the PCs finally came out of their isolation [Andrade et al. 1996].

The traditional client/server model is based on a 2-tier architecture with an application running on the client side and for example a DBMS running on the server side. Network security was not a big problem since the systems were running on private LANs.

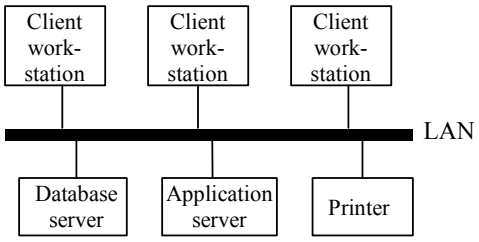


Figure 2 An example client/server environment.

2.1.2 3-Tier Architecture

The 3-tier architecture is a design philosophy that was introduced in the early 1980s for mini-computers. The 3-tier architecture is based on the traditional client/server model with the addition of an extra tier, a middle tier, as seen in Figure 3. Troy Kauffman gives the following description:

"The key characteristic of a 3-tier client/server architecture is the separation of a distributed computing environment into presentation, functionality and data processing components, such that there is a well-defined interface between each component, and the software used to implement each component can be replaced easily." [Kauffman 1997]

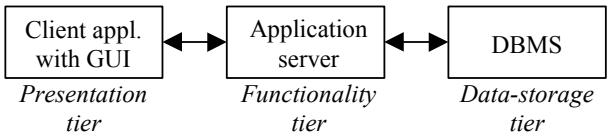


Figure 3 3-Tier Architecture.

Following is Kauffman's description of the three tiers:

- **Presentation tier:** This tier interfaces with the user. It consists of a thin client running the GUI and maintaining a connection with the middle-tier.
- **Functionality tier:** This is where the actual data processing occurs. The tier consists of applications running on a so-called application server.
- **Data-storage tier:** This is where the data is stored and managed. The tier consists of a database management system.

Benefits

A 3-tier architecture has many benefits over the 2-tier architecture [Kauffman 1997]. One of the most obvious benefits is that system administration is less complex since the applications can be centrally managed on a server. All business logic is running on a central application server and only a client maintaining the GUI is distributed to the users. For this reason, new versions of an application need not be distributed to the users.

The architecture guarantees excellent scalability since application components can be distributed on many servers. When a system grows out of itself, it is quite easy to install a new server to increase overall performance by load balancing The new server can then work side by side with the old servers [Thomas 1998].

These factors result in savings in system administration costs and software development costs. It also results in an increased level of security.

Security

It is easier to maintain a high security profile in a 3-tier architecture, since the system is divided into a few clearly separated layers where the communication between the layers can be easily monitored and secured via for example encryption and authentication. Mission critical data can be kept safe at a secure level behind a firewall. Authentication when accessing the data-storage is also

easier to achieve because only a few application servers need to be identified by the data-storage server [Monson-Haefel 1999].

The future for 3-tier architecture

The reason why the traditional 2-tier client/server model became so widespread, is probably because of the high quality of the tools that were designed to ease splitting the client from the data storage; tools such as DBMSs, remote SQL, ODBC, and so on. Over the past few years, the development of application servers and related software has accelerated. Since application servers are to the new functionality tier what a DBMSs are to the data storage tier, this should indicate a prosperous future for the 3-tier model as well [Monson-Haefel 1999].

2.2 Component Transaction Monitors

Component Transaction Monitors (CTM) play an important roll in this project and are discussed below. CTMs can be seen as a hybrid of TP monitors and ORBs, which are also discussed below.

2.2.1 TP Monitors

TP monitors (transaction processing monitors) have been around for some 30 years and provide reliable server platforms for distributed mission critical applications. Originally TP monitors were used as operating systems for large distributed business systems written in COBOL where the business logic was running on a central mainframe. Today, TP monitors such as Tuxedo from BEA Systems are still in use for managing the functionality tier in 3-tier systems. Vogel and colleagues [1999] give the following definition of a TP monitor:

> *"An operating system that specialises in creation, execution and management of transaction processing applications."*

To call the TP monitor an operating system might sound strange, but it gives a good description of what it actually does. The TP monitor is a resource manager that makes the best use of available resources, such as network and database resources. This resource management is needed to avoid overloading the system when the number of requests from clients becomes large.

The business logic in TP monitors is made up of procedural applications that are accessed over network through remote procedure calls (RPC).

2.2.2 Object Request Brokers (ORB)

TP monitors have proven very useful for procedural-based applications. After the advent of object-oriented programming, systems for distributed object technologies were developed, such as Java RMI and CORBA. These systems are

called object request brokers (ORBs) because they provide a communications backbone for objects. It should be stated that the communication infrastructure in CORBA is more elaborate then in Java RMI, but they both operate according to the same principle.

As opposed to TP monitors, ORBs can not be called operating systems, because they do not provide services for handling concurrency, transactions, security, persistence, resource management, and fault tolerance. All these services have to be implemented in the applications by each and every application programmer, which has proven to be an enormous task.

2.2.3 Component Transaction Monitors (CTM)

Component transaction monitors (CTM) are a hybrid of TP monitors and ORBs. They combine the best from two worlds and end up with a platform for distributed object-based applications that provide services for concurrency, transactions, resource management, fault tolerance, persistence and security [Monson-Haefel 1999].

TP monitors have been around for a long time so the technology behind them is rock solid. Inheriting this technology makes CTMs suitable for running mission-critical systems.

CTMs are often referred to as *application servers*, and this term will be used in this thesis.

2.3 Java RMI

Java RMI is a standard developed by Sun Microsystems for distributed computing in a Java environment [Sun/1 1999]. At the most basic level, RMI is Java's remote procedure call mechanism (RPC), in that it enables a client application to execute program code residing on a server. However, whereas the RPC mechanisms only allow the client to make subroutine calls on the server, RMI introduces object-oriented features. RMI can pass full objects as arguments and return values instead of just predefined data types.

RMI uses a stub-skeleton mechanism for providing an easy way for a client application to call a method in a remote server object. When a client application wants to invoke a method of a remote server object, it calls a local surrogate object called a stub. The *stub* is an empty object with an interface identical to the remote object that handles all network communication between client and server. For the client application it is just as if the remote object was running locally. A corresponding surrogate object on the server side called a *skeleton* has a similar purpose. It relays the call from the stub in such a way, that the server object perceives it as being a local call. It should be said that the stub and the skeleton

must have been compiled and distributed before the remote method invocation can take place.

To enable the client application to locate a remote object, an RMI registry on the server is maintained. Stubs use the RMI registry to lookup and get reference to remote objects. Before the lookup take place, the server object must therefore have registered itself.

2.4 WebLogic Application Server

The WebLogic application server developed by BEA Systems is one of the leading component transaction monitors on the market. It provides an integrated platform for distributed 3-tier computing. WebLogic supports a large variety of server component standards, such as Enterprise JavaBeans, RMI, CORBA and Tuxedo components. The WebLogic server provides a platform for storing and managing these server components, as discussed in section 2.2.3. It also adds services for security management (such as authentication and access control) and database connectivity [BEA/1 1999].

The server is built to handle large volumes of transactions and to be easily scalable. This is accomplished by using transactional techniques such as resource pooling and caching. Resource pooling is a technique for making the best use of database resources. Instead of letting each application connect to databases on their own initiative, each database transaction is relayed via a resource pool, managed by the server. The pool manages only a few database connections and handles transactions by bundling them into larger database requests.

Figure 4 below gives a schematic overview of the WebLogic application server model. The diagram exemplifies storage of two forms of server components: RMI classes and Enterprise JavaBeans. Java clients use JNDI (Java Naming and Directory Interface) to gain access to server components. JNDI is a registry maintained by the server that keeps track of all components stored on the server.

When connecting to the WebLogic server, Java clients establish a connection using BEA's RichSocket™ protocol, which multiplexes various protocols over a single network socket connection. A RichSocket™ connection can carry all kinds of WebLogic traffic simultaneously; for example JDBC traffic, RMI traffic and WebLogic Event traffic [BEA/2 1999].

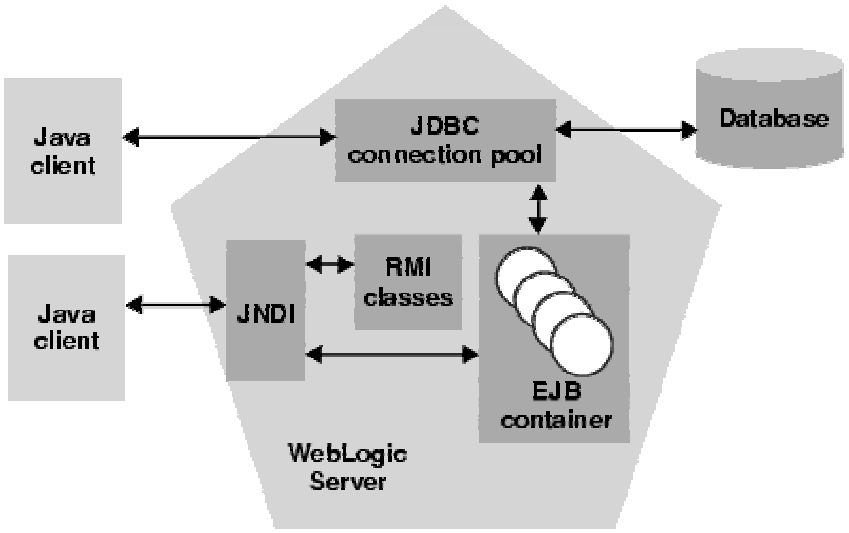


Figure 4 WebLogic application server model. From [BEA/1 1999].

2.5 Enterprise JavaBeans

Enterprise JavaBeans (EJB) is a specification from Sun Microsystems that sets forth a distributed component model for the Java programming language. EJBs simplify the process of setting up a distributed system.

2.5.1 Overview

EJBs can be seen as the latest technology abstraction in the Java family, as they provide an abstraction for component transaction monitors (CTM). As discussed in section 2.2.3, CTMs are servers that are specialised at managing so-called server-side components and managing services such as transactions, persistence and security. Server-side components can be seen as standardised containers placed on the server, containing the application code. Thus, a level of abstraction is achieved as the application programmer only needs to have knowledge about the development environment provided within the container, and the server software only needs to know how to store and manage the containers [Monson-Haefel 1999]. Because of this level of abstraction, EJBs can easily be moved from one CTM to another, as long as they both follow the component model standard specified by Sun. This is referred to as "Write Once, Run Anywhere™" portability [Thomas 1998].

2.5.2 Architectural Overview

The EJB architecture can basically be divided into three components: the enterprise beans, which are the very objects that contain application code, the

EJB containers that manage the beans and the EJB server which manages the containers [Monson-Haefel 1999].

Enterprise beans are program components that are stored in EJB containers and that follow a specified interface that simplifies the distribution of application code to servers. When an enterprise bean is deployed into the EJB container, two interfaces must be specified by the developer [Thomas 1998]. The first one is the *remote interface*, which intercepts all method calls from the client and implement transactions, persistence and security services for the bean. The second one is the *home interface*, which is accessible through JNDI and implements all lifecycle services for the bean, such as services for creating, finding and removing enterprise beans. See Figure 5 for an overview.

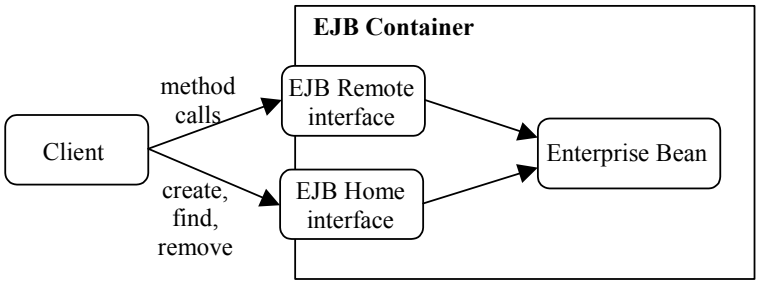


Figure 5 Functionality of the EJB Container.

Central to the EJB architecture is the EJB container, which manages the beans contained in it. For each bean the container is responsible for registering the object, maintaining the remote interface for the object, creating and destroying object instances, checking security for the object and co-ordinating distributed transactions [Thomas 1998]. The EJB containers are themselves maintained by the EJB server.

There exists two different kinds of enterprise beans used in different situations: entity beans and session beans.

2.5.3 Entity Beans

Entity beans are objects with a persistent representation. This means that they are stored in a database between sessions. To enable database storage, each entity bean must have a unique identity, called a primary key. Each entity bean must implement at least one find method, which is used by the client for locating beans. An example of an entity bean is an object that represents a banking account. The status of the bank account must be maintained between sessions, thus persistence is required. [Monson-Haefel 1999]

2.5.4 Session Beans

Session beans are objects that implement high-level services. As opposed to entity beans, session beans are not persistent. Instead a new bean is instantiated in every session. Session beans usually perform services on entity beans, but they can also operate in isolation. An example of a session bean is an object that provides services for depositing money and calculating interest on the banking account bean described above [Monson-Haefel 1999].

3 Theory on Security

This chapter presents the necessary background theory in the area of computer security needed to fully understand the work done in this project. First, however, we give a motivation for the need of security services.

3.1 Motivation

Security issues in public networks, such as the Internet, have become an increasing problem over the past years. As the use of Internet has increased, the number of security violations has increased correspondingly. This is demonstrated clearly in the statistics over security violations presented by the network security organisation CERT. As shown in *Table 1*, the number of security violations reported to CERT has more than doubled from 1998 to 1999 [CERT 2000]. Thus, it is necessary to implement adequate security services in open distributed environments.

Table 1 *Number of network security incidents reported to CERT.*

Year	1990	1991	1992	1993	1994	1995	1996	1997	1998	1999
Incidents	252	406	773	1,334	2,340	2,412	2,573	2,134	3,734	8,268

3.2 Security Threats and Counter Measures

According to Stallings [1999] attacks on the security of a computer system in a distributed environment can be divided into four different categories: interruption, interception, fabrication and modification. The first category involves the hindering of communication between two parties and is outside the scope of this thesis. The remaining three categories of security attacks are discussed below.

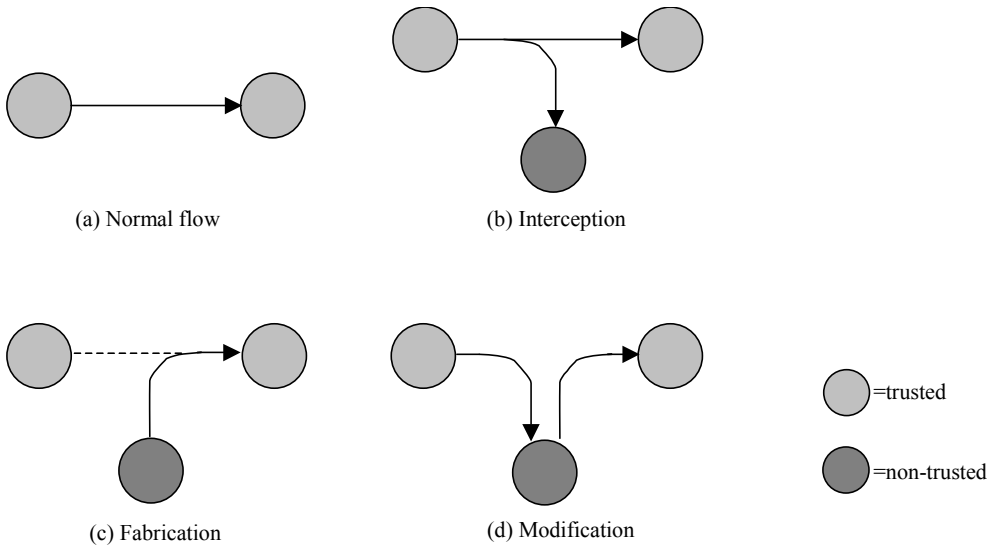


Figure 6 Principal security threats [Stallings 1999].

Figure 6 above gives a schematic presentation of the principal security threats. Figure 6 *(a)* displays the normal data flow from sender to intended receiver. The remaining parts of the figure shows the security threats discussed below:

- **Interception:** An unauthorised third party (represented in dark grey) gains access to transmitted data by wire-tapping the network. See Figure 6 *(b)*. This threat is discussed in section 3.2.1 below.
- **Fabrication:** An unauthorised party fabricates data and transmits it through the network using the identity of a trusted party so that the receiver believes that the source of the transmission is another than it really is. See Figure 6 *(c)*. This threat is discussed in section 3.2.2 below.
- **Modification:** An unauthorised party intercepts a message from a network and modifies it before reinserting it into the network. See Figure 6 *(d)*. This threat is discussed in section 3.2.3 below.

3.2.1 Confidentiality

To counter the security hazards that *interception* of messages introduces, confidentiality in the transmitted data is needed. This is accomplished by using cryptographic algorithms. Basically cryptographic algorithms can be divided into two distinct groups: symmetric-key algorithms and public-key algorithms [Stallings 1999]. However, all algorithms work according to the scheme presented in Figure 7. Plaintext is fed into the encryption algorithm where it is converted into ciphertext before it is transmitted to the recipient. At the recipient's side the ciphertext is converted back to plaintext via a decryption algorithm. Both the encryption and decryption algorithms take a key as input [Schneier 1994].

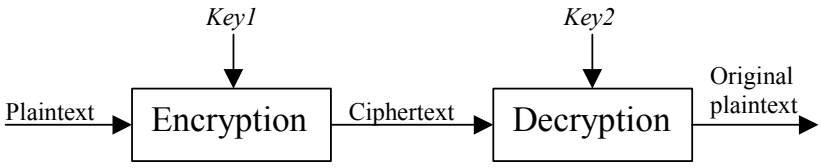


Figure 7 Encryption and decryption.

Mathematically the process can be described as follows

```
Ciphertext = E(Plaintext, key1)
Plaintext = D(Ciphertext, key2)
```

where *E* denotes the encryption algorithm and *D* denotes the decryption algorithm. For *E* and *D* holds that

```
Plaintext = D(E(Plaintext, key1), key2)
```

Symmetric-key algorithms

A symmetric-key algorithm is an encryption algorithm that requires that both the sender and the receiver of an encrypted message have access to the same key. Thus, in Figure 7 *Key1* and *Key2* would be identical [Schneier 1994]. There are generally no limitations on the design of symmetric-key algorithms, but in reality most symmetric-key algorithms are block ciphers. The opposite of block ciphers is stream ciphers. These two classes of symmetric-key algorithms are discussed in section 3.4.

Public-key algorithms

A public-key algorithm is an encryption algorithm that enables two parties to send encrypted messages to one another without sharing a common key. Each party maintains two keys - one public-key and one private key. When for example A want to send a secret message to B, he encrypts the massage using B's public-key. The message can then only be decrypted using B's private key. In Figure 7 *Key1* would be B's public-key and *Key2* would be B's private key. [Schneier 1994]

One-time Pad is the only fully secure method

According to [Schneier 1994] the only fully secure method of encryption is to use a one-time pad, that is, to use a key that meets the following three criteria:

- the key is completely random
- no part of the key is ever reused
- the key has the same length as the message that is to be encrypted.

The one-time pad algorithm is a symmetric-key algorithm, which means that the sender and the receiver of a message need to have access to the same key. Off

course, these criteria render the method more or less useless in real life. However, in situations where one beforehand knows that a limited amount of secret information will be transmitted from one point to another, and the information is of such a kind that it would be devastating if it became public, then this algorithm is useful. For this reason, the one-time pad algorithm is often called “the diplomatic encryption algorithm.”

3.2.2 Authentication

The authentication service is involved in assuring that a message is authentic and not *fabricated*. Authentic in this case means that the source of a message really is the source that it claims to be. This service involves two actions. First, when a connection is initiated the service must assure that the two participating parties are authentic. Second, the service must assure that the connection is not interfered in such a way that a third party can masquerade as one of the two legitimate parties for the purpose of unauthorised transmission or reception [Stallings 1999].

3.2.3 Integrity

The integrity service involves assuring that a message has not been *modified* during transfer. This is often done by calculating a checksum of the message (for example by using a hash function) and appending this checksum to the message which is to be transferred. The recipient can then recalculate the checksum on arrival and compare this with the checksum appended to the message. The checksum that is appended to the message must be encrypted or in some way depend of a secret key to prevent an unauthorised third party from changing it after interception [Stallings 1999].

3.3 Level of Security

According to Olovsson [1992] there exists no absolutely secure systems. Instead security can be measured on a continuous scale from 0 to 1 or from completely insecure to totally secure. A secure system would then be defined as a system where the intruder has to spend an unacceptable amount time or money in order to make an intrusion.

Increased security most often results in increased costs for the system. The cost for security is a combination of many factors, for example cost for decreased system performance, cost for increased system complexity, cost for decreased system usability, and increased maintenance cost. Thus, it is necessary to determine an optimal level of security for each system, where you combine the cost for the security measures and the expected gains from increased security. See Figure 8 for the cost/security graph.

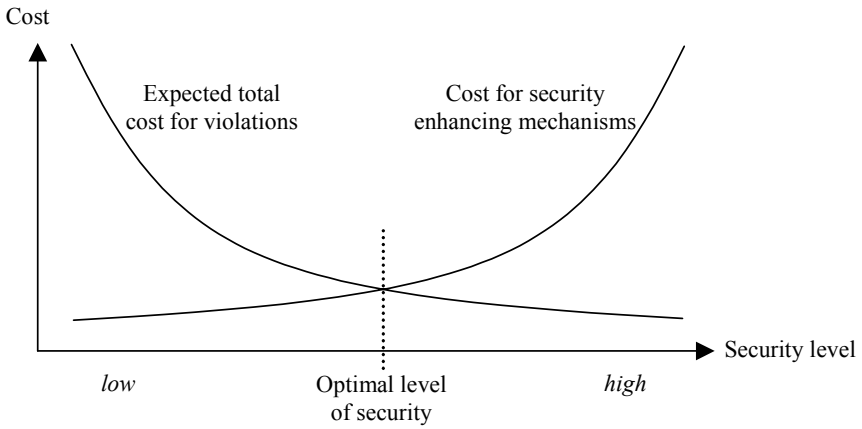


Figure 8 The cost/security graph. Based on [Olovsson 1992].

3.4 Symmetric-key Cryptography

As mentioned before in section 3.2.1, cryptography is a method used for providing confidentiality in data transfer. Symmetric-key cryptography is by far the biggest and most important area of cryptography. Symmetric-key algorithms are used when large amounts of data needs to be encrypted, as opposed to public-key cryptography which is used for encrypting very limited amounts of data such as session keys. In this chapter two kinds of symmetric-key algorithms will be discussed: block ciphers and stream ciphers.

3.4.1 Block Ciphers

Block ciphers are the most widely used type of symmetric-key algorithms. Block ciphers are a group of ciphers that encrypt data streams in blocks of for example 64 bits at a time. All block ciphers presented in this thesis are based on the Feistel network.

Structure of a Feistel Network

The Feistel network was invented by Horst Feistel in 1973 [Feistel 1973], but the structure of the cipher dates back to Claude Shannon's legendary paper on information security [Shannon 1949].

In a block cipher based on the Feistel network the data being encrypted in each block is split into two halves. Iteratively an encryption function F is then applied on one half of the data together with a sub-key in each round and the output from F is XORed with the other half of the data. After each round the data is swapped. See Figure 9 for an overview of the design of the Feistel network. In each round a new sub-key is calculated from a master key using a sub-key generation algorithm. This reuse of the master key is possible without weakening the

algorithm because the Feistel network is designed to create a so-called *avalanche effect* (i.e. a small change in the plaintext or the key produces a significant change in the ciphertext).

The Feistel cipher algorithm is reversible, so for decryption the algorithm is simple followed backwards.

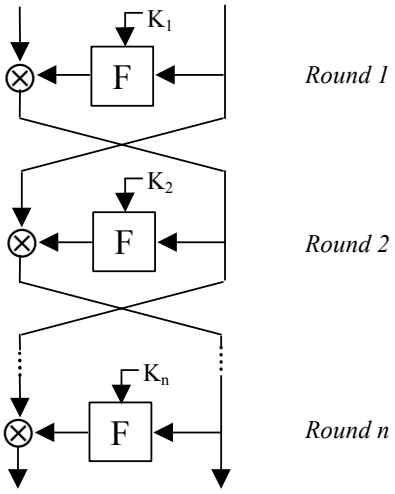


Figure 9 Principle design of a Feistel network.

Block Cipher Modes

Block ciphers can run in several different modes depending on if and how the different cipher blocks in a message are chained. Four modes are worth mentioning:

- **Electronic Codebook Mode** (ECB). In this mode each block is encrypted separately. The name comes from the fact that a specific block of input data is always encrypted into the same ciphertext (if the key is the same, that is). This means that each block is treated separately and that blocks of data in the middle of a message can be encrypted before the first blocks are encrypted. This independence between the blocks introduces an integrity problem in ECB since it is possible to cut-and-paste blocks between different messages encrypted with the same key.
- **Cipher Block Chaining** (CBC). To solve the "cut-and-paste" problem in ECB, CBC uses a chaining mechanism where the result from the encryption of one block is fed back into the encryption of the next block. Thus, the blocks of data in a message has to be encrypted linearly (from the first block to the last block).
- **Cipher Feedback Mode** (CFB). With CBC encryption cannot begin until a complete block of data is received. This is a problem in many network

applications, for example in terminals that must transmit one character at a time. CFB enables the usage of smaller encryption blocks (typically 8 bits).

- **Output Feedback** (OFB). This mode is a variant of CFB with minor internal changes.

Stallings [1999] gives the following list of typical application areas for each mode: **ECB** is useful in transmission of single values (such as keys), **CBC** is used for general purpose block-oriented transmission and authentication, **CFB** is used for stream-oriented transmission and authentication and **OFB** is used for transmission over noisy channels.

3.4.2 Stream Ciphers

A stream cipher is a cipher that handles messages of arbitrary size by encrypting individual bits in a stream. This avoids the need to accumulate data into a block before encryption, as is necessary in a block cipher. A conventional stream cipher is very simple. A random key sequence is used to encrypt a message bit by bit via an exclusive-OR operation. The ultimate stream cipher is the one-time pad algorithm, discussed in section 3.2.1, where each bit in the key is used only once. A more common approach is to use a random key generator, such as a Linear Feedback Shift Register, for generating a seemingly random key sequence from a shorter key [Fåk 1997].

3.5 Public-Key Cryptography

The advent of public-key cryptography was a big step forward in the field of secure communication over public networks. Stallings [1999] describes it as "the greatest and perhaps the only true revolution in the entire history of cryptography."

3.5.1 General Description

As mentioned earlier, public-key ciphers enable two parties to communicate securely without sharing a common key. This is accomplished by using so called *one-way functions*.

One-way functions

The notion of one-way functions is central to public-key ciphers. Bruce Schneier gives the following description:

> *"A one-way function is a function that is relatively easy to compute but significantly harder to undo or reverse. That is, given x it is easy to compute f(x), but given f(x) it is hard to compute x. In this context, 'hard' means, in effect, that it would take millions of years to compute the function*

even if all the computers in the world were assigned to the problem." [Schneier 1994, p. 27]

An example of the characteristics of one-way functions can be seen in $f(x) = x^2$. $f(x)$ is simple to compute, but the inverse, $f^{-1}(x) = \sqrt{x}$, is much harder.

A one-way function in itself is not very impressive - it can not be used in encryption. What good would it be to encrypt a message if you could not decrypt it! Instead in cryptography a special variant of one-way functions called *trap-door one-way functions* is used. A trap-door one-way function is a function that is easy to compute in one direction, and very difficult to compute backwards, unless you know the secret trap door. The trap door enables one to easily compute the function backwards.

Conceptual framework

Public-key algorithms rely on the usage of two keys, instead of just one key as in conventional algorithms. One key is used for encryption and a different but related key is used for decryption. For this reason these algorithms are sometimes referred to as asymmetric-key algorithms. These algorithms have the important characteristic that it is computationally infeasible to determine the decryption key (which is kept private) given only knowledge of the cryptographic algorithm and the encryption key (which is kept public).

The data that is to be encrypted is processed using a trap-door one-way function together with the encryption key. It is then impossible to read the cipher without access to the decryption key. The two keys are generated in such a way that they complement each other. Thus, the decryption key works as a trap door [Schneier 1994].

Scenario

As a scenario of the usage of public-key algorithms, let us see how we can solve the problem of establishing a secure channel between two persons, Alice and Bob, who have never had any previous contact. If Alice wants to send a message to Bob, this requires that both Alice and Bob maintain one public encryption key E and a secret decryption key D. Alice encrypts the message using the function F and Bob's public-key E_B [Stallings 1999].

$$Ciphertext = F(E_B, Plaintext)$$

When Bob receives the ciphertext he can easily decrypt it using his private key D_B and the same function F:

$$Plaintext = F(D_B, Ciphertext) = F(D_B, F(E_B, Plaintext))$$

3.6 Message Authentication

Message authentication is a procedure to verify the identity of the sender of a message and to verify that the received message has not been altered during transfer. Thus, it involves the sub-areas authentication and integrity. According to Stallings [1999] message authentication mechanisms can be divided into three classes:

- **Message encryption**
- **One-way hash function**
- **Message authentication code (MAC).**

3.6.1 Message encryption

According to Stallings [1999], in certain cases message encryption can provide guarantees that a transmitted message has not been modified during transfer and that the identity of the sender is correct. If a message transmitted from source A to destination B is encrypted using a secret key that is shared only by A and B, then clearly the recipient can be sure about the identity of the sender if and only if the message follows a predefined standard. For example if the recipient knows that the sender will send a message containing written text, then the predefined standard is that the message should be readable. If the cipher has been modified during transfer, then decryption will result in a message containing completely random data, that is certainly not readable.

So far so good. However, suppose that the message can be any arbitrary bit pattern. In this case there is no way to automatically determine that the message is not a fake message and that it has not been tampered with. Thus, message encryption in itself is not enough for providing message authentication [Stallings 1999].

In the discussion above it has been taken for granted that the sender and receiver share a common secret key. However, the discussion applies equally well to a situation where public-key encryption is used. In this case, A would encrypt the message using his own secret key and B would decrypt the cipher using A's public-key. The same problems remain.

3.6.2 One-Way Hash Function

One way to overcome the problems mentioned above, is to calculate a so-called message digest of the message and appending it to the end of the message. The receiver can then automatically see if the message has been tampered with by calculating a new message digest and comparing it with the appended digest, because even if the message in itself contains random data the message digest always follows a predefined format.

A *message digest* is a function that "digests" a message and generates a fixed-size code which is characteristic for the message. A one-way hash function is a popular form of message digest. The hash function accepts a message as input and generates as output a hash code, which depends on every single bit in the message.

Encryption is then needed for the message digest to provide message authentication Stallings [1999]. The following possibilities exist:

- Append the message digest to the message and encrypt the entire block. This is used when both message authentication and message confidentiality is needed.
- Encrypt only the message digest and append the encrypted digest to the unencrypted message. This is used when message confidentiality is not needed, but only message authentication.

3.6.3 Message Authentication Code

A message authentication code (MAC) is similar to a hash code (message digest) in that it generates a fixed-size code that depends on the entire message. What makes the MAC different from the hash code is that the generation of the MAC involves a secret key. For that reason the MAC is also known as a cryptographic checksum. Both the sender and receiver of a message need access to a common secret key for generating and validating the MAC [Stallings 1999].

The cryptographic function used to generate the MAC has one important difference from standard cryptographic functions in that it only works one way. A standard encryption function can be used to generate a cipher from plain text and then recreate the plain text from the cipher. However, in the case of a MAC the sender and receiver perform the same operation. The receiver generates a new MAC of the received message and compares this MAC to the appended one to authenticate the message.

For mathematical reasons the feature that the cryptographic function does not have to be reversible actually results in stronger encryption. Non-reversible ciphers are also faster to generate than standard ciphers. For this reason, using a MAC is preferable to using an encrypted hash code in situations where message authentication is needed but not confidentiality.

3.7 Network Security

Since a majority of public networks are based on the TCP/IP protocol stack, the discussion on network security in this chapter will be focused on this category of networks. However, the concepts should apply to all kinds of networks that follow the OSI standard reference model.

In a computer network based on the TCP/IP protocol the different security services can be classified based on what level in the protocol stack the security service is placed. Three possibilities exist: network layer security, transport layer security and application layer security [Stallings 1999].

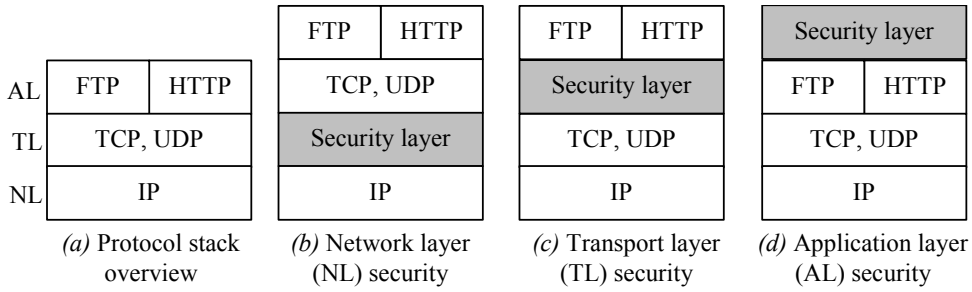


Figure 10 Possible locations of security services in the TCP/IP protocol stack.

Figure 10 above gives a schematic presentation of the possible positions for the security layer in the protocol stack. Figure 10 *(a)* gives an overview of the TCP/IP protocol stack. The different layers in the stack handle the network communication on different levels of abstraction. It should be said that the lowest protocol layers (the data link layer and the physical layer) which handle the low-level physical network communication are not included in the figure. The *network layer* is the lowest layer in the figure and it handles the sending and receiving of IP packets. The *transport layer* is responsible for maintaining a continuous and reliable connection between the two communicating parties. It consists of two protocols, TCP and UDP, where TCP guarantees that all data sent will arrive at the receiver and UDP does not. At the top is the *application layer*, which provides services to applications. An example of such an application layer service is file transfer [Tanenbaum 1996].

Below are the different security approaches [Stallings 1999]:

- **Network layer security:** In Figure 10 *(b)* a security layer has been placed above the network layer. Since the security service is placed below the transport layer it is transparent to the applications and secures all network communication. Thus, because of the transparency there is no need to rewrite the software in the system. The only available security service on this layer is IPSec, which is discussed in section 3.7.1.
- **Transport layer security:** In Figure 10 *(c)* a security layer has been placed above the transport layer. This has as a result that the security is transparent to the user, just as with network layer security. The only available security service here, to our knowledge, is an implementation of SSL, as discussed in section 3.7.2.
- **Application layer security:** In Figure 10 *(d)* a security layer has been placed above the application layer. Application specific security services are embedded within particular applications. This is discussed in section 3.7.3.

3.7.1 Network Layer Security

If the security service is placed in the network layer a so-called virtual private network (VPN) can be created within a public and unprotected network (in this case the Internet). In Figure 11 a virtual private network has been created between two private LANs and one autonomous computer running for example a web-browser.

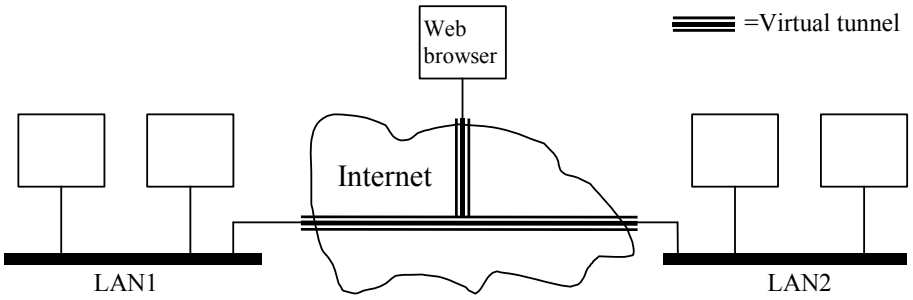


Figure 11 Virtual private network within a public network.

The only possible network layer security is the IPSec standard, an extension of the IP standard. The IPSec standard was proposed by the Internet Architecture Board in 1995 to counter the prevailing security problems on the Internet. IPSec ensures confidentiality, integrity and authenticity. The IPSec standard is discussed more in detail in section 4.2.1.

3.7.2 Transport Layer Security

SSL (Secure Socket Layer) is a protocol proposed by Netscape for enabling secure communications on the web. By origin SSL was designed with input from the industry and thus soon become a de facto standard for network security.

The SSL security mechanism is placed on the transport layer, which have the result that all network communication within a session[1] is secured. SSL has three security aims [Stallings 1999]:

1. To authenticate the server and the client using public-key signatures and digital certificates.
2. To provide an encrypted connection for the client and server to exchange messages confidentially.
3. To ensure that messages are not altered in transfer.

To provide reliable end-to-end security all data sent within one session is encrypted using symmetric-key cryptography and authenticated by generating

[1] An *SSL session* is an association between a client and a server, created after handshake. Several applications can run within one session.

and appending a MAC. Public-key cryptography is used to securely exchange a session-key when the session is established.

The SSL standard includes support for a number of different algorithms for encryption (such as DES, IDEA and RC4), authentication (such as MD5 and SHA-1) and key exchange (such as RSA and Diffie-Hellman). A collection of three such algorithms is called a cipher suite.

The SSL standard is not a single protocol, but rather a number of different protocols [Freier et al. 1996]. The two more interesting protocols are:

- **SSL Handshake Protocol:** This protocol allows the server and the client to authenticate each other and to negotiate what cipher suite to use when the session is established. Key-exchange is also managed by this protocol.
- **SSL Record Protocol:** This is the main protocol that provide services for confidentiality and message authentication.

Figure 12 shows the over all operation of the SSL Record Protocol. The Record Protocol takes application data as input, fragments it into blocks of 16,384 bytes or less, compresses each block, adds a MAC and performs encryption. Finally, an SSL header is appended.

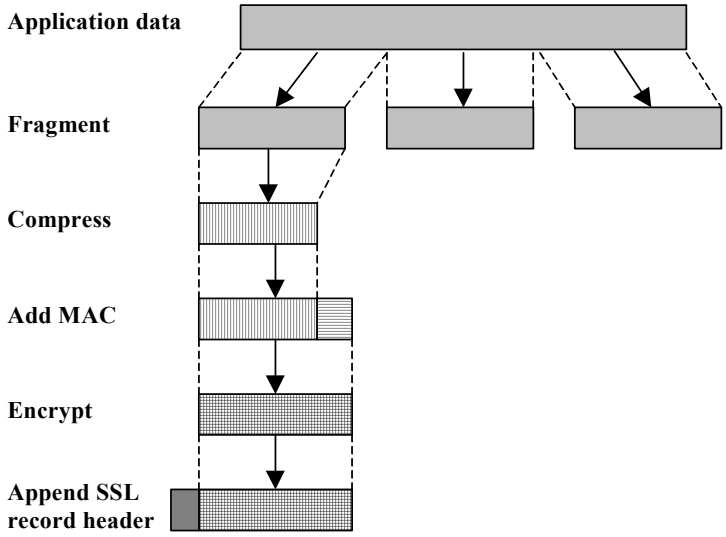


Figure 12 SSL Record Protocol operation [Stallings 1999].

Authentication in SSL

The way in which SSL authenticates messages is worth a discussion since it has implications on the test phase. As mentioned earlier in chapter 3.6.2, a quite sufficient method for authenticating a message and to guarantee message integrity is to generate a hash code of the message, appending it to the message and finally encrypting everything using a standard encryption algorithm.

However, in SSL a different approach has been selected. Message authentication is achieved by generating a so-called cryptographic checksum (or a MAC) according to an SSL-specific algorithm (unofficially referred to as SSL-MAC). The SSL-MAC algorithm uses an ordinary hash function for generating hash codes but it also incorporates a secret key in the authentication process. The operation can be simplified as follows:

1. Generate two keys from the secret key.
2. Append one of the keys to the message and generate a hash code of the lot (using for example MD5 or SHA-1).
3. Append the other key to the hash code and generate yet another hash code of everything. This new hash code is the cryptographic checksum.

If this approach to authenticate messages instead of just generating a simple hash code is good or bad depends on weather the messages needs to be kept confidential or not.

- **If confidentiality is not required**, then SSL-MAC is better than generating a simple hash code, because the hash code has to be encrypted regardless on whether the rest of the message is encrypted or not, and it is faster to generate an SSL-MAC than to encrypt a hash code.
- **If confidentiality is required** then the opposite applies. If you generate an SSL-MAC, append it to the message and then encrypt the lot, then you can say that the checksum has been encrypted twice. This is inefficient.

3.7.3 Application Layer Security

Application layer security is embedded within a particular application. The advantage of this approach is that the service can be tailored for specific needs of a given application. The disadvantage is that it is time-consuming to introduce new security mechanisms in an existing software platform, because parts of the software has to be rewritten [Stallings 1999].

Application layer security is often achieved by implementing known encryption and authentication algorithms directly into the code of an application. Even more common is to use a precompiled package containing implementations of the most useful algorithms and a clear API. An example of this is Java Cryptography Extension (JCE) developed by Sun Microsystems. This package will be presented in section 4.2.3.

4 Security Services

In this chapter the security services that will be implemented and tested are selected. First the results of a survey of available security services are presented and an initial selection is performed, weeding out the most unsuitable services. The services that passed the initial selection are then presented in detail. After that we select the criteria that the services should be evaluated upon, and finally the services are evaluated theoretically based on the selection criteria.

4.1 Survey of Available Security Services

In this section the results of a survey of possible security services or ways of attaining security are presented. A preliminary and informal selection is then performed and presented.

Available services

As said in section 3.7, security services can be classified as network layer security, transport layer security and application layer security depending on what level in the protocol stack the security service is placed. The available services are divided according to the following classes:

- **Network layer security**: The only available choice in this class is IPSec.
- **Transport layer security**: In this class two different implementations of SSL 3.0 can be found, namely WebLogic SSL and JSSE (Java secure socket extension). TLS 1.0 (transport layer security) is also an option.
- **Application layer security**: In this class we can see three possibilities for implementing security. The first possibility is to use JCE (Java cryptography extension) from Sun Microsystems, which is a ready to use package of algorithms. The second possibility is to implement cryptography algorithms by hand and include the code in the software. The third possibility is to use the SESAME security system.

Preliminary selection

At this point **JSSE** (which is a Java implementation of SSL) can be discarded before a formal evaluation. The reason for this is that JSSE is impossible to implement in the test environment. It has earlier been decided that the tests will be based on the WebLogic server. This server works as an integrated environment making it is impossible to introduce a new transport layer security service.

TLS 1.0 is basically the same as SSL 3.0, but with a different name. Therefore this service is discarded.

We can also discard the second one of the possible application layer security services, that is, to implement the algorithms by hand and include the code into the software. This method is simply too tedious and would hardly work in PAX-NG.

4.2 Description of Security Services

In this section the four candidate services (IPSec, WebLogic SSL, JCE, and SESAME) are presented.

4.2.1 IPSec

The main work on IP Security begun in 1994 when the Internet Architecture Board (IAB) issued a report entitled "Security in the Internet architecture" [RFC 1636]. The report presented a survey of the prevailing security problems on the Internet and identified key areas for necessary security mechanisms.

In 1995 the Internet Engineering Task Force (IETF) published five proposals of security-related standards. These standards were then implemented as the IP Security protocol (IPSec) standard in [RFC 2207]. IPSec was designed to work with both the current version of the Internet Protocol, IPv4, as well as with the future IPv6. IPSec provides similar services as SSL, but at the network level, in a way that is completely transparent to the applications [OpenBSD 2000].

To avoid IPSec from becoming obsolete and useless as algorithms for encryption and authentication become out of date and unreliable, IPSec only defines the mechanisms for security and specifies default algorithms. New algorithms can then be introduced without affecting other parts of the IPSec standard. The four security mechanisms in IPSec are [Stallings 1999]:

- Security Association
- Key Management
- Authentication
- Encryption.

Each of these four mechanisms is described below.

Security Association

A Security Association (SA) is a one-to-one relation between the sender and the receiver of data. The SA is used for storing security-related information such as information about algorithms, sequence numbers, keys, and so on. A full list of SA parameters can be found in [RFC 1825].

Key Management

The key management portion of IPSec involves generation, distribution and management of keys. The IPSec architecture supports two types of key management:

- **Manual:** A system administrator manually configures each system with its own keys. Practical for small static environments.
- **Automated:** An automated system enables the on-demand creation of keys. Suitable for large distributed systems.

Authentication

IPSec specifies the use of an Authentication Header (AH) as a mechanism for providing strong integrity and message authentication for IP packages [RFC 1826]. The AH may also be used as a mechanism for providing non-repudiation if a public-key cryptographic algorithm, such as RSA, is used. It should be mentioned that IPSec does not provide user-to-user authentication, only machine-to-machine authentication.

The default algorithm specified by IPSec is HMAC with MD5 or SHA-1 (both combinations has to be implemented). MD5 is generally considered to be faster than SHA-1, but SHA-1 offers a higher level of security [Stallings 1999]. AHs can operate in two different modes: transport mode and tunnel mode. In both cases the entire packet is authorised.

For **transport mode** the AH is inserted after the original IP header and before the IP payload as shown in Figure 13 *(a)*.

For **tunnelling mode** the entire IP packet is inserted into the payload of a new IP packet. The AH is then placed after the new IP header but before the old header as shown in Figure 13 *(b)*.

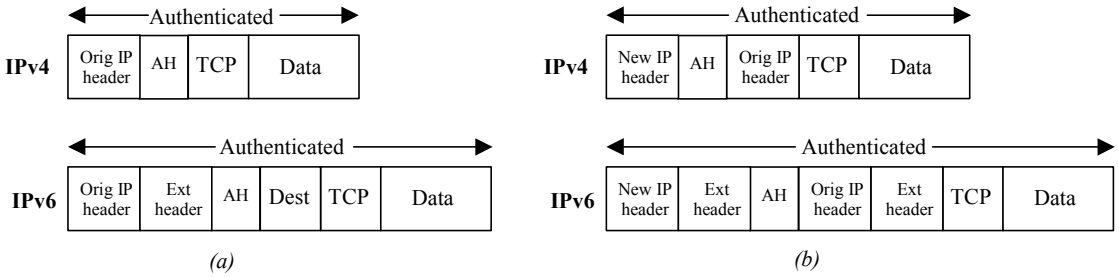


Figure 13 Modes of AH authentication: (a) Transport mode and (b) Tunnel mode.

Encryption

IPSec specifies the use of the Encapsulation Security Payload (ESP) extension header for encryption of IP packets. More information on the format of the ESP header can be found in [RFC 1827].

Support for many different security algorithms (such as 3DES, Blowfish and IDEA) have been included. The default encryption algorithm is DES in Cipher Block Chaining mode (CBC).

Just as with AH, ESP can be used in two modes: transport mode and tunnel mode. See the description in the former paragraph for more information on these two modes.

4.2.2 WebLogic SSL

WebLogic SSL is an implementation of SSL 3.0 as specified by Netscape [BEA/2 1999]. As mentioned in section 3.7.2, SSL provide secure encrypted and authenticated connections across insecure networks. The default algorithms used by WebLogic SSL are RC4 for encryption, RSA for key exchange and MD5 for generating message digests. WebLogic SSL supports certificates, for authentication, generated by certificate authorities such as VeriSign, GTE CyberTrust or Entrust.

4.2.3 JCE

Java Cryptography Extension (JCE) is part of a larger framework called Java Cryptography Architecture (JCA) provided by Sun Microsystems for accessing and developing security services for the Java platform [Pistoia et al. 1999]. Because of a number of export restrictions on cryptographic algorithms, Sun broke the cryptography architecture into a set of interfaces included in JDK and the implementation of these interfaces into JCE [Steel 2000]. The JCA was designed around the following two principles [Sundsted 1999]:

- **Implementation independence and interoperability**: A developer must be able to write applications without tying them too closely to a particular algorithm. In addition, as new algorithms are developed, they must be easily integrated with existing algorithms.
- **Algorithm independence and extensibility**: A developer must be able to write applications without tying them to a particular vendor's implementation of an algorithm.

Designing a system of engines and providers satisfies these requirements. An engine is an abstract representation of a cryptographic service that does not have a concrete implementation [Pistoia et al. 1999]. A security service provider on the other hand is a package that provides an implementation of some subset of the cryptographic services. A schematic presentation of the basic functionality of

the provider concept can be seen in Figure 14. JCE can be seen as a wrapper interface into which a service provider can be inserted, like a light-bulb into a socket. All communication with the provider goes through JCE.

As mentioned above, JCA works as a framework for the collection of cryptographic services placed in JDK and in JCE. More concretely, the division works like this: Java 2 JDK store all interfaces for services involved in JCA and implement services for generating message digests and digital signatures. All services that are under export restriction, such as encryption, secret-key distribution and MAC generation is implemented in JCE. Just as there are many service providers that implement security algorithms, there also exist several "clean room" implementations of JCE, apart from the standard implementation from Sun. In this way it is possible to go around export restrictions.

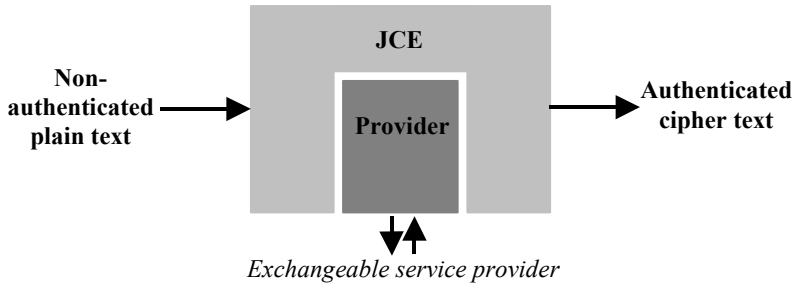


Figure 14 Basic functionality of JCE. The JCE interface works as a wrapper around a replaceable security service provider.

4.2.4 SESAME

SESAME (a Secure European System for Applications in a Multi-vendor Environment) is a technology that has been developed to provide security to client server systems [Ashley et al. 1999]. SESAME is similar to SSL in that it provides a wide spectrum of services for secure network communications. The large difference between the two is that SSL is situated in the transport layer and SESAME is situated in the application layer. This has as a result that SSL in theory is completely transparent for the user and SESAME is not. Thus, SSL only provides security services on a machine-to-machine level and SESAME provides services on a user-to-user level. This has as a result that the two services, in theory, perform authentication differently. While SESAME authenticates users, SSL only authenticates machines. However, SSL deviates from the theoretical model and actually does provide user-to-user authentication.

Ashley and colleagues [1999] argue in their report that SESAME in theory is a better choice than SSL, since it provides all services that SSL provides and more. SESAME for example provides services for access control. Thus, SSL is considered to be a subset of SESAME. See *Table 2*.

However, since today's browsers and web servers do not support any other security service than SSL, implementing SESAME in a web-based system is difficult. Ashley and colleagues [1999] therefore concludes that SESAME is best suited for intranet solutions, handling internal communication within organisations.

Table 2 *Comparison between SESAME and SSL.*

Security Service	SESAME	SSL
User Authentication	Yes	(Yes)
Data Confidentiality	Yes	Yes
Data Authentication	Yes	Yes
Access Control	Yes	No
Non-repudiation of Origin	Yes	No
Auditing	Yes	No

4.3 Identification and Selection of Criteria

To be able to perform an objective evaluation of the security services, we need to initially identify what criteria the services should be judged upon.

Selected criteria

The following criteria have been selected:

- **Performance:** The security services need to be fast to be suitable for every day use in a large system handling large amounts of data.
- **Level of security:** Since PAX-NG will be used for handling private and possibly classified information, it is important that the selected security services have a high security profile.
- **Simplicity:** The selected security services should be simple to implement and maintain in the system. However, since PAX is quite a large software system with many customers, this criterion is not crucial since the implementation of security is such a small part of the software development process.
- **Enterprise JavaBeans compatibility:** The precondition for this project is to evaluate ways to attain security in a distributed system based on the Enterprise JavaBeans standard. Thus, the selected security services must be implementable in an EJB environment.

Discarded criteria

From the original set of possible criteria, two were discarded because they did not seem important. A criterion for evaluating the **cost** (as in price) for purchasing a security service was discarded, because of the difficulties to estimate this cost. The cost for such a purchase would probably be determined

through negotiation. A criterion for evaluating **memory requirements** was also discarded with motivation that limited memory capacity could hardly be a problem on today's computers.

4.4 Theoretical Evaluation and Selection

In this section we present the results from a theoretical evaluation of the four security services based on the selection criteria. A summary of the evaluation can be found in paragraph 4.4.5.

4.4.1 Evaluation of IPSec

The **performance** of IPSec is good. In an evaluation of a Linux implementation of IPSec by [Gautier 1999] transfer speeds around 35 % of normal (non-secured) transfer speeds were reached when IPSec was activated.

According to a cryptographic evaluation of IPSec performed by Ferguson and colleagues [1999], IPSec is a great disappointment from a security perspective. They argue that the complexity of the system makes it difficult to evaluate security weaknesses, and virtually impossible to keep security-critical errors out. Further more they say that the complexities of the system combined with exceptionally bad documentation makes it more or less certain for a systems designer to create a system that does not implement IPSec the right way and thus not achieve its security goals. Because of this we conclude that IPSec does not meet the criteria on **security** and **simplicity**.

Since IPSec operate on the network level, it is completely out of scope of the EJB applications programmer. Thus, we conclude that the security service is not **Enterprise JavaBeans Compliant**.

Since IPSec did not meet three out of four criteria, we find it necessary to discard it from the list of possible security services.

4.4.2 Evaluation of WebLogic SSL

We have not found any references examining the **performance** of SSL. For this reason, the criterion can not be evaluated at this point.

After an extensive theoretical evaluation of the cryptographic strength of SSL 3.0, Bruce Schneier and David Wagner conclude that SSL 3.0 has an excellent over all **security** profile [Schneier et al. 1997]. Even though there are some minor security flaws, they do not constitute a reasonable threat to security. Since WebLogic SSL implements SSL 3.0 the same should apply for it.

SSL is in itself designed to be easy to use in applications, as the security service is placed at the transport layer and thus is virtually transparent to the application programmer. Activating WebLogic SSL on the WebLogic server is simply done

by specifying that a certain transport protocol (implementing SSL) should be used. Thus, it can be stated that WebLogic SSL meets the **simplicity** criterion with excellence.

Since the Enterprise JavaBeans standard supported by the WebLogic server, the WebLogic SSL security service is **Enterprise JavaBeans compliant**.

WebLogic SSL meets the criteria with excellence. The service is therefore selected for implementation and testing.

4.4.3 Evaluation of JCE

JCE does not contain any cryptographic algorithms in it self, so it is impossible to evaluate the **security** and **performance** of JCE. Instead one should evaluate the cryptographic strength of the algorithms that are implemented by the individual service providers. However, there exist providers from trustworthy companies, such as RSA, so we consider the security criterion met, yet with some hesitation. For the same reason it is impossible to evaluate the **performance** of JCE.

JCE has a well-defined, straightforward interface and is rather easy to use [Sundsted 1999]. The **simplicity** criterion is met.

JCE is a Java standard developed by Sun Microsystems, just the same as the Enterprise JavaBeans standard. JCE therefore meets the **Enterprise JavaBeans compliance** criterion with excellence.

The JCE security service meets all criteria, even though with some hesitation concerning **security**, and is therefore selected for implementation and testing.

4.4.4 Evaluation of SESAME

We have not found any references examining the **performance** of SESAME. For this reason, the criterion can not be evaluated at this point.

Ashley and colleagues [1999] mentioned that SESAME provides a similar cryptographic strength as SSL. Thus, the **security** criterion is met.

As mentioned earlier, today's web browsers do not support any other security service than SSL, and even though there is an interface for integrating SESAME with a web browser, GSS-API, this is rather difficult to do. Since the idea is that it should be possible to run PAX-NG clients on web browsers, we consider that the **simplicity** criterion is not met.

We have not been able to determine weather SESAME is **Enterprise JavaBeans compliant** or not.

Since the security service did not meet the simplicity criterion and there were some uncertainty concerning EJB compliance, we decide to discard SESAME as an option.

4.4.5 Summery

Two of the security services, **WebLogic SSL** and **JCE**, were selected as they comply with all the criteria mentioned earlier. **IPSec** and **SESAME** were discarded, as they did not meet all criteria.

These two security services will be implemented and their performance evaluated in the next chapter.

5 Testing and Evaluation

This chapter describes the testing phase of the project. After a general overview of the objectives with the tests, each test bench that is used for testing the selected security services is presented thoroughly. Finally the test results are presented, together with a discussion of the validity of the measurements and an evaluation of the test results.

5.1 Test Objectives

The objective of the tests was to evaluate the performance[2] of two selected services for achieving secure communication between a client and a server in a distributed environment. The two selected services are WebLogic SSL and Java Cryptography Extension (JCE). In JCE we specifically tested four encryption algorithms and two algorithms for generating message digests. The selection of these algorithms is presented in section 5.3.

5.2 Test Benches

This section presents the test benches that we developed in this project for testing and comparing the two selected security services.

5.2.1 Overview

Since the goal of this project was to investigate security issues in a distributed Enterprise JavaBeans (EJB) environment, our first step in the implementation phase was to develop a simple test bench implementing the EJB standard. This test bench was then cloned into two new test benches, which were then separately developed further to introduce the two selected security services. See Figure 15 for an overview. In the figure, the boxes represent the test benches and the two shapes that make up each box symbolise the client application and EJB pair that makes up each test bench.

Thus, each *test bench* is built as a client/EJB pair, where the business-logic is running as an EJB on a WebLogic application server, and the client application consists of the GUI and logic necessary for establishing communication between the client and the EJB. It should be mentioned that the application server and the client are running as two processes on the same computer, so the test results are not affected by network performance.

[2] *Performance* in this case refers to processing time solely.

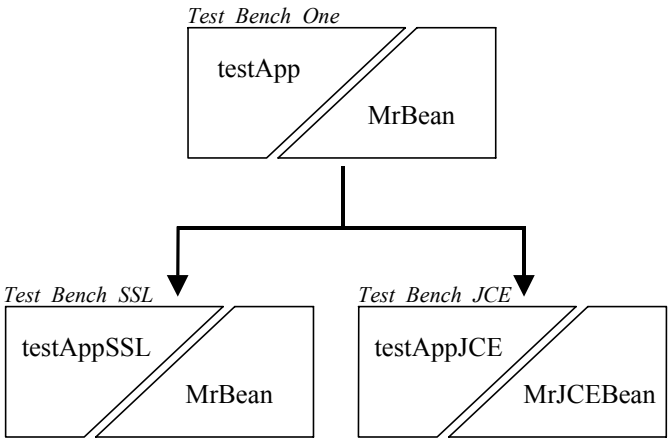


Figure 15 Development of test benches. The original Test_Bench_One is cloned into two new test benches, which implement security services.

5.2.2 Test_Bench_One

The functionality of the first test bench is very simple. The client application, called *testApp* (see Figure 16 for a screen-shot) requests services from a simple EJB called *MrBean* that runs on the application server.

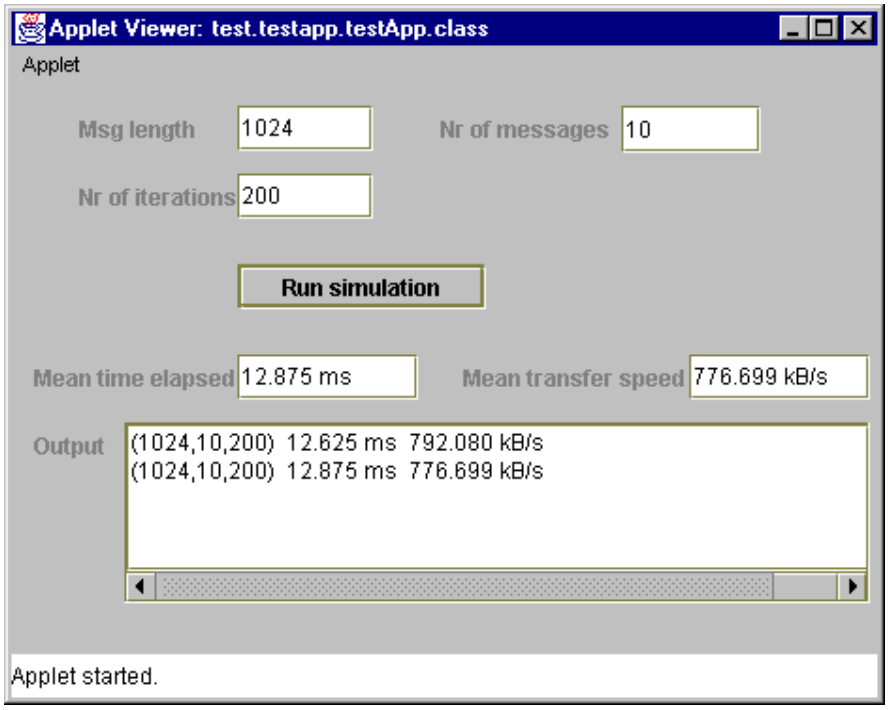


Figure 16 Screen-shot from testApp.

The test bench simulates data transfer between the EJB and the client and provides services for measuring and calculating transfer speeds. In three text

fields in the client-application the user enters parameters that control the simulation. Two parameters control the size of the test data blocks and the number of blocks that should be sent within one test iteration. The third parameter controls the number of iterations the simulation should be run. An *iteration* begins and end with starting and stopping the timer. Within one iteration one or more blocks of test data is transmitted. The purpose of running the simulation many times is to get good mean values since measured transfer times differ between simulations. The test data used consisted of a string with the character A repeated a certain number of times.

Figure 17 shows the call structure between testApp and MrBean. Only one iteration is demonstrated, but in reality the calls within the dashed lines (one iteration) should be repeated a large number of times. MrBean provides the following services to the client:

- `create()` – Establish connection with the application server and initialise the bean.
- `initialiseData(int length)` – Generate test data and store it in the bean. In the test process the same test data is then reused to save time.
- `initialiseTimer()` – Start the timer.
- `getData()` – Return test data (in other words a message).
- `stopTimer()` – Store stop time.
- `getTimeElapsed()` – Calculate and return time elapsed (stop time minus start time).
- `getTransferSpeed()` – Calculate and return transfer speed.

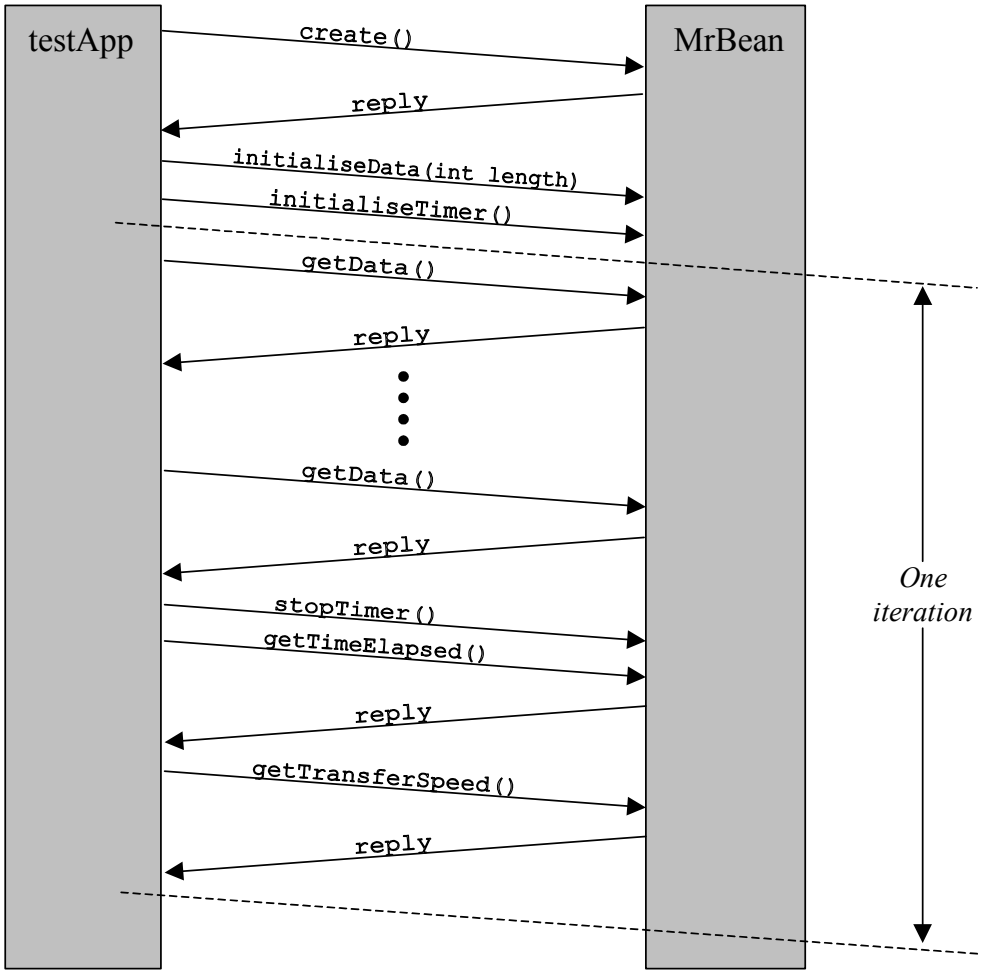


Figure 17 Call structure between testApp and MrBean in the first test bench.

5.2.3 Test_Bench_SSL

The functionality of this test bench is identical to the original Test_Bench_One with the only difference that the communication between the client application and the EJB is secured through WebLogic SSL. This security service is provided by the WebLogic server and is simply activated by specifying that a specific protocol should be used in the communication between the client-application and the server. Because of this we could reuse MrBean and simply modify the client-application (the new client-application was called *testAppSSL*). The GUI was the same as for the original Test_Bench_One (see Figure 16), and the call structure between testAppSSL and MrBean was identical to the call structure between testApp and MrBean (see Figure 17).

5.2.4 Test_Bench_JCE

This test bench is based on the original Test_Bench_One. The functionality is the same, but several services have been modified to support security. The new client application and EJB are called *testAppJCE* and *MrJCEBean* and they both implement the Java Cryptography Extension (JCE). Secure communication is achieved by generating and appending a message digest to the test data and then encrypting everything. This is similar to the SSL standard (see 3.7.2). The message digest ensures message integrity and encryption ensures confidentiality. The algorithms that were tested are presented in section 5.3.

The GUI is almost the same as in the other test benches, with the addition of two drop-down menus for selecting which encryption algorithm and message digest to use. See Figure 18 for a screen-shot.

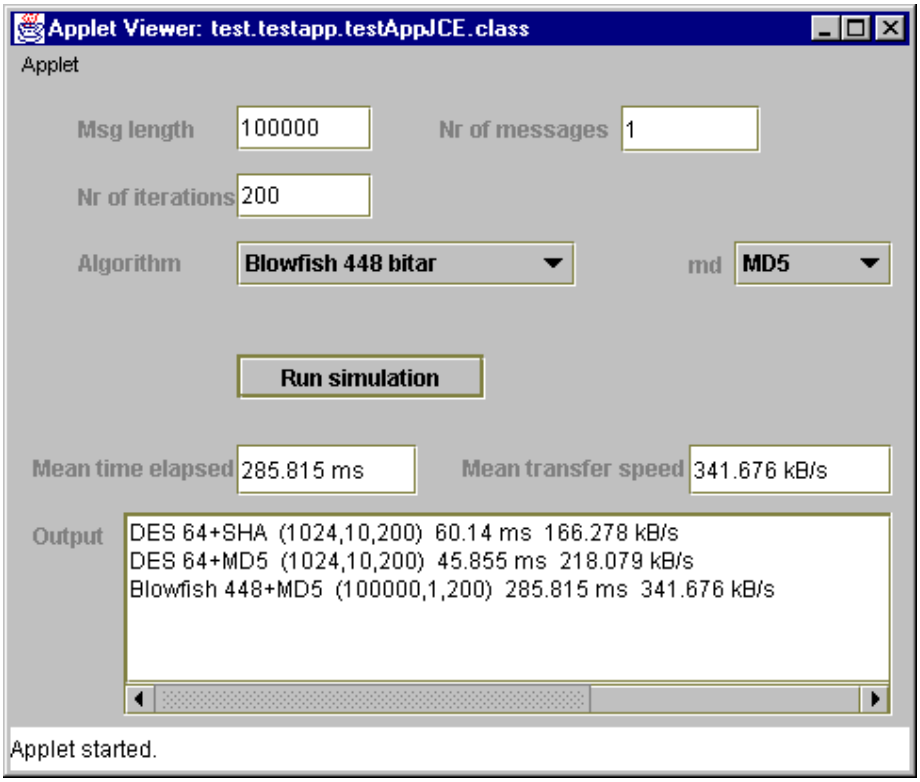


Figure 18 Screen-shot from testAppJCE.

Figure 19 show the call structure between testAppJCE and MrJCEBean. As before, only one iteration is demonstrated. MrJCEBean provides the following services to the client:

- `create()` – Establish connection with the application server and initialise the bean.

- `initialiseSimulation(int length, String algName, String digestName, int algKeyLength)` – Generate test data, initialise cipher and message digest objects and generate and return a session key.
- `initialiseTimer()` – Start the timer.
- `getData()` – Return the encrypted test data with a message digest appended to it.
- `stopTimer()` – Store stop time.
- `getTimeElapsed()` – Calculate and return time elapsed (stop time minus start time).
- `getTransferSpeed()` – Calculate and return transfer speed.

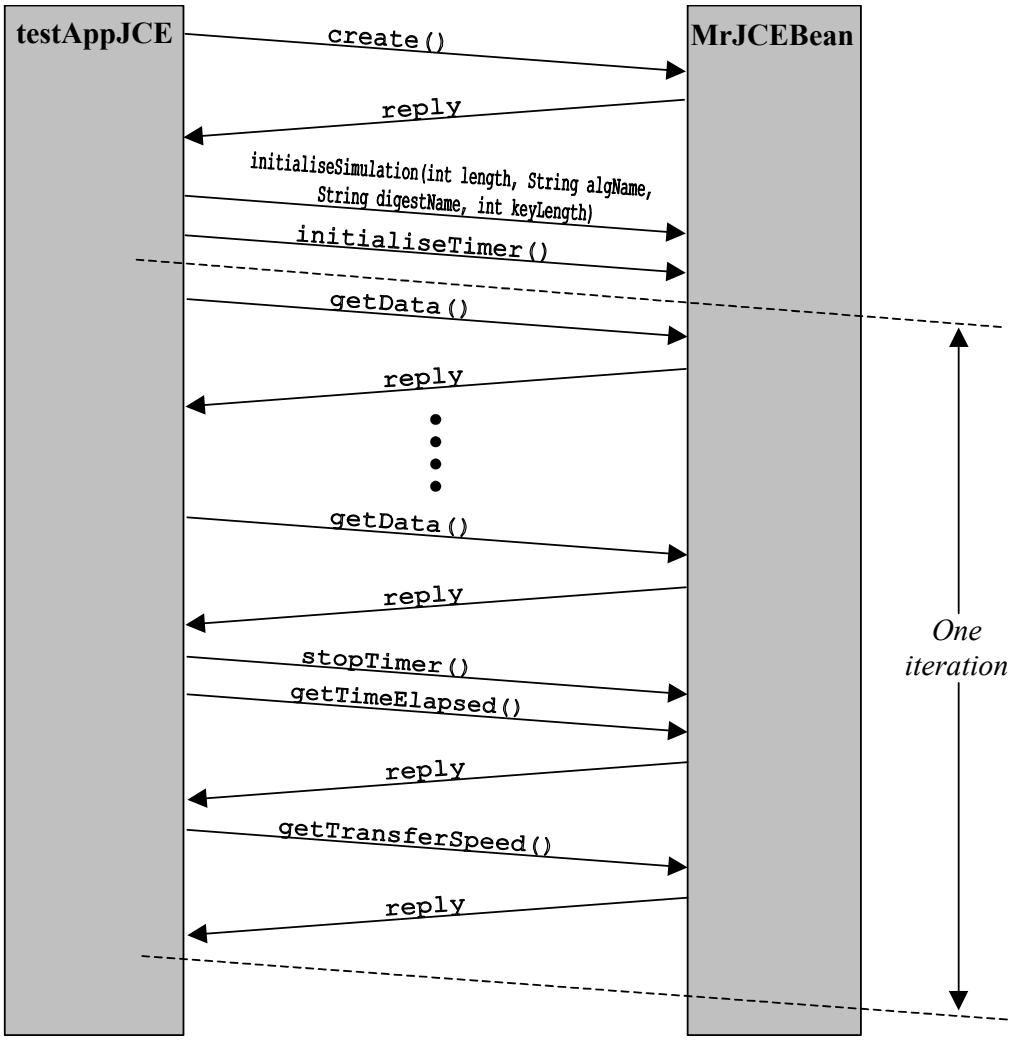


Figure 19 Call structure between testAppJCE and MrJCEBean in Test_Bench_JCE.

5.3 Selection of Algorithms

In this section we present a list of available and selected algorithms for encryption and generation of message digests that were used in Test_Bench_JCE. The security service that was tested on Test_Bench_JCE provided many different security algorithms, and to simplify testing we selected to only test half of them. The algorithms were selected to represent the diversity of algorithms.

Encryption algorithms

The following ciphers were available: DES, 3DES, IDEA, RC2, RC4, RC5, and Blowfish.

DES and IDEA are among the most well known block ciphers around, so these were selected automatically. The stream cipher RC4 is known for its high performance and was selected for this reason. Blowfish is the newest block cipher of the ones available and was selected for this reason. 3DES was discarded since it is so similar to DES (in fact, it is DES, repeated three times). RC2 and RC5 were also discarded since they do not provide any features that the other selected algorithms do not.

These selected algorithms are studied in detail in appendix B.1.

Message digest algorithms

The following algorithms for generating message digests were provided: MD2, MD5, SHA-1, and RIPEMD-160.

MD5 and SHA-1 are the most well known algorithms and were selected for this reason. RIPEMD-160 was discarded because it is not commonly used. MD2 was designed in 1989 and are for this reason optimised for 8-bit machines. Therefore it was discarded.

These selected algorithms are studied in detail in appendix B.3.

5.4 Test Environment

The tests were conducted on a Pentium-II with 128 MB RAM running Windows NT 4.0. WebLogic 4.5.1 from BEA was used as application server. Visual Café 3.0c using JDK1.1.7B was used to develop the test benches and the client applications were also run from Visual Café. The fact that the client applications were run from Visual Café and not from a stand-alone Java run-time environment, might have influenced the test results. It is probable that we would get higher transfer speed measurements, because of increased memory availability, under different test conditions. However, this is not significant, since we were only interested in a relative comparison between different security services.

JCSI 1.0 from the Australian research centre DSTC Pty Ltd was used as provider of encryption and authentication algorithms. This is worth mentioning, since it is possible that we would have got different test results if we had used a different provider. We did try to find and install a second provider, but this proved difficult. The few providers we found were either out of reach because of export restrictions and licence fees or did not work properly in our test environment.

As mentioned before, the application server and the client application were run as separate processes on the same machine, so the test results are not influenced by network performance in any way.

5.5 Test Results

The results from the test series are presented in this section.

5.5.1 Measurements

On the three test benches we conducted a series of tests using test data blocks of five sizes to see if the security services behave differently under different conditions. The data sizes used was 1 byte, 512 bytes, 1,024 bytes, 10,240 bytes and 100,000 bytes.

In the test series we always used the same number of iterations and the same number of messages within each iteration in every test case where we tested a certain block size. However there have been some differences between test cases. Generally 200 iterations were used in each simulation and 100 messages were sent within each iteration. When testing data blocks of size 100,000 bytes we used 200 iterations, but only one message was sent in each iteration because of the large processing time. For data blocks of size 1 byte, 100 iterations were used and 1,000 messages were sent within each iteration.

On Test_Bench_JCE we tested the performance of the four encryption algorithms separately and used MD5 for generating message digests. We also tested using SHA-1 for generating message digests, but the processing time was always longer for that algorithm. This conclusion is consistent with [Stallings 1999]. For this reason, we decided to terminate the test cases using SHA-1 and only complete the test cases using MD5.

The results from the test series are presented in *Table 3*. The numbers presented are transfer speeds in kilobytes per second.

***Table 3**. Transfer speeds in kilobyte per second measured in the three test benches. The values within parentheses are transfer speeds relative to the speeds measured on Test_Bench_One.*

Size of test data blocks	Test_Bench One	Test_Bench SSL	Test_Bench_JCE (using MD5)			
			DES	IDEA	Blowfish	RC4
100,000 bytes	3488 (100 %)	1007 (29 %)	286 (8 %)	399 (11 %)	341 (10 %)	528 (15 %)
10,240 bytes	2857 (100 %)	877 (31 %)	286 (10 %)	405 (14 %)	341 (12 %)	543 (19 %)
1,024 bytes	901 (100 %)	353 (39 %)	218 (24 %)	291 (32 %)	249 (28 %)	351 (39 %)
512 bytes	472 (100 %)	203 (43 %)	169 (36 %)	212 (45 %)	185 (39 %)	244 (52 %)
1 byte	1.07 (100 %)	0.48 (45 %)	0.77 (72 %)	0.80 (75 %)	0.78 (73 %)	0.81 (76 %)

Figure 20 shows a diagram of the transfer speeds for the tested security services presented in *Table 3*. The test results from the different encryption algorithms tested in Test_Bench_JCE are presented separately. The test results are presented as relative transfer speeds compared to the transfer speeds that were measured in the original Test_Bench_One, that is, when no security service was activated. For example, the diagram shows that the measured transfer speed when sending data blocks of 100,000 bytes in Test_Bench_SSL are just below 30 % of the transfer speed measured in the original Test_Bench_One.

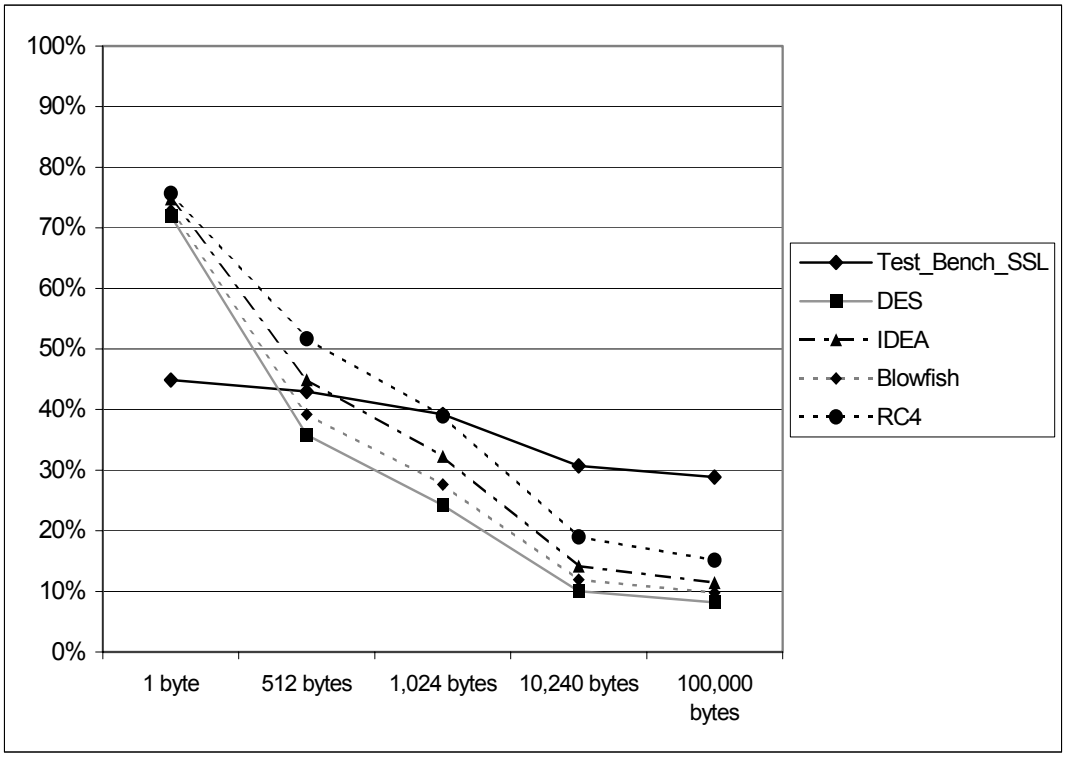


Figure 20 Diagram showing the transfer speeds for different block sizes measured in Test_Bench_SSL and Test_Bench_JCE relative to the transfer speed measured in the original Test_Bench_One.

5.6 Validation of Test Results

In this section we will try to evaluate the validity of the test results presented above. We do this by focusing on some key areas that might have influence on the test results.

5.6.1 Number of iterations and messages

As said in section 5.5.1 we did not use the same number of iterations and number of messages within each iteration for all test series. These parameters differed for test series using data sizes 1 byte and 100,000 bytes. Could this have effected the test results? To check this we ran a few tests (with data size 1,024 bytes) using different numbers of iterations and number of messages within each iteration. The results showed no evidence that these parameters greatly effect the performance of the security services. The only visible difference is that the accuracy of the test results decline if the total amount of data sent (that is, data size multiplied with number of iterations multiplied by number of messages) is too small. And we can not see that that this could be a problem in the test cases mentioned above.

5.6.2 Characteristics of the test data

To save time in the test initialising process in the test series, we did not use random test data in the tests. Instead a simple string consisting of a selected number of the character A was used as test data. Could this effect the test results? To find out if the processing times of the security services depended on the characteristics of the test data, we ran a number of validation tests where we used both random and non-random test data. The test results were identical so we conclude that the test results do not depend on the characteristics of the test data.

5.6.3 Key length

Does the key length used in a certain cryptographic algorithm effect the performance of that algorithm? In other words, will we get different test results for each algorithm if we use keys of different length?

The motivation to this question is that the security service included in the WebLogic application server exists in two different versions that use the same algorithms, but with keys of different length. One version supports strong encryption with a 128-bit key and the other version supports a weaker form of encryption with a 40-bit key (this due to United States export regulations). Since we only tested the version with weak encryption (because of difficulties in getting access to the other version) we wanted to know whether the key length used did affect the performance.

To answer this question we performed a validation test where we used two algorithms that support keys with variable length (Blowfish and RC4) and tested them in a separate JCE environment using keys of different length. The test results showed clearly that the key length does not effect the performance of the algorithms. This result is expected since the algorithms probably use an internal key of fixed length and simply stuff the shorter external key with nonsense bits to produce a longer key.

5.6.4 Providers

As mentioned in section 5.4, we only used one provider of encryption and authentication algorithms in the test series. This can affect the validity of the test results. However, we feel quite confident in our estimation that the test results from a different provider's algorithms could not have differed so much from the present test results that it would affect our evaluation, since every provider basically implements the same algorithms.

5.7 Evaluation of Test Results

The results from the test series showed that the evaluated security services behave very differently depending on the size of the test data blocks used. The test results presented in Figure 20 show that WebLogic SSL (evaluated in Test_Bench_SSL) is faster than every algorithm tested in Sun JCE (evaluated in Test_Bench_JCE) for data blocks of size one kilobyte or larger. JCE was faster for data blocks shorter than one kilobyte.

With data of size one byte, the fastest algorithm tested on JCE (RC4) was 40 % faster than WebLogic SSL. This difference is quite large and it would be interesting to understand why Sun JCE reached transfer speeds that were so much higher than those reached by WebLogic SSL for small data blocks.

Our first idea was that it might have to do with the activities that precede the actual data encryption, authentication and transfer. These activities include generating and distributing keys and preparing for encryption, authentication and for the communication activities in general. However, since all these activities are performed before the actual timing occurs, it is difficult to explain the differences due to this. (For simplification, the test bench operations can be summarised in the following steps: Initialisation, start timer, send data, stop timer and calculate transfer speed.)

The explanation that seems most probable at this time has to do with the slight difference on how authentication is performed in the two test benches. In Test_Bench_JCE authentication is performed by simply generating a message digest, appending it to the data block and encrypting everything. In WebLogic SSL the data is authenticated by generating a cryptographic checksum,

appending it to the data block and encrypting everything. Since the checksum is encrypted twice this results in some overhead. It is possible that this is more visible when small-sized data blocks are used.

Despite the origin of the differences mentioned above, it is safe to say that WebLogic SSL is the security service that displays the best over all performance. This can be said with confidence knowing that the typical data blocks being communicated between client and server applications exceed one kilobyte in size. The usage of test data blocks of size one byte can be seen as being merely of an academic interest.

6 Conclusions

In this chapter we present the conclusions from both the theoretical and the practical evaluation of available security services and provide a recommendation for a security service that is most suitable to use in the PAX-NG software system.

In the theoretical evaluation, one of the selected services excelled over the others, and this was WebLogic SSL. This service was reliable and showed an excellent level of security. Further more, this service was simplest to implement of them all.

The results from the performance evaluation of the implemented services are more uncertain. The results showed that WebLogic SSL was the best service for sending data blocks larger that one kilobyte in size and that JCE was the best service for sending data blocks shorter than one kilobyte in size. However, in the evaluation of the test results (section 5.7) we concluded that the data blocks that will be sent within PAX-NG are most certainly larger than one kilobyte, and that WebLogic SSL thus showed the best over all performance.

Since WebLogic SSL was selected as the best service in both the theoretical and the practical evaluation, this is the security service that we recommend for implementation in PAX-NG and also for implementation in similar distributed systems.

7 Future Work

Following are some recommendations on future work.

Providers

In the test series on Test_Bench_JCE only one security service provider was tested and evaluated. As mentioned earlier, this did probably not effect the validity of the test results to such a large extent that we would have come to a different conclusion if we had used some other service provider. However, this is only our estimation. If possible the best thing would be to extend the test series to include the testing of different providers. It is plausible that such extended testing will become easier to perform in the near future, since the trend is that export regulations on cryptographic services (which proved to be a problem) are being relaxed.

Full-scale testing

Another item for future work would be to implement the selected security service in a large-scale system and perform full-scale testing. Such a test was not possible during the writing of this thesis since the target system, PAX-NG, only exists as a design and conceptual idea. The best thing to do is of course to redo all the testing on the target system, but as this is probably not feasible, the second best thing would be to implement and evaluate the selected security service and establish that response times are not unexpectedly long.

Internal system security

Many interesting questions on internal system security have come to our mind when writing this thesis. One of these questions is to investigate what methods exist for regulating access control (as mentioned in 1.5), i.e., to regulate what a specific user can and can not do in a system. It would be interesting and useful to evaluate how secure and reliable such access control methods really are.

A good idea for further work is also to evaluate the over all system security in PAX-NG from a client perspective, for example to evaluate if attacks on security, such as buffer-overflow attacks or TOCTTOU (time of check to time of use) attacks, are possible.

8 References

[Andrade et al. 1996] Andrade, J. M., Carges, M. T., Dwyer, T. J., Felts, S. D., *The Tuxedo System,* Addison-Wesley Publishing Company, Inc. 1996, ISBN 0-201-63493-7.

[Ashley et al. 1999] Ashley, P., Vandenwauver, M., Claessens, J., *Using SESAME to secure web based applications on an intranet*, Secure Information Networks, Proceedings of the IFIP TC6/TC11 Joint Working Conference on Communications and Multimedia Security, Belgium, September 1999, pp 303-317.

[BEA/1 1999] *Introduction to WebLogic Server 4.5, Version 1.0,* BEA Systems, Inc. 1999. http://www.weblogic.com/docs45/intro/index.html (Accessed 2000-05-16)

[BEA/2 1999] *WebLogic Developers Guide*, BEA Systems, Inc. 1999. http://www.weblogic.com/docs45/classdocs/index.html (Accessed 2000-05-16)

[CERT 2000] http://www.cert.org/stats/cert_stats.html (Accessed 2000-05-16)

[Edwards 1997] Edwards, J., *3-Tier Client/Server at Work*, John Wiley & Sons, Inc. 1997, ISBN 0-471-18443-8.

[Eriksson 1999] Eriksson, L., *Master thesis: Security in Open Distributed Environments*, Linköping: Linköping University, Department of Computer and Information Science 1999, ISRN LiTH-IDA-Ex-99/60.

[Feistel 1973] Feistel, H., *Cryptography and Computer Privacy*, Scientific American, May 1973.

[Ferguson et al. 1999] Ferguson, N., Schneier, B., *A Cryptographic Evaluation of IPSec*, Counterpane Internet Security, Inc. 1999. http://www.counterpane.com/ipsec.html (Accessed 2000-05-26)

[Freier et al. 1996] Freier, A., Karlton, P., Kocher, P., *The SSL Protocol Version 3.0,* Netscape Communications, 1996. http://home.netscape.com/eng/ssl3/ssl-toc.html

[Fåk 1997] Fåk, Viiveke, *Course notes for Cryptology,* Dept of EE, Linköping University, 1997.

[Gautier 1999] Gautier, Robert, J., *Performance of the FreeS/WAN IPSec Implementation,* University of Wales, Computer Science Dept, 1999. http://tsc.llwybr.org.uk/public/reports/SWANTIME/

[Kauffman 1997] Kauffman, T., *3-Tier Client/Server Environment Goals and Specifications*, http://sandbox.aiss.uiuc.edu/3-tier/goals.htm

[Monson-Haefel 1999] Monson-Haefel, Richard, *Enterprise JavaBeans*, O'Reilly & Associates, Inc., 1999, ISBN 1-56592-605-6.

[Olovsson 1992] Olovsson, T., *A Structured Approach to Computer Security*, Technical Report No 122, Chalmers University of Technology, 1992.

[OpenBSD 2000] OpenBSD, *Using IPSec,* document version 1.32, 2000. http://www.openbsd.org/faq/faq13.html

[Pistoia et al. 1999] Pistoia, M., Reller, D. F., Gupta, D., Nagnur, M., Ramani, A. K., *Java 2 Network Security 2nd ed.*, New Jersey: Prentice-Hall Inc. 1999, ISBN 0-13-015592-6.

[RFC 1636] *Security in the Internet Architecture*, Braden, R., Clark, D., Crocker, S., Huitema, C., IETF, 1994.

[RFC 1825] *Security Architecture for the Internet Protocol*, Atkinson, R., 1995.

[RFC 1826] *IP Authentication Header*, Atkinson, R., IETF, 1995.

[RFC 1827] *IP Encapsulating Security Payload*, Atkinson, R., IETF, 1995.

[RFC 2207] *RSVP Extensions for IPSEC Data Flows*, Berger, L., O'Malley, T., IETF, 1997.

[Schneier 1994] Schneier, B., *Applied Cryptography – Protocols, algorithms, and source code in C,* John Wiley & Sons, Inc. 1994, ISBN 0-471-59756-2.

[Schneier/BF 1994] Schneier, B., *The Blowfish Encryption Algorithm*, Dr. Dobb's Journal, April 1994.

[Schneier et al. 1997] Schneier, B., Wagner, D., *Analysis of the SSL 3.0 protocol,* The Second USENIX Workshop on Electronic Commerce Proceedings, USENIX Press, November 1996, revised 1997. http://www.counterpane.com/ssl.html

[Shannon 1949] Shannon, C., *Communication Theory of Secrecy Systems*, Bell Systems Technical Journal, No. 4, 1949

[Stallings 1999] Stallings, W., *Cryptography and Network Security – Principles and Practice 2nd ed.,* New Jersey: Prentice-Hall Inc. 1999, ISBN 0-13-869017-0.

[Steel 2000] Steel, Christopher, *Constructing a secure distributed architecture,* Java Report, January 2000.

[Sun/1 1999] *Java Remote Method Invocation – Distributed Computing for Java*, Sun Microsystems, Inc., 1999. http://java.sun.com/marketing/collateral/javarmi.html (Accessed 2000-05-16)

[Sundsted 1999] Sundsted, Todd, *In Java we trust*, Java World, January 1999.

[Tanenbaum 1996] Tanenbaum, A. S., *Computer Networks 3rd ed.*, New Jersey: Prentice-Hall Inc. 1996, ISBN 0-13-394248-1.

[Thomas 1998] Thomas, Anne, *Enterprise JavaBeans Technology - Server Component Model for the Java Platform,* Patricia Seybold Group, prepared for Sun Microsystems, 1998. http://www.java.sun.com/products/ejb/white_paper.html

[Vogel et al. 1999] Vogel, A., Rangarao, M., *Programming with Enterprise JavaBeans, JTS and OTS*, John Wiley & Sons, Inc. 1999, ISBN 0-471-31972-4.

Appendix A: Terminology

A.1 Abbreviations

3DES	Triple-DES
CTM	Component Transaction Monitor
DES	Data Encryption Standard
DBMS	Data Base Management System
EJB	Enterprise JavaBean
GUI	Graphical User Interface
Ida	Ida Systems Ab
IDEA	International Data Encryption Algorithm
IETF	Internet Engineering Task Force
IP	Internet Protocol
IPSec	IP Security protocol
JCE	Java Cryptography Extension
KDC	Key Distribution Center
LAN	Local Area Network
MAC	Message Authentication Code
Mbps	Mega bits per second
OSI	Open Systems Interconnection
PAX-E	PAX-Enterprise
PAX-NG	PAX-Next Generation
RFC	Request For Comment
RMI	Remote Method Invocation
RSA	Rivest-Shamir-Adelman
SSL	Secure Socket Layer
TP monitor	Transaction Processing Monitor

A.2 Glossary

3-tier architecture A model for distributed computing that divides the application into three separate layers: client-layer, server-layer and data storage-layer. In a 3-tier architecture all the business logic and possibly also logic for presenting a GUI is executed on the server.

Application layer A layer in the *OSI* reference model that hosts the protocols that explicitly deals with network communication such as HTTP, SMTP etc.

Asymmetric-key algorithm See *Public-key algorithm.*

Avalanche effect A small change in the plaintext or the key in a cipher algorithm produces a significant change in the ciphertext (for example Feistel networks generate an avalanche effect).

Brute-force attack The method of breaking a cipher (when the encryption algorithms is known) by testing all possible keys in the key-space. Sometimes referred to as exhaustive key search.

Cipher text The output from a cryptography algorithm. A cipher holds the characteristic that it stores some information in such a way that it should be as difficult as possible for someone not intended to read the information to do that.

Component Transaction Monitors (CTM) A combination of transaction monitors and object request brokers (ORB) which are used for maintaining server-side components in a distributed architecture.

Cryptanalysis The process of trying to decrypt a cipher without access to the key.

Feistel network A Feistel network is a design principle for block ciphers. The Feistel network alternates substitution and permutation in many rounds to achieve confusion and diffusion of the plaintext.

Internet Engineering Task Force This is a non-profit organisation that governs and proposes new standards on the Internet. See www.ietf.org

Network layer The layer in the *OSI* reference model that makes sure that networks with different addressing and data packing schemes are able to be interconnected. A common transport layer protocol is the Internet Protocol (IP).

Open distributed environment A system running in an open distributed environment is called an *open distributed system* (see that).

Open distributed system A system consisting of several different applications running on different machines distributed geographically and connected via a *public network*.

Open Systems Interconnection (OSI) reference model A theoretical model proposed by the International Standards Organisation that presents a standard protocol stack for interconnecting open systems. The model is divided into seven layers, where each layer hosts one specific protocol. The layers are: Physical layer, data link layer, *network layer*, *transport layer*, session layer, presentation layer and *application layer*.

Public-key algorithm An encryption algorithm that enables two parties to send encrypted messages to one another without sharing a common key. Each party maintain two keys – one public and one private. When for example A want's to send a secret message to B, he encrypts the

massage using B's public-key. The message can then only be decrypted using B's private key. For example RSA is a public-key algorithm.

Public network A computer network accessible for the public as opposed to a private network that is exclusively accessible for a limited number of people, for example employees within a company. Internet is an example of a public network.

Plain text The opposite of *cipher text*.

Remote Procedure Call (RPC) A protocol that one program can use to request service from another program located on a different computer in a network.

Request For Comment Technical reports within the Internet Engineering Task Force (the organisation that propose new standards within the Internet) are called Request For Comment (RFC). These are available online: www.ietf.org/rfc.html

Security service A service that provides a suite of algorithms for encryption, authentication, integrity checks and key exchange.

Symmetric-key algorithm An encryption algorithm that requires that both the sender and the receiver of an encrypted message have access to the same key (for example DES and IDEA).

Transport layer The layer in the *OSI* reference model that provides end-to-end communication, with error recovery and flow-control. A common transport layer protocol is the Transport Control Protocol (TCP).

Appendix B: Theory on Security Methods

This appendix presents an overview of the encryption and authentication algorithms referred to in the thesis.

B.1 Symmetric-key Cryptography

The following is a description of four symmetric-key algorithms. That is, encryption algorithms that require the two communicating parties to have access to a common secret key. Four of the algorithms are block-ciphers and the fourth is a stream-cipher.

B.1.1 DES

DES (Data Encryption Standard) is one of the more renowned encryption algorithms. IBM developed DES in the early 1970s. In 1977 it was adopted a standard for non-security classified information within the U.S. Government after some modifications dictated by the U.S. National Security Agency (NSA). Originally IBM proposed Horst Feistel's encryption algorithm Lucifer [Feistel 1973] as a standard, but NSA argued against this and recommended IBM to develop a new algorithm with a reduced key size from Lucifer's original 112 bits to 56 bits. Thus, you can say that DES is a relaxed variant of Lucifer.

Overview of the algorithm

DES is a block cipher based on the Feistel network. As mentioned above the key size is 54 bits and the block-size is 64 bits. The algorithm is designed to combine two principles of encryption: confusion and diffusion. This is done by attacking the plaintext block in 16 different rounds as shown in Figure 21 and by using both permutation and substitution in the encryption function. DES exhibits a strong avalanche affect.

How safe is it?

Since its creation there has been questions about the level of security provided by DES. It is a general opinion that the key size of DES was limited at 56 bits so that the cipher would be strong enough to hold against security attack by individual people but weak enough to enable big government agencies (such as NSA) to break the cipher through brute-force attacks.

Because of the limited key-space it is today relatively simple to break a DES-cipher if you have access to a big enough computational power. For example a €10,000,000 machine can break a DES-cipher in 21 minutes [Stallings 1999].

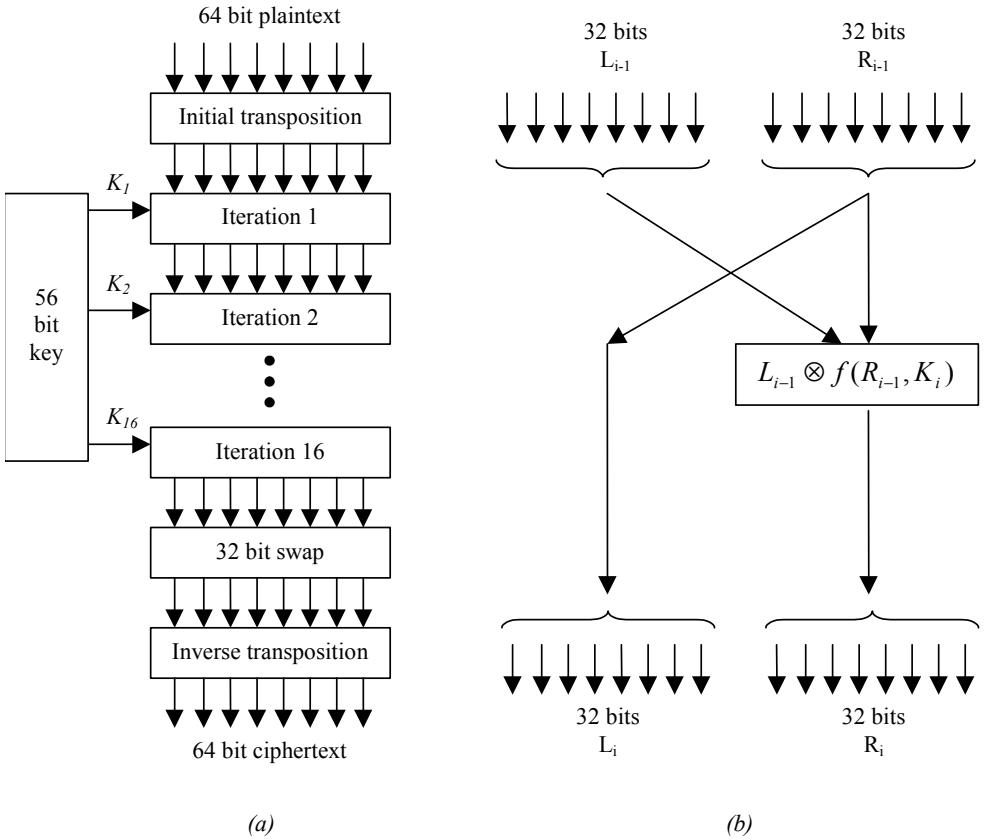


Figure 21 DES. (a) General outline. (b) Detail of one iteration [Tanenbaum 1996].

How fast is it?

DES was originally designed to be implemented only in hardware, and is therefore extremely slow in software.

3DES

Because of the relative weakness of DES, a new variant of DES called triple-DES (3DES) has been proposed. In 3DES the plaintext is encrypted three times using the ordinary DES algorithm with two or three different keys. Two keys raises the cost of a brute-force attack to 2^{112} which is beyond what is practical now and in a foreseeable future.

B.1.2 IDEA

IDEA (International Data Encryption Algorithm) was developed by Xuejia Lai and James Massey in Switzerland in 1991 and is generally considered one of the best and most secure block-algorithms. Moreover, it is not troubled with any government interference, like DES. In recent years IDEA has been proposed as a

replacement to DES. For example IDEA is used in the PGP-algorithm (Pretty Good Privacy) for achieving mail secrecy.

Overview of the algorithm

Like DES IDEA is a block cipher based on the Feistel network. The design philosophy behind the algorithm is one of mixing operations from different algebraic groups. IDEA uses three different operations (exclusive-or, addition modulo 2^{16} and multiplication modulo 2^{16}) making cryptanalysis much more difficult than with an algorithm such as DES, which relies solely on exclusive-or. See Figure 22 for an overview of the algorithm.

How safe is it?

IDEA takes 64-bit blocks as input and the key size is 128 bits which should protect it against brute-force attacks for a foreseeable future. An earlier version of IDEA was said to be open for a cryptanalytic method called differential cryptanalysis, but the presently available version of IDEA should be immune against such attacks.

How fast is it?

IDEA is considered a relatively fast algorithm. Encryption speeds of 9 Mbps should easily be achievable in software running on a modern PC [Tanenbaum 1996].

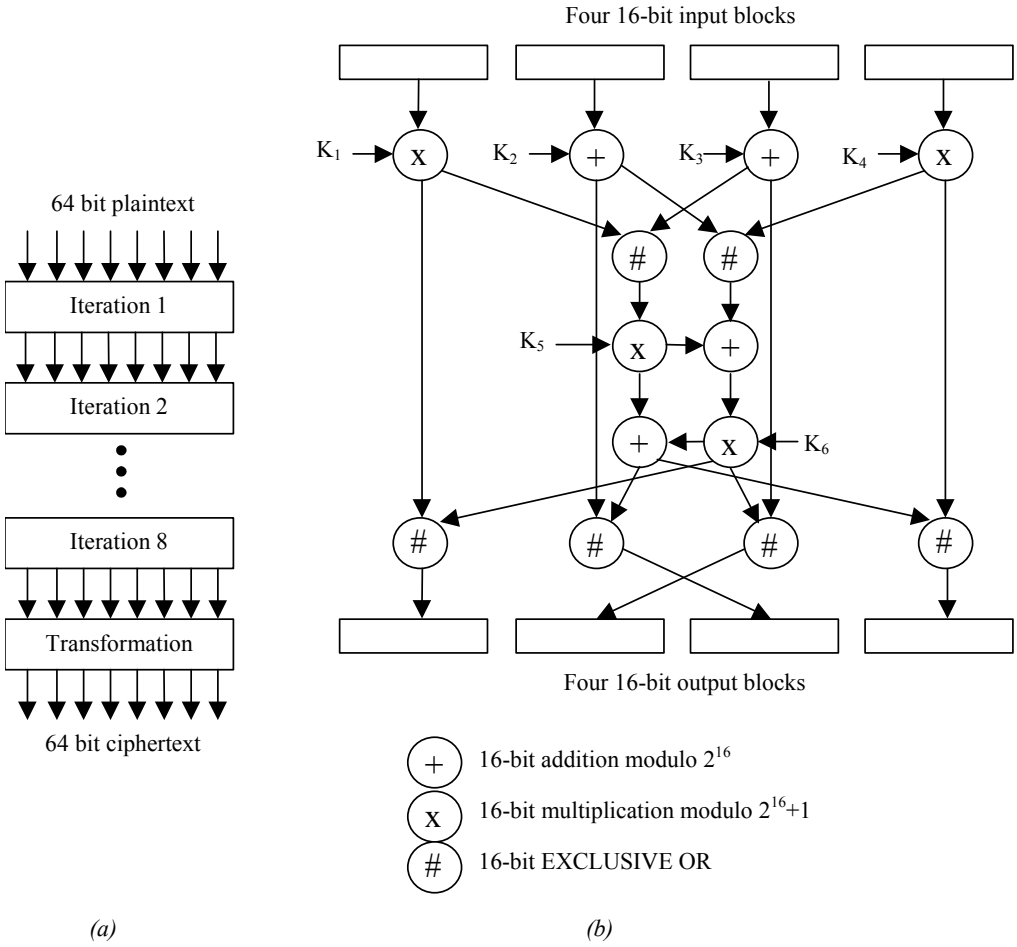


Figure 22 IDEA. (a) General routine. (b) Detail of one iteration [Tanenbaum 1996].

B.1.3 Blowfish

Blowfish is a symmetric block cipher developed in 1993 by Bruce Schneier [Schneier/BF 1994] and is by many described as the best security algorithm available today. It is designed to have the following characteristics:

- **Fast** (Blowfish encrypts data on a 32-bit microprocessor at a speed of 18 clock cycles per byte)
- **Compact** (Blowfish can run in less than 5K in memory)
- **Simple** (Blowfish is easy to implement)
- **Variable security level** (The key size is variable from 32 bits to 448 bits).

Overview of the algorithm

Like the algorithms mentioned above, Blowfish is based on the Feistel network with a block size of 64 bits. Unlike DES and IDEA in Blowfish the entire block

(both the left and the right half) is effected in each round. There are 16 rounds in the algorithm and the structure of one round is displayed in Figure 23. The algorithm uses two primitive operations (addition modulo 2^{32} and exclusive-or) to make cryptanalysis more difficult.

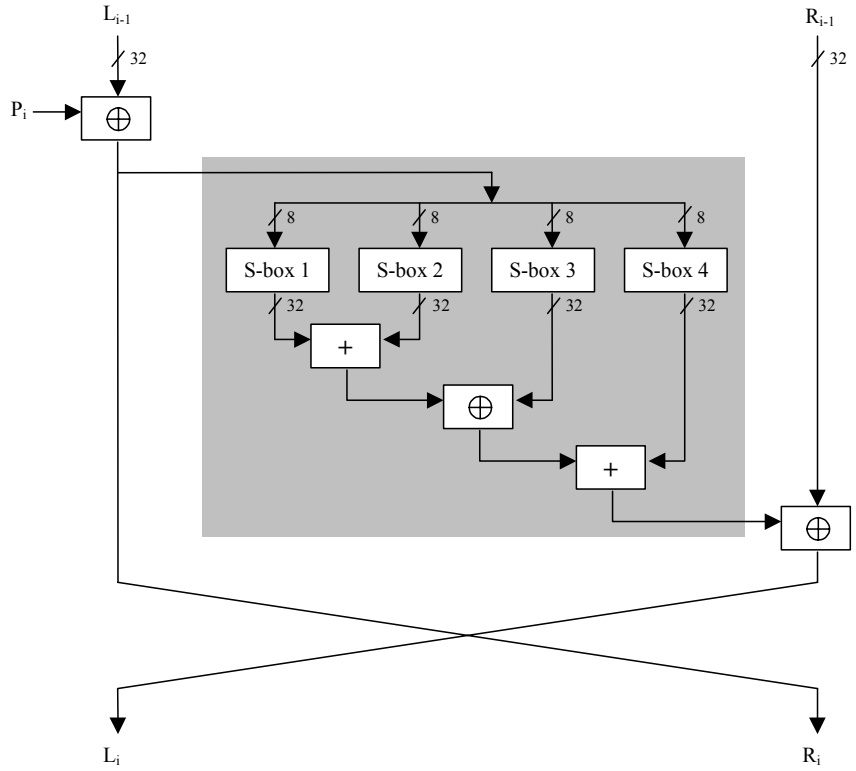


Figure 23 One round in the Blowfish algorithm [Stallings 1999].

How safe is it?

So far, there have been a few published papers on Blowfish cryptanalysis, but no practical weaknesses have been found. Thus, the only available method of breaking a Blowfish cipher is to perform a brute-force attack and considering that a key size of 448 bits is allowed, the algorithm can be considered to be literally invulnerable.

B.1.4 RC4

RC4 is a stream cipher developed by Ron Rivest of RSA Laboratories. RC4 used to be a trade secret of RSA Laboratories, but an anonymous person published the source code to the algorithm in 1995. Thus, RC4 is now free for public use.

Overview of the algorithm

The RC4 algorithm is essentially a pseudo random number generator that takes a shorter key as input and generates a key sequence with the same length as the message that is to be encrypted. The message and the key sequence is then bit-by-bit XORed. The algorithm accepts key-lengths ranging from 40 bits to 448 bits.

How safe is it?

There have been many reports on successful attempts to break the 40-bit RC4 cipher using large numbers of computers connected in networks. In 1996 it was reported that an MIT student was able to brake a 40-bit RC4 cipher in eight days on a single $83,000 graphics machine. Thus, the weaker 40-bit algorithm can not be considered suitable for encrypting highly classified information. The RC4 algorithm in itself though is considered secure, so provided that a longer key is used, the algorithm can be considered safe.

How fast is it?

RC4 is known as one of the fastest encryption algorithms around. Because of its speed, RC4 is often used as default encryption algorithm in implementations of SSL.

B.2 Public-key Cryptography

The following is a description of two algorithms for public-key encryption.

B.2.1 Diffie-Hellman

The concept of public-key cryptography was invented in 1976 by a group of researchers at Stanford University: Whitfield Diffie, Martin Hellman and Ralph Merkle.

Diffie and Hellman developed the first useful public-key algorithm; an algorithm used for exchanging keys over an unprotected network. The algorithm is based on discrete logarithms, which display the characteristics of one-way functions.

Overview of the algorithm

The algorithm is very simple. If for example Alice want to communicate with Bob, the protocol goes as follows:

1. Alice and Bob first agree on two numbers *n* and *g*, such that *g* is less than *n* but greater than one. The two numbers don't have to be kept secret and can be communicated over a public network.
2. Alice chooses a random large integer *x* and computes $X=g^x \bmod n$

3. Bob chooses a random large integer y and computes $Y=g^y \bmod n$

4. Alice sends X to Bob and Bob sends Y to Alice.

5. Alice computes $k=Y^x \bmod n$

6. Bob computes $k'=X^y \bmod n$

Both k and k' are equal so Alice and Bob has now agreed on a common key to use in further communication (using for example a block cipher).

How safe is it?

The security of the Diffie-Hellman key exchange lies in the fact that it is very difficult to calculate discrete logarithms. For large primes, the task is considered infeasible.

B.2.2 RSA

The Diffie-Hellman algorithm is a key-exchange algorithm and was not designed to be used as a so called bulk-cipher (i.e. to encrypt a stream of data in a connection). RSA was the first all-purpose public key encryption algorithm. It can be used for key-exchange and bulk-ciphering as well as for message authentication.

RSA is named after its three inventors Ron Rivest, Adi Shamir and Leonard Adleman, who first introduced the algorithm in 1978. It has since withstood years of extensive cryptanalysis. The RSA algorithm is based on the difficulty in factoring large numbers. The private and public keys are functions of a pair of large (100 to 200 decimal digits or larger) prime numbers. Recovering the plaintext from one of the keys and the ciphertext is equivalent to factoring the product of the two primes [Schneier 1994].

Overview of the algorithm

The algorithm is quite simple. Both parties in a communication generates two key-pairs according to the following protocol:

1. Select two large prime's p and q

2. Calculate $n=p*q$

3. Calculate $\phi(n)=(p-1)*(q-1)$

4. Select an integer e such that

$$\gcd(\phi(n),e)=1, \quad 1<e<\phi(n)$$

5. Calculate $d=e^{-1} \bmod \phi(n)$

The public key is now the pair [e, n] and the private key is the pair [d, n]. To encrypt a text, divide it into blocks M_i with size less than n and calculate

$$C_i=M_i^e \bmod n$$

To decrypt the cipher, simply calculate

$$M_i = C_i^d \bmod n$$

How safe is it?

According to Stallings [1999] there are three approaches to attacking RSA ciphers:

- **Brute-force attacks**. This simply involves trying all possible keys. The success of this approach depends on the key size, which is not defined in RSA. In 1994 a distributed network of computers broke a RSA cipher with a key size of 129 decimal digits (448 bits). A key size of 200 decimal digits (664 bits) can be considered unbreakable for many years to come, as far as brute force attacks are concerned.
- **Mathematical attacks**. This involves trying to find a fast way of factoring the product of two large primes. Even though it hasn't been proven that there doesn't exist a fast way to do perform prime factorisation, mathematicians have been working on this problem for more than 300 years.
- **Timing attacks**. This is a new approach in attacking ciphers that shows how difficult it is to assess the security of cryptographic algorithms. It was presented as late as in 1996 by Paul Kocher. Kocher displayed that a snooper could determine the key size used in a cipher (and thus limit the search space for a brute-force attack) by estimating how long time it takes for a computer to decipher the message. Introducing a random delay when encrypting and decrypting messages can eliminate this security flaw.

How fast is it?

Because of the heavy calculations, the RSA algorithm is very slow. A rule of thumb says that in hardware it is 1000 times slower than DES and in software 100 times slower than DES. Therefore RSA is often used simply for key-exchange, just like Diffie-Hellman, and for digital signatures.

B.3 Message Authentication

The following is a description of two algorithms for generating message digests.

B.3.1 MD5

MD5 is a message-digest algorithm developed by Ron Rivest at MIT. MD5 was the most widely used hash algorithm until a few years ago, when concerns were raised on the security against brute-force attacks and cryptoanalytic attacks. However MD5 is described in a RFC (Request for Comment) by IETF so it remains heavily used on the Internet.

Overview of the algorithm

The algorithm takes as input a message of arbitrary length and produces a 128-bit message digest. The algorithm works by mangling the bits in a sufficiently complicated way so that every output bit is effected by every input bit. The data is processed in four rounds where each round consists of 16 non-linear functions.

How safe is it?

As mentioned earlier, there have been some doubts about the security of MD5. Advances in computing power and hash function cryptanalysis has led to a decline in the popularity of first MD4 and then MD5 in favour of newer hash functions with longer hash-codes and with features designed to resist specific cryptoanalytic attacks.

B.3.2 SHA-1

SHA-1 is a message-digest algorithm that was developed by NSA as part of the Secure Hash Standard (SHS). SHA-1 is based on MD4 and is very similar to MD5.

Overview of the algorithm

The algorithm takes a text of arbitrary length and produces a 160 bit long message-digest. Just like MD5, the data is processed in four rounds. Each round consists of 20 non-linear functions.

How safe is it?

The primary difference between MD5 and SHA-1 is the increased key-length from 128 bits to 160 bits. This should protect the algorithm from brute-force attacks for a foreseeable future. As opposed to MD5, SHA-1 does not appear to be vulnerable to cryptoanalytic attacks. However little is publicly known about the design of SHA-1, so the strength is difficult to judge.


idasystems

Ida Systems Ab
Box 576
SE-581 07 Linköping
Sweden

Tel: +46 (0)13 37 37 00
Fax: +46 (0)13 37 37 90
E-mail: info@idasys.se
Internet: www.idasys.se