A Real Time, System Independent, Secure, Internet Based Auctioning System.

Cuan Brown

Submitted in fullfillment of the academic requirements for the degree of MSc (Information Security) in the School of Geological and Computer Sciences, University of Natal, Durban

May 2000

ABSTRACT

This thesis outlines the creation of a secure, real time, system independent, Internet based auctioning application. The system has been developed to meet the needs of today's stringent requirements on secure Internet based applications. To attain this goal, the latest cryptographic algorithms and development platforms have been used. The result is a JAVA based server and client auctioning application. The client application is designed to run in any common web browser, and the server to execute on any JAVA enabled operating system with a web server and Internet connection. The real time system uses a relatively secure hybrid cryptosystem for communication. This involves the use of RSA for secure key exchange, and RC6 and MARS for secure communication.

PREFACE

The work described in this dissertation was carried out in the School of Geological and Computer Sciences, University of Natal, Durban, from February 1998 to May 2000, under the supervision of Dr. Hugh Murrell and co-supervision of Prof. Henda Swart.

The studies represent the original work by the author and have not otherwise been submitted in any form for any degree or diploma to any tertiary institution. Where use has been made of the work of others it is duly acknowledged in the text.

CONTENTS

ACKNOWLEDGEMENTS

I would like to thank my parents and grandfather for giving me the opportunity to study as far as I have. I would also like to thank Chantal Friart for all the encouragement and putting up with me when I was difficult. Mark Lewis for always pushing me that much further as well as the intellectual and moral support and Hugh Murrell whose persistence and guidance helped me through this beast.
Thanks to Henda Swart for her financial and moral support.
TGIO!

1) Introduction

The Internet thus far has made a significant impact on the world we currently live in and will undoubtedly form an integral part of the future of the entire planet. In a day and age when information is the key to many fortunes, it is imperative to control and distribute this commodity effectively. The Internet has provided the basis of this information sharing, from a simple email to inter corporation co-operation.

Over the past ten years the growth has been nothing short of astounding, with millions of new users joining each month. This is the beginning of a borderless global economy.

However, with the advent of the most resource intensive tool on the planet, comes a number of problems. The Internet has been plagued by the disparity of systems utilized. This includes the non-adherence to specifications of applications and formats. The result is a complex integration of architectures, operating systems and protocols. This constant state of flux is argued as one of the strengths of the Internet, but also, its greatest inhibitor.

Another problem with this sharing of information, is not the means of communication, but the actual information concerned. With such vast amounts of sensitive information available, it becomes a crucial resource to protect. The "Digital Age" has been entered; all and most forms of information have made the conversion to some form of digital format. However there has been little forethought to the securing and protection of this digital information. The safety deposit box of the twentieth century needs to be replaced by the secure server of the twenty-first century.

Currently wherever one looks, there are articles, adverts and programs expressing the promise that the Internet holds. Over the last two years there has been a wild rush to become net ready, opening the doors to web shoppers and commercial success. The Internet has levelled the commercial playing fields and allowed the average company to parade their wares on an international stage. This e-business as it has come to be known, generated about two billion U.S dollars in 1999. It is estimated that the amount spent online will increase to twenty billion U.S dollars by the year 2002. The ease at which business can be conducted online will push more and more corporations into the fray, any of whom that ignore the biggest new market in over four hundred years, do so at their own peril.

1.1) Proposed project

The proposed project is as follows:

To create a real time, auctioning application. The application must have no preference to any architecture and operating system. The aim was for a single item to be auctioned at a time, with the possibility of extending the application at a later date for multiple items. The server application controls all aspects of the auction, including details, finish times and the client handling elements. This includes any administration tasks associated with the application. The entire system is aimed at working over a network, specifically the Internet.

The bidding on auctioned items is to be orchestrated by the client application. This client application will communicate with the auction server over the Internet. The client application notifies the client, concerning all changes in the current auction. The client application allows the client to place bids. Similarly no client type or operating system has been targeted. This requires an application able to operate on any of the wide scope of systems available on the Internet today.

The application is also designed to take advantages of the future in secure communications and so affording the clients peace of mind when participating in the auctions. The security of the application is not exclusive to secure communication, but rather as an aspect of the entire application. This security feature incorporates, design techniques, secure communication and secure programming.

The language used to develop this application must take advantage of the Internet, and be able to support the wide range of features available. However it must not limit or target any platform available on the web. It should be a robust, secure and multithreaded development tool to prevent any limitations to the application.

The project will attempt to create a marketable application that would match real world demands. The application is designed to harness the power of the Internet, without limiting the clients to any hardware or software requirements, thereby obtaining a wider range of clients. Not only that, but the application is designed to meet the requirements of secure Internet based communications, allowing commercial transactions to take place without threat.

1.2) The Internet

A Brief History

The Internet is now over thirty years of age. Admittedly it has changed dramatically, but its roots lie shrouded in the cold war. In the 1960's nuclear threat was a real possibility, and the American government developed a communications scheme that would be capable of withstanding such a strike. The result was a network of computers that had multiple transmission routes to anywhere else on the network

The communication protocol that was designed to take advantage of this was TCP/IP. This enhanced the system by adding reliability to the way in which messages were processed. In the late sixties the defence department connected to a number of universities around the States. This was the beginning of the ARPANET (Advanced Research Projects Agency Network). In the early seventies the ARPANET hosted more than fifty universities and defence departments. This network allowed academics to quickly and easily share information amongst colleagues.

However in the early eighties the ARPANET has out grown itself, and the defence department wanted to limit access to this network. This resulted in the two networks splitting up. The main problem was that there was no real support for this network in most operating systems. This resulted in a free operating system being developed called UNIX, by the University of California. This o/s had natural support for TCP/IP and allowed anyone to connect to the available networks. The Internet was born.

However this network was still primarily an academic tool. It was not until the late eighties and early nineties that the Internet started to become the force that we know it today. The reason for its huge success was the development on a new mark-up language called HTML. This was developed by the CERN (Collective of European high-energy physics researchers) organization. The result was met with immediate support and applications were developed to support this fantastic new medium of data. These included the likes of Mosaic, Netscape and later Internet Explorer. However the Internet was very firmly separated, and by mid nineties no company or operating system enforced any dominance over the Internet.

Growth and Opportunities

The Internet started out with ten hosts in the early seventies, and has now grown to astronomical proportions. In the early eighties the number of hosts was as little as 1000, by the late eighties that figure was over 100 000 hosts. Ten years later that figure has grown to over fifty million hosts world-wide [R.Zakon, 1999, online].

This dramatic increase serves to demonstrate the popularity of the Internet. However it was primarily the introduction of the World Wide Web (WWW) that lead to this huge explosion. In fact, by 1993 web traffic had increased annually by over three hundred thousand percent. As can be seen by the diagram, the number of sites on the Internet is growing at an exponential rate.

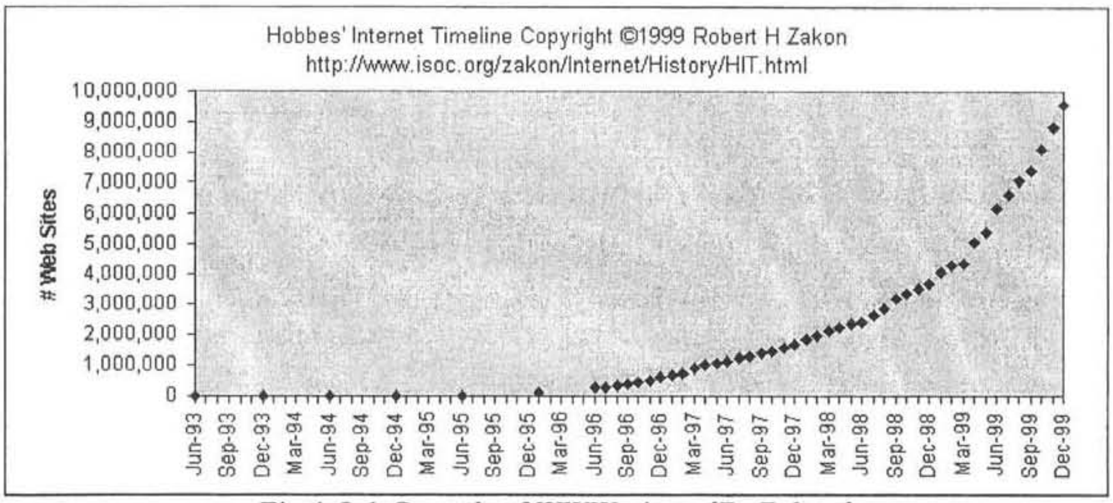


Fig 1.2.1 Growth of WWW sites. [R. Zakon]

It is a difficult to estimate the amount of people on-line, and the figures have been contentiously debated. However most statistics show that there will be over one hundred and fifty million web users in the year 2000. The likes of such a huge target audience have never been seen before and with this amount of potential, it cannot be ignored as a new medium of distribution for retailers.

The Internet was widely considered as an academic and glorified public relations effort by most companies in the early nineties. However a few insightful people managed to start making fortunes, and lead the charge of Internet based companies. The likes of Amazon.com and Yahoo.com (Both been valued over one million U.S dollars) have shown what is capable of been achieved. It is not only the public retailers that will benefit, but also business to business transactions. The Internet has been transformed into a data network that facilitates inter business communication [S.M.Bellovin 1998].

Dell computers has reported earnings of over one million U.S dollars a day from its corporate market share. The average corporate order in the U.S takes fifty odd U.S dollars to process. Using the Internet as a distribution medium, that average corporate Internet order is now below five U.S dollars. This results in cost reduction, quicker order processing and improved customer services.

The Players

The fight for dominance or control of the Internet is an ongoing struggle and will likely never be resolved. The two major players are Microsoft and Unix. A discussion of the pros and cons has been highlighted in the media on numerous occasions. Ultimately it will not come down to the operating system that one uses, but rather the application that allows people to enjoy the benefits of the Internet.

Thus far the web browser is a distinctive winner. However the war between web browsers is an ongoing affair, Netscape, Internet Explorer (Microsoft) and a number of other upstarts are what it all boils down to. As the Internet can undergo entire changes in a matter of months, the outcome of this battle is not yet settled.
One of the major problems associated with this battle is the complete disregard for international specifications. The various players have implemented HTML, DHTML and XML in different manners. This leads to confusion and difficulty in the

maintenance of most web sites. However once standards are met, the web browser will be nothing more than a name brand facilitating the Internet experience.

The Future

Although the Internet has been heralded as the new digital age, it will still take awhile for all these benefits to be seen. Some of the fantastic new features that will impact the world are the connection of personal data assistants (PDA's), cellphones and other electronic goods to the Internet. This will allow one to buy groceries, do banking and other Internet related services, while travelling at home or sitting in an armchair. The potential markets are enormous.

The Internet promises enormous amounts, however there are a number of issues that need to be surmounted. The primary issue is the dominance of operating systems and web browsers. The Internet is divided by this issue, not only on opinion but actual usage. The result is a target market that is divided, and hence difficult to entice. The solution is a common standard and format that will alleviate the interoperability problems of the past.

1.3) Security

Secrets have been part of history, whenever there has been sensitive information, secrecy follows. Since the Roman times different methods have been used to ensure the secrecy of messages. For example placing messages on messenger's body parts, which are then subsequently hidden from view. Fortunately these techniques are not used in today's digital world, however crucial information still needs to be protected to maintain a competitive edge. The capitalists of today have replaced the dictators of yesteryear, but both have a need for secrecy and protection of their vital information.

The question is asked why has the Internet not taken over all monetary transactions? If it is so useful and geared for digital communication what is the hold-up? Thus far one of the contributing factors to slow growth of Internet based transactions is the security factor. The awareness of problems involved with sensitive data communication has been intensive. People will happily give credit card numbers over the telephone, or to waiters in restaurants, but very few will do so on the Internet even though there is a reasonable security set-up in place. This psychological barrier is going to be harder to break than the actual security systems in place.

Today, there are millions of packets of information travelling around the Internet. Not only are the chances slim that an undesirable will obtain the information required, but the opportunity cost of actually breaking the encoded message is invariably not even profitable. However this opinion must not lead to an apathetic attitude about security. Until the early nineties there was very little secure communication, however there are a number of reasonable frameworks in place today. These include Secure Socket Layers (SSL) and Secure Electronic Transactions (SET). The problem, is really the fact that security of e-commerce is not only based on secure communications, but trustworthy parties, applications and procedures as well. Due to the anonymity of the Internet, a few problems are posed:

Can I trust this party?

Is the party I am dealing with who they claim to be. Currently the use of digital certificates rectifies this problem. A digital certificate is issued by a Central Authority (CA), and will certify the authenticity of the party involved.

Is our communication secure?

The like of SSL takes care of secure communication between a client web browser and a server. It is certainly not an optimal solution as it is based on dated algorithms and was meant to be a temporary solution.

The major problem here is a small term called "computational security". This means that it will be infeasible, in monetary and time constraints, for an undesirable to break the crypto-system with today's resources. One must bear in mind that according to Moore's law, the power of computing doubles every eighteen months. With such exponential growth, crypto-systems will have to adapt as quickly as the hardware used to implement them. A brief example of this is the DES crypto-system. Its inception in the seventies, and support by most banking institutions world-wide, has lead to it being a standard. Less than thirty years later the system has been broken on a one hundred thousand U.S dollar machine, and yet it still remains a banking security standard. A little worrying to say the least.

These issues are the driving factor for the continuous development and modification of new crypto-systems. Until a superior and unbreakable crypto-system is created, if ever!, it is still a necessity to escalate the quality of security. Due to the vast amount of money dealt with in e-commerce, this continuos re-assessment and progress in the crypto world will continue. In the light of this it is important to study, analyse and finally approve or disapprove the new algorithms.

It must be stressed though that security is not just the algorithm used to encrypt or decrypt information. Security is also dependent on the implementation of the algorithm. It takes only one weak link in the final application to potentially open up the system to an intruder. The worst must be expected to happen, and when it does, the product must be extended to overcome any difficulties or weakness.

1.4) Conclusion

One could easily be mislead into thinking that the Internet will control all aspects of our lives very soon. This could be a reality, but extensive amounts of integration and development is still required. Undoubtedly though the Internet will impact on any digital forms of communication and transactions. The possibilities are endless, the infrastructure is set-up and the cost effectiveness of this resource cannot be ignored. Profitable business is abundant due to the huge target market afforded by the Internet.

There are a few problems that need to be resolved to make the entire process run more efficiently; Addressing the interoperability barriers caused by the diversity of the Internet systems. Overcoming the security issues involved with the transferring of sensitive information over this medium, as well as other security issues regarding implementation. Once these have been addressed it will allow the full potential of the Internet and its benefits will be attained.

The following section will examine the criteria laid out for the thesis.

2) The Application Criteria

2.1) Overview

As has been previously stated the goal of this thesis is to create a "*real time, system independent, secure, Internet based auctioning application"* i.e. an application that uses the Internet, which everyone can access and ensures secrecy. It is easiest to discuss each of these criteria and give a clear picture of what each requirement entails.

2.2) A Real Time System

A real time system is one in which communication and action takes places immediately. A good example of a real time system is an ATM bank teller machine. Commands are issued and responded to almost immediately. The goal is to devise an application that user's could use and see the results of, as the application is executed.

2.3) Internet Based Application

As some form of client/server real world application is to be used, a communications type is required. There are a number of networks available, however none have quite made the impact the Internet has. The application therefore is required to use the most dynamic network available on the planet. The reason for using the Internet, is that it has huge fiscal potential and an enormous client base.

2.4) System Independent

As the application is to work over the Internet it will expose itself to the widest range of operating systems, hardware and software. This diversity therefore makes it rather difficult to target a particular audience or user. Many applications are designed and targeted for specific operating systems (i.e. Microsoft, UNIX), and insisting that the application is system independent ones target audience increases dramatically.

2.5) Security

The importance of security has increased dramatically over the last few years, especially with a number of incidents being profiled by the media. As there is communication between the client and the server application, the very nature of the information exchanged it makes it necessary to provide security.

This security has to incorporate two aspects:

- Secure communication
- Application security

Each of these cannot be academically viewed, but must be practically implemented within the confines of the application. The security model implemented must minimise network bandwidth, and hardware resources (CPU usage, Memory Usage), and must prevent any risks to the client. All of these are reasonable real world application criteria.

Above all, the security must not hamper the performance or practicality of the application. The secure communication has to be handled in such a way that it is effective and actually workable in a real world environment. Secure communication is not all that's required, rather it is a part of the overall security. A secure communications channel is useless if the applications communicating are vulnerable. Hence the requirement that the application itself must be relatively secure, thereby ensuring a global security.

2.6) Auction Application

The application created needed to resolve a real world issue, thereby giving it some credibility. Recently, Auctioning sites on the Internet have come into their own (Ebay.com and bidorbuy.co.za), but they are plagued with numerous problems. The first is that there are large amounts of fraudulent happenings and secondly many auctioning applications are not of a real time nature. The application needs to adhere to the latest trends, such as object orientated programming, thereby making it reusable and scalable.

This thesis will attempt to resolve these problems and in doing so create a viable real world auctioning application.

2.7) Conclusion

By defining the requirements and scope of the thesis, a viable and practical solution can be worked on. The remainder of this thesis will examine the proposed application and the implemented solution that meets the criteria discussed above.

The following section examines Java and explains why it was chosen as the implementation vehicle.

3) Java

To tackle the auctioning problem it was decided from the very outset that a unique programming environment would be needed. It would have to be a stable, multithreaded, networked and most importantly a cross platform environment. Fortunately Java has all those features and more. A number of the benefits of the language will be highlighted and discussed to enlighten the reader as to why Java was chosen to accomplish the job.

3.1) What is Java?

"Java--an *architecture-neutral* and *portable* programming language--provides an attractive and simple solution to the problem of distributing your applications across heterogeneous network-based computing platforms. In addition, the simplicity and *robustness* of the underlying Java language results in higher quality, reliable applications in which users can have a high level of confidence"

Sun Microsystems, java.sun.com

Java is essentially a C/C++ like language, and in fact draws a lot of its strengths from C/C++. However there are some fundamental differences, which shall be revealed.

3.2) How does Java work?

It is important to understand how the Java system operates. As indicated earlier Java is hardware and operating system independent. As the creators claim, it is architecturally neutral. However the system executing the application must have the Java Platform, which is known as the Java runtime environment (Jre). This is made up of two basic parts, the Java Virtual Machine (JVM), and the Java Application Programming Interface (Java API). The JVM is where all applications execute, oblivious to their underlying operating system. The Java API are the core libraries that will be available to any Java based application. These give the applications their functionality. Thereafter any program developed in pure Java will run anywhere. No special exception, and no special compilers.

When an application is compiled in most languages, it is usually compiled to the machine code of the current operating system. This means that the application will only work on that system. However Java works in a slightly different manner compared to that of C/C++ or other languages. A program is compiled into bytecode. Bytecode is almost an intermediate compilation step. This bytecode is interpreted at run-time by the Java interpreter for the current system. Alternatively Java uses a just-in time (JIT) compiler, that compiles the Java bytecode into native machine code on execution. Any libraries are loaded as they are required from the core Java API.

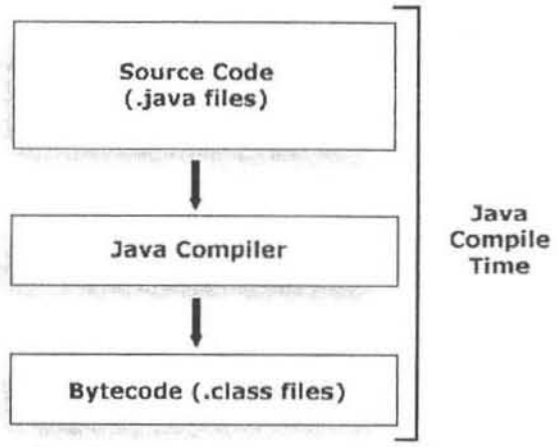


Fig 3.2.1 The Java Compile Time Environment

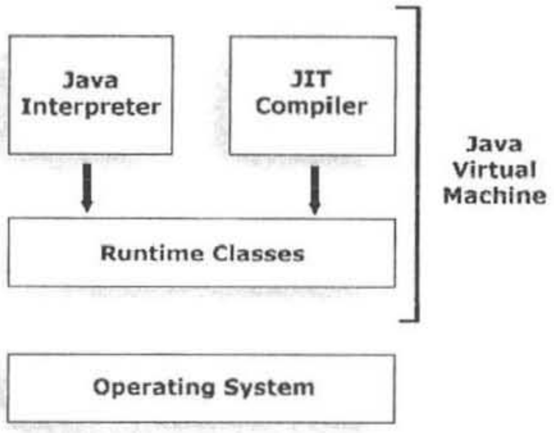


Fig 3.2.2 The Java Runtime Environment

3.3) Object Orientated Programming (OOP)

As the digital age has matured, so have the structures used to create it. Originally the classical programming techniques were sufficient to meet most requirements. However as the computing industry progressed, it needed to simulate more complex real world environments, which classical (procedural or Top down) programming could not support. This brought about the advent of OOP.

"Object-orientated technology is a study of behaviour, how to organize or classify behaviour, and how to assign those behaviours to groups of objects"

Adele GoldBerg

OOP attempts to break the environment down into separate objects, and emulate the interactions between objects; as opposed to classical programming, which attempts to imitate the flow of an environment, from beginning to end. Each object has characteristics and actions. These are the actual properties that each object has in the environment. Each object is a self-contained unit of information and work. The beauty about this modular approach is that code can be re-used. As an example think of a torch. It is made up of the following simple objects: batteries, a switch and light bulb. The batteries have a property determining the electrical charge. Similarly for the switch, its state is either ON or OFF. If all the separate objects are combined and all requirements met by the light bulb object, it will change its own state to display light. The strength of this modular approach allows one to divide and conquer the problem into its tiniest individual elements.

Java uses this idea not only as a programming concept but also, as the construction of the entire language. Everything in Java is an object. To be a truly OOP language, 4 features must be supported, Encapsulation, Polymorphism, Inheritance and Dynamic binding. Encapsulation essentially hides all the inner workings of the objects. Polymorphism is the method by which different object will react differently to the same command or action. Inheritance is the way in which classes can inherit methods and properties from other classes, thereby allowing code reusability. Dynamic Binding allows for object creation to be oblivious to type and position of the object.

3.4) Garbage Collector

The use of all these objects would seem to be quite complicated and so would any form of memory management. However, Java has a feature called the Garbage Collector. It runs inside the Java Runtime Environment (JRE) when the program is executing. When an object is used it will be referenced by some variable. When the variables goes out of scope, or points to another object, the current reference to the object is lost and the object is no longer accessible. This is where the Garbage Collector is called into action. It realises that an object that can no longer be utilised is taking up memory, so it destroys the unneeded object. The Garbage Collector then releases the memory back to the program, which can then be reallocated.

The programmer therefore need not worry about memory management and whether it's safe or not to release objects. The Garbage collector will not destroy objects that still have references to it. This helps eliminate the problem of memory leaks and makes the program more efficient and robust.

3.5) Distributed Computing

Another huge advantage with Java is its seamless integration with a networked environment. Java has integral support for Internet based applications. Not only that but a natural multithreaded ability enhances these capabilities even further.

The reason that Java is splendid for distributed computing is the fact that there is no linking of classes at compilation. Java is dynamic, and only at run time are the class references resolved. The class need not reside on the local machine, but can be pulled off of a server, as and when it's required. Applets are a special form of Java applications. They are designed to run within other applications specifically Java enabled web-browsers. These programs are downloaded from a web server and then the bytecode is executed on the client's machine, inside the web-browsers. The ability of this feature is limited severely for security reasons. However it is a powerful mechanism for running applications over the Internet.

3.6) Security

As Java is designed for a distributed computing environment, it was essential to incorporate security mechanisms into the language. This is needed to protect the client machine from malicious applications. Most of Java's intrinsic securities apply to its network and file streams.

The Java security model is made up of four levels, the Compiler, the Bytecode verifier, the Classloader and the Security Manager.

The Compiler

The reason that Java is a secure language starts with the compiler. It has already been shown that the user need not worry about memory management. Java does not have pointers, which solves many memory and security related problems.

Each class and hence object has a strict access control associated with it. This is important for object creation, usage, and inheritance. The compiler checks all of this before the Java code will be converted to bytecode.

Added to that the Java compiler is extremely strict on syntax, casting and other features of programming. This eliminates run time bugs and makes the language more secure and robust.

The bytecode verifier

The bytecode verifier is the second line of defence in the security model. As the Java runtime environment (Jre) may receive bytecode from anywhere it does not assume that the bytecode is trustworthy. It is not beyond the realms of possibility that a tweaked Compiler was used to produce dangerous bytecode. Therefore all bytecode used is always verified.

The bytecode verifier follows a four-pass process. The verification process firstly verifies the standard format of the bytecode (Checks first four bytes which are 0xCAFEBABE). In pass two it checks classes and objects. In pass three proper bytecode verification takes place making sure there are no disallowed operations. For instance it checks methods arguments to make sure they are correct. It's interesting to note that pass four is not performed until a method is called. Pass four deals with access restrictions and member accesses. So if a method is not called, it will never be checked.

Then the verifier checks to ensure that there is no violation of access restrictions, illegal type conversions, or uncontrolled memory access and usage. An important fact to note here is that because object linking is dynamic and the verifier confirms memory access, no forged pointers can be created, and hence securing the language is ensured. Once the bytecode has been verified, it is assured to all other parts on the Jre that the code is secure and usable. This also speeds up Interpretation or compilation as none of these checks need to takes place at these stages.

By default only files coming in from a network are verified, and all local files in the classpath are ignored. However one can force the runtime environment to check these files with the parameter *-Xverify:all* .

The Classloader

The classloader loads classes into their own namespaces. The reason for this is simple. Each class has a name or reference and it would not be desirable to have a class coming in off the Internet that replaces system classes on the client's machine. For that reason, the classloader separates classes from different network locations as well as the local classes. The classloader attempts to find the class being referenced in the local system, this is specified by the classpath attribute on the local system. If it does not exist on the local system, it will then attempt to get the class from some network source.

The classloader essentially enforces policies and rules on class usage, preventing any replacement of the system or Java API classes. The integrity of the system is maintained. The Classloader will not load any classes over a network in the *java.** package as this would comprise the security model by overriding the core Java API.

Runtime checking

As Java is a late binding language (It resolves class names and object at runtime), runtime checking is done to validate links these files. It also checks dynamic array bounds here, and if there are any problems at this level it will throw an error. This only occurs once the program is executing.

The Security Manager

The previous three features are the intrinsic security of the Java system. The developer has no control over it and need not worry about it. However the security manager is a resource level security mechanism. The developer must understand what and how it operates to be able to control the Java environment to its fullest extent.

Once the security manger obtains the bytecode it is still not guaranteed that the bytecode is safe for use. Thus far all that is ensured is that the bytecode conforms to all syntax and format regulations. The security manager prevents unauthorised stream manipulations. The most dangerous streams that may be manipulated are those of network and files. It determines the level of access that is allowed for the current runtime environment and enforces those policies on all commands issued. If for instance the application wanted to destroy the kernel of the operating system or any other protected file, the security manager would check the level of authorisation for the current object, based on this level the operation will succeed or fail.

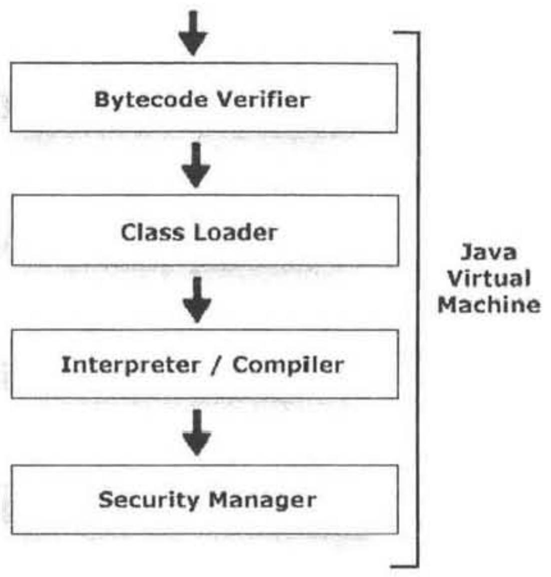


Fig 3.6.1 The Java Security Model

3.7) Applets and security restrictions

An applet is a Java program that is usually downloaded off a web site, and executed within a Java enabled web browser. The ability to download and execute on the fly is a fantastic feature. However the darker side of the Internet community would have abused this open invitation had it not been for the restrictive security mechanisms that are currently in place.

These restrictions are not placed on standalone Java applications, as one knowingly executes the program oneself, as opposed to a Java applet that could execute unbeknownst to the user when visiting a web page.

The browser that executes the applet in imposes most of these restrictions. Due to the fact there are a number of different web browsers available, the strictness of these restrictions varies as well. Essentially the applet is secured by the Security Manager, which is also known as the sandbox. The applet runs within the sandbox and has no access outside of it. The following important restrictions are placed on applets

Network restrictions

The restrictions are quite simple. All communication with the applet may only be to and from the originating web server. This is the server that the applet was originally downloaded from. No other network connection whatsoever may be created or used. This is because most client systems hide behind firewalls and it would be a security breach to send information out to anywhere in the world about the machines behind firewalls.

File restrictions

The restrictions on files are total. There is no reading or writing to any files whatsoever on the client's machine. This prevents sensitive information from being read and sent out into the Internet void. No files may be damaged or tampered with, thereby preventing virus's or malicious attacks on the clients system.

Miscellaneous restrictions

There are a few other restrictions that should be discussed. For instance, only a number of system properties are available (as read only) to an Applet, and no system properties may be modified. However it is interesting to note if one executes a local applet with "File:://applet.class" , instead of the web address it allows one to read and write the system properties. The manipulation of these system properties may potentially open up the system to intruders.

An Applet may not modify any threads not owned by itself, and in some web browsers, not even manipulate its own threads.

All frames that the applet opens up have a warning message at the bottom, so the user realises that it has been generated by the Applet, and is not another program or part of another application.

Applets may not execute local commands on the client's machine, as this would open a huge security hole in the system. The reason being that an undesirable could wreak havoc if file system control was made available.

Applets my not override predefined package names, such as the *Java.<libraryname>* or *Netscape.<libraryname>* packages, as these are the names of the local system packages. This prevents any sort of system breach.

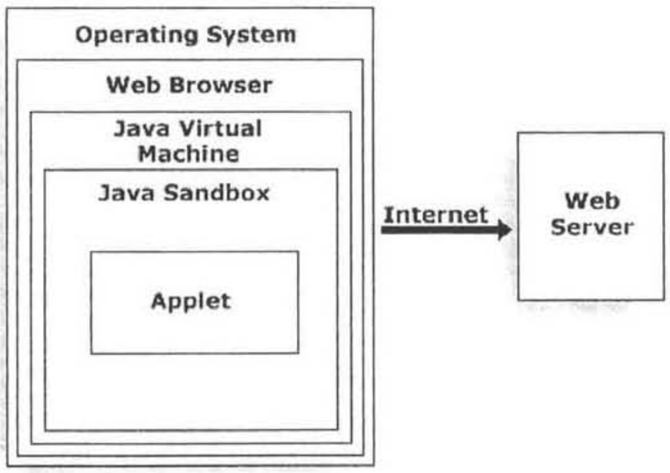


Fig 1.7.1 The Java Sandbox

3.8) Just in Time (JIT) Compiler & Java HotSpot ™

The JIT is a now a defunct feature in Java, and it is designed to increase the speed of applications. The main reason that people find Java slow is the fact that the bytecode is interpreted at run-time, and one is not executing standard native code. With the advent of JIT, Java bytecode is actually compiled into native machine language for that operating system.

The advent of Java HotSpot™ replaces the JIT in newer versions of Java. HotSpot uses adaptive optimisation and improved garbage collection. Adaptive optimisation finds the code that is the most frequently used code in the application. This is then compiled to native machine code and cached. This compiles to native machine code, the code that is used most frequently, instead of the entire application. Thereby speeding up code that is commonly executed. The Garbage collector has been improved and speeded up which assists in memory usage.

3.9) Java Vs C++

It is a well-known fact that Java's syntax is based on C++. However there are some critical factors that differentiate the two languages.

Pointers

The are no pointers whatsoever in the Java language. As previously stated, everything is treated as objects, and one uses references to those objects. This gives all the versatility of pointers without all the problems. Arrays, which are also objects, have automatic boundary limit checks, unlike C++. Java has built in support for objects such as strings and arrays.

Data types

Variables can only be one of two types in Java, an object or a primitive data type (int, float, etc). C++ has many types, from primitive data types all the way through to pointers. All Java primitives are fixed in length, and are based on Unicode. C++ has varying sizes of data types that are based on ASCII. In Java all variables are initialised on creation. C++ has no default initialization.

OOP

The Java language is intrinsically object orientated. C++ is a mix of OOP and classical procedural programming. Java cannot have more than one class being inherited by a new class. C++ can have a new class inheriting many classes. There is no overloading of Java operators whatsoever.

General

Java is developed to interact with networks and has a special affinity to the Internet. It is system independent and supports multithreading. As Java is compiled and interpreted it tends to be slower. C++ has no natural support for networks. It is system dependent and is not a multithreaded language. Due to the fact that it is compiled to native code it tends to be quicker than Java.

3.10) Java Programming Security Techniques

A language is only as secure as the programmer makes it. As has been pointed out [G. McGraw & E. Felten, 1999] there are a number of rules to ensure that Java code is a secure and safe as possible.

Don't depend on initialisation

This rule holds true for all objects and variables. To use an object, one usually has to initialise the object. However there are ways of avoiding this, which can lead to uncontrolled results and values in the application. Some suggestions are to firstly make all variables private, and use get/set methods to edit them. Secondly have a private variable to check that the initializer has been called. If it has not been called methods and properties should not respond to any requests.

Control access

Wherever possible have absolute control over the class, methods and variables. Make sure variables are not public, and hence publicly editable. Classes and methods should be private, unless there is a need for them to be public. Anything exposed is potentially a hole into the system.

Final everything

Using the *Final* constructor will prevent any further inheritance of the class, and therefore make it unavailable to prying eyes.

No inner classes

The general consensus is that an inner class can only be accessed by the class its enclosed by. This is incorrect, as once the classes are converted to bytecode, there is no concept of inner and outer classes. Secondly all variables that are private in the outer class are treated as public to the inner class. This is a potentially serious loophole, as by using inner class, one would have access to all the variables and methods of the outer class.

No signing

This is a bone of contention, as most people will not run serious applications unless they have been digitally signed to prove that it has not been tampered with. However a problem arises because signed Java applets are allowed special privileges. These privileges can then be used to manipulate files or networks.

No cloning

The cloning object allows one to make copies of class objects residing in memory. This means that someone can get an instance of your class without actually initialising the class. This is potentially dangerous. However the cloning feature can be disabled for classes, and this should be implemented.

No serialisation

Serialisation is the ability of the class's internal state to be copied. This is a useful feature, but it means that one can copy the current state of an object. Thus finding out all the information residing in private variables and methods.

Obfuscate

Once a Java application is compiled to bytecode, it is still possible to reverse engineer the bytecode with a Java disassembler. That means any hard coded values, such as cryptographic keys can be compromised. Rather generate values on the fly. Using long variable names can be easy for programming, but it makes the reverse engineered java code easier to read. After the final version is complete it is recommended that variables such as, New_game_starting_value, be replaced with zxxq or something similar.

Place useless code in the application, such as loops that do nothing or random variables. This will serve to confuse and make it more difficult to unravel the reverse engineered code.

Finally use professional obfuscating applications to make your code as difficult to read as possible. This will make the disassembled code almost impossible to fathom.

3.11) Conclusion

This chapter has highlighted a few of the reasons and strengths for the use of Java as the language of choice. It has been shown that Java meets the requirements of supporting network communications, specifically the Internet, as well as setting rigid security policies that will ensure the clients safety during network communication.

As the Internet is made up of many different platforms it would have been impossible to develop an Internet based application for all of them. Thus Java's unique capability of being system independent allows any Java application to be supported by Java enabled systems.

Added to all this, Java has proved itself as a robust, secure and simple language to program. Ensuring that the development of the application would be a far simpler process.

The next chapter will show how the Auctioning specification given in chapter two can be implemented within this Java framework.

4) Auction Implementation

4.1) Overview

The auction application is essentially the device leading to an understanding of Internet based security. Before one may understand the actual application it is wise to understand the scope of the problem posed.

The aim was to create an extendable, robust, real time, system independent auctioning application. The auction specifications and design have lead to the following solution; using a multithreaded networked based client / server application. The application is divided into two halves. One is the Server application, which controls the auction and all administrative features. The other is the client application, which is in the form of a Java applet. The auction server resides on a web server, from where the client applet is executed. The development of a web site, completes the application as a whole and make it easier for the client to participate in auctions.

4.2) Hardware requirements

As far as hardware requirement goes a Pentium 166 or greater should suffice for the client. The server will require a Pentium II or greater. Both should have 64Mb Ram or more. Both the client and server must be connected to the Internet and have valid IP addresses.

4.3) Software requirements

There are a number of requirements that have to be met before the client or server may be executed. Firstly the web site will have to be set-up in accordance with the structure of the servers directories. The reasons for this will be made clear later. Secondly it must be made clear from the start that the Java Development Kit 1.22, or what is now known as JDK 2 was used. This has a number of new features such as the swing graphical user interface classes. Therefore the Server hosting the web site and the server application must have the corresponding version (or greater) of the Jre libraries. The client applet has similar demands. The Jre Plugin 1.22 or greater will be required for the web browsers. The web browsers must also be Java enabled. This software may be downloaded from the Sun Microsystems site (Java.sun.com)

The server machine must be able to host a web site capable of using Active Server Pages (ASP) or a CGI scripting language. There are no operating system requirements.

4.4) The Server

The most obvious place to start is the heart of the entire application. The server, or more aptly named AucServer. The idea was for the server to execute and set-up all the auctioned items information. Then once it was prepared, to start the auction for a single element at a specified time. Once the current auction was complete the rest of the items would then proceed to be auctioned in an orderly fashion.

As there are over seven thousand lines of code it would be foolish to attempt to explain every method and variable usage. The easiest method for demonstrating the execution of the server is to take it task by task.

4.5) The Server Files

The server is made up of thirteen Java files. Each has its own unique responsibilities and duties. The aim from the outset was to have a multithreaded application that could be expanded if need be. A brief explanation is required for each file so that it will make the description easier to follow.

AuctionServer.Java

This file is the executable for the AucServer. This file starts up all the required services and maintains strict control on the following steps. Starting and Stopping of the server and auctions.

AucMemCheck.Java

This file starts a thread that monitors the amount of memory used by the application every two and a half seconds.

AuctionHandler.Java

This file controls all aspect of the auctions in progress. This includes everything from the set-up to the final outcome of the auction.

AuctionInterface.Java

This file controls the entire graphical user interface for the server application. Anything that is seen on the screen is controlled here.

AuctionNotify.Java

This file monitors the state of the auction every one second and notifies the objects concerned if the auction has been started, extended or stopped.

ServerListener.Java

This class listens for new clients attempting to join the server and starts all the necessary threads.

BidInfo.Java

This class is used for storing and retrieving the client's information for each bid in the current auction.

ClientHandler.Java

This class looks after each individual client's communication with the server and auction status.

ItemInfo.Java

This class stores and retrieves all the relevant information about the items being auctioned.

CryptLib.Java

This is a common library of methods used for the encryption and decryption processes.

MARSCrypt.Java

This class contains all the necessary algorithms and key schedules for the Mars encryption algorithm.

RC6Crypt.Java

This class contains all the necessary algorithms and key schedules for the RC6 encryption algorithm.

RSACrypt.Java

This class contains all the necessary algorithms and key schedules for the RSA encryption algorithm.

Starting the Application

The class file that starts the execution of the application is the AuctionServer.Java file. Its main procedure proceeds as follows.

```
// AuctionServer – Main Method

Initialize AuctionServer
Initialize AuctionInterface
Setup Frame
Initialize MemCheck
Start MemCheck
```

It creates an instance of itself, which is now, a thread and will not stop executing until instructed. An instance of the AuctionInterface is initialized, which controls the user interface. It then creates the application frame with all the required graphical elements. An instance of the AucMemCheck class is created and started. Now that the application is started it sits dormant until the user decides to fire up the actual server, and get the auctions started.

The Server State

The server has a few states of activity once the application has started. The server is in a stopped state by default. Once the server is running it may then start auctions. Once the auction is started it will eventually stop (either manually or automatically once the time runs out). From this state it may move back into the auction started state again, or the server may move to the server stopped state.

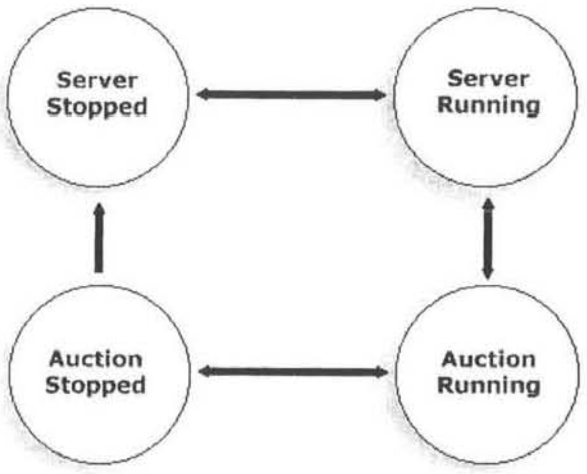


Fig 4.5.1 States of the Auction Server application.

The only state change that is automatically controlled by the server is that of the Auction Started to Auction Stopped state. The user manipulating the program can cause all state changes.

Server Running State

Hint: To start the server, Click on the Server | Start Server menu items.

To get into this state the user must start the server. The AuctionServer class will attempt to run the following code:

```
// AuctionServer – StartServerNow Method
Call StartServer method
StartListener


// AuctionServer – StartServer Method
Get Servers IP
Create Server Socket (Port number, Max Users, IP number)
Update User Interface
```

First the application obtains the local hosts IP number. Then it attempts to create a server socket on the current Port Number (3002 by default), with the current IP number and a maximum limit of Users (40 by default). If any error occurs at this stage the server will display an error message with the problem. Otherwise the server notifies the user that the server has been started. Once the server has been started it then starts the Serverlistener method which then waits for clients. When the user interface is updated, this includes all information displayed as well as the updating of menu items into the correct states.

It was originally decided to randomly generate a port number, but as this may lead to problems it was changed to an initial port number of 3002. There are about sixty five thousand port numbers available to servers over a TCP/IP network. Under a thousand are reserved but the number is steadily growing. For that explicit reason, if the server fails to start it will inform the AucServer administrator who can then manually change it to another port number before attempting to restart the server.

Server Stopped State

Hint: To stop the server, Click on the Server | Stop Server menu items.

This state may only be moved to if the server is in the running state (or if the auction is not running, but to get to that state the server must be in the running state). In attempting to move into the server stopped state a number of checks must be carried out first.

```
// AuctionServer – StopServerNow Method
Call StopServer method
```

```
// AuctionServer – StopServer Method
If (Auction Status = Auction Running) then Stop Auction
Stop Server
Update User Interface
```

The Auction can be in one of two states (Running or stopped). If the Auction is not running the Server may move to the stopped state. However the only way to get from an Auction Running state to the Server Stopped state is to first stop the auction (Auction Stopped State), and then the server may move to a Stopped state.

The Auction status is checked prior to the stopping of the server. If it is running the current auction is ended and the server is stopped. If any errors occur the user is notified. The user interface is updated accordingly.

Auction Started State

Hint: To start the auction, Click on the Auction | Start Auction menu items or click the black arrow on the menu bar.

Before one can start an auction it is wise to set-up all the preliminary information, such as the item details and the auction stop time. If item details are not entered the user will be prevented from starting an auction until there is an item to auction. Also if one does not set the time the auction will start and stop immediately. Once these details are completed the auction may then be started.

```
//AuctionServer StartAuctionNow method
Initialise the AuctionHandler class
Set AuctionHandler Auction Details
AuctionHandler.StartAuctionnow
Create AuctionNotify ( AuctionHandler, AuctionInterface, AuctionServer)
Start AuctionNotify
```

The following things occur to get the auction started. The auctionhandler class is (re) – initialised, all the details for the current auction are then set. Once this task has been completed, it notifies the Auctionhandler that the auction is starting, and starts the Auctionnotify class which checks every second on the current status of the auction. The Auctionnotify class accepts copies of the AuctionHandler , AuctionServer and AuctionInterface class. This is so it may communicate with these classes.

Auction Stopped State

Hint: To stop the auction, Click on the Auction | Stop Auction menu items or click the black square on the menu bar.

This state may only be changed to if the auction is running.

```
//AuctionServer – StopAuctionNow method
AuctionHandler.StopAuction
AuctionNotify.CheckTimer
AuctionNotify.Stop
Update UserInterface
```

The Auctionhandler is notified that the auction has ended. Then the Auctionnotify class is forced to check the current status of the auction (Stopped). It then notifies all required listeners. Once this is done the Auctionnotify class is stopped. Once this has been completed the User interface is updated accordingly.

Now that one has an understanding of what happens to the states and what classes are involved it would probably be easiest to show the flow of the auction application.

4.6) What Happens when the Server is started?

The server starts operating once it has obtained all the required elements of a server (TCP/IP and port numbers). It creates what is known as a server socket. A socket is a channel that allows communication over a network. A server socket waits for the initial connection and then determines whether or not to allow further communication. The server socket limits the number of clients that may join the server (default is forty clients).

However before allowing communication to proceed between the client and the server a number of services are required.

```
//AuctionServer,Java – StartListener method
RSA = New RSA object
Initialise new AuctionHandler (MAXBIDDERS, UI, PortNumber, RSA)
Initialise new ServerListener (AuctionServer, AuctionHandler, UI, RSA)
Start ServerListener
```

An RSA (Rivest, Shamir & Adelman encryption algorithm) object is created, this controls public key secure communication and the same public key is used for all clients. (Once keys have been shared a different key is used for communication between the server and each client.) The Auctionhandler object is started and passed all the required objects it will need to have access to. MAXBIDDERS is a variable that specifies the number of top bids to retain. The default is set to five. The Serverlistener takes similar objects, which it uses to pass to the client's threads once they are created. This allows communication between the client threads and the auctionhandler. The serverlistener is then started.

The serverlistener is a separate thread that listens for client communication. If the maximum limit has not been exceeded the serverlistener thread performs the following operation.

```
//ServerListener Class - StartListener Method
New Socket = Accept Client Communcition
Create New ClientHandler (Socket ,AuctionHandler,RSA)
Start ClientHandler
Update User Interface
```

First it accepts the connection of the client and creates a local socket of communication with the server to the client. For each port there are sixty five thousand sockets. As there are approximately sixty five thousand ports available, it adds up to an enormous amount of client connections on one machine.

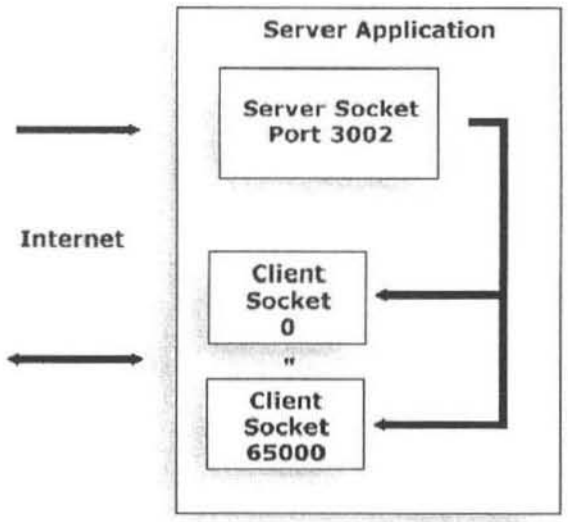


Fig 4.6.1 Server Socket accepting connections.

Once a new socket is created, it is used for exclusive communication to the client concerned. Once the communication channel is open and there are no problems the serverlistener starts a new ClientHandler thread for the current client. The local socket is passed to the clienthandler, which then deals with all communication with the client. What this means is that each client that joins will have its own thread. The reason for this is that the server will be able to take advantage of a multiprocessor environment, thereby improving performance and scalability of the application.

The Serverlistener class sits in a non-blocking loop waiting for clients. The reason for making it a thread is quite simple. The anticipated volume of clients joining and leaving makes it a busy process, which would have made it quite difficult for the auctionhandler class to deal with this facility as well. To facilitate the modular design the server listener was written as a thread

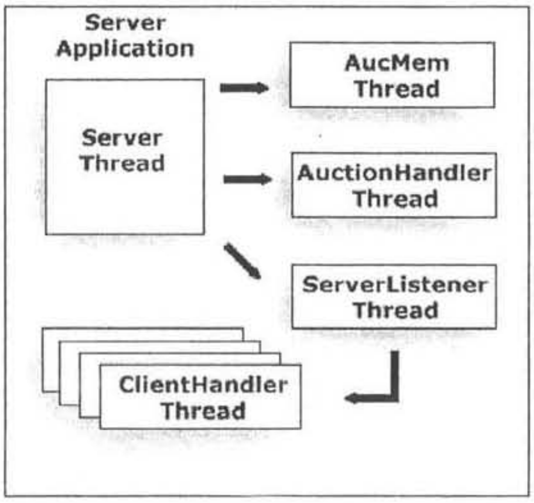


Fig 4.6.2 Initial threads created by the Server.

The server application starts the AucMem,AuctionHandler and ServerListener threads. Thereafter the ServerListener thread starts the ClientHandler threads. As will be shown later at any one stage there is a minimum of four threads executing.

4.7) What happens when the Auction is started?

The auctionhandler object is a reactive object. It responds to requests and calls from other objects. These other objects are the AuctionNotify object, the Server object and all the ClientHandler objects.

However it is important to understand the make up of the AuctionHandler class, as it is crucial to the entire program. When the AuctionHandler class is initialised, it initialises the variables used to control the flow of the application.

```
//AuctionHandler - AuctionHandler method (Constructor)
Set local References to AuctionInterface & RSA
StopTime = CurrentTime
MaxItems = 10;
Client List = New Vector List
Set Root Directory string = current directory.
Open Log Files
Initialise lists
```

This method is called when the Auctionhandler class is initiated for the first time. It sets all required references and variables. It's interesting to note that the Client List is a Vector List. This list keeps a list of all the clients current joined to the server. The power of Vector is in the fact that they are a dynamic array of objects. Vectors are synchronised objects, which means there are none of the problems associated with a multithreaded environment. (These are usually locking and sharing issues). In the scope of the application, the arrays of objects in the Vectors are references to the ClientHandler objects for each client

.

The Initialisation of lists incorporates the resetting of the ItemInfo lists and the BidInfo lists, which are vectors used for storing information about each item in an auction.

The Auctionhandler object has a Boolean variable called auction_status. This variable is used to indicate the state of the current Auction. As the auction is started it is set to True. It has another Boolean variable called *values_changed*. This variable is set to true if the top bid changes in the auction application .The Auctionhandler class has a StopTime variable of type Calendar. Calendar is a high precision date and time data type. It is set to the stop time for the current auction. The AuctionHandler class has all the methods to set and retrieve the variables discussed. The pro-active object that updates the state of the auction is the AuctionNotify class. Once started it sleeps for one second and then checks to see if the state of the Auction or its properties has changed.

One of three possible scenarios could have occurred, 1 (No change), 2 (Top bid change) and 3 (Auction Over). To determine this, the AuctionHandler object is queried.

One
If no change has occurred the Auctionnotify class will do nothing and proceed to sleep for another second.

Two
To find out if the bids have changed the value of *values_changed* is returned from a method [1]. If it is true then a new bid has been placed within the last second. The Auctionnotify class then notifies the Auctionhandler who then updates all parties concerned.

Three
To find out if the auction is over a method called *auctionStatus()* in the AuctionHandler class is executed. The method determines whether or not the current time is greater than the finish time of the current auction. If the auction is over, true is returned. The AuctionNotify class then updates the user interface, writes information to the log files and notifies the AuctionHandler that the auction is over.

As the AuctionNotify class works once every second it was decided to utilise this class for more than one purpose. That is to update the user interface with the changing time. It would have been pointless and rather a waste of resources to create a new thread for this purpose. The flow of the program really boils down to interaction between the AuctionNotify class and the AuctionHandler class. One could see the Auctionhandler class as the central repository for the entire, pertinent and essential auction control variables. However the AuctionHandler will not do anything until instructed to do so. The reason for keeping everything within one class is that it is easier to control the access to these variables. Hence it is more secure. The AuctionNotify class has no knowledge of these variables, and has to request them from the Auctionhandler class.

[1] All variables are set or retrieved using a Get / Set method. There is no direct access to the variables in classes.

4.8) How does the user interface work?

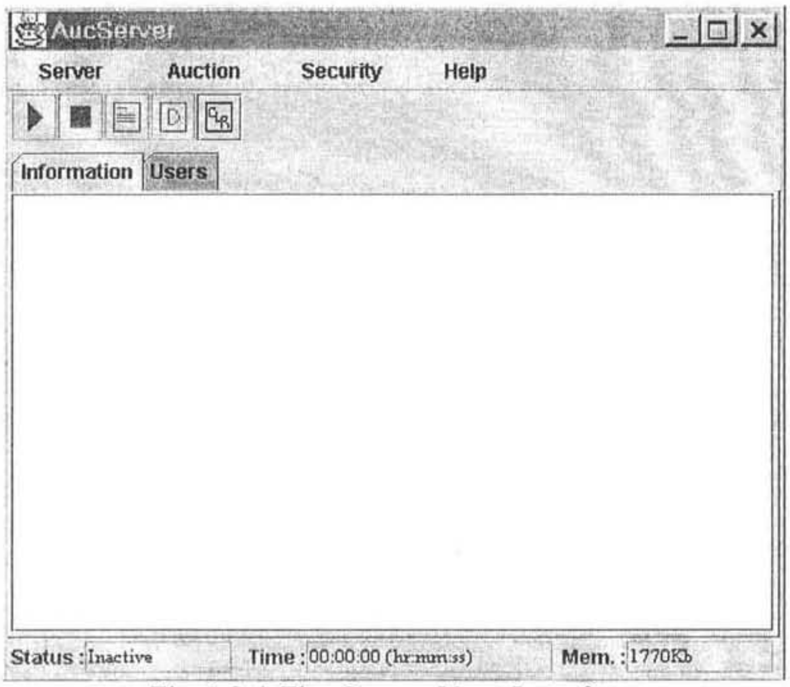


Fig 4.8.1 The Server User Interface

The user interface was developed using the Java Foundation Class (JFC), Swing class. Sun Microsystems has redeveloped the original Abstract Windows Toolkit (AWT), with help from Netscape, IBM and a few other companies. The result was the JFC. This is made up of AWT, Java 2D,Accessibility, Drag and Drop and Swing. Swing determines how the GUI is presented, independent of platform. Swing essentially extends the current AWT with a more advanced lightweight graphical library. One of the beauties of Swing is the "Look & Feel" concept. "Look & Feel" means the application will look like a native application to the current system. That is the application can either have the look and feel of the current platform, or it can be fixed to a specific platform.

The user interface for the server application is controlled entirely by the AuctionInterface application. On initialisation of this class the frame and everything within it is set-up and displayed. The entire visual interface is generated and then placed on the frame. When everything has been prepared it is then displayed. This technique is used often to imitate faster program execution.

The application frame is broken up into four sections. The menu, menu bar, tabbed application pane and the status bar.

The Menu

The menu has four menu headings. Those are Server, Auction, Security and Help. The Server menu deals with the starting and stopping of the server. The setting of the port number and exiting of the application.

The Auction menu deals with the starting and stopping of an auction. The demo menu item starts the Demo Auction. The Item details menu items deal with the populating of auction item details. The Stop Time menu item sets the finish time for the current auction.

The Security menu is divided into two halves. The first half allows one to specify the key length used for encryption, and the regeneration of keys. The second half of the menu is dynamic. It lists the available encryption algorithms. For the sake of extensibility it has been designed to allow new algorithms to be incorporated. Currently the two algorithms dealt with are RC6 (default) and MARS.

The Help menu has one item, which is the Help element. This opens a new window that is linked to the default Help HTML page. The window is essentially a mini web browser and all help pages available are in a specified directory in HTML format.

The Menu Bar

The menu bar is made up of five buttons. Starting from left to right: The Black arrow starts an auction. The black square stops an auction. The page icon displays the item details for the current auction. The "D" button starts the demo auction. The "CLR" button clears the Information page of all text.

The menu bar uses a useful utility called tool tips. These allow a short description to pop up, when the mouse is passed over the button in question. This aids in the overall user friendliness of the application.

The Tabbed Pane

The Tabbed pane is divided into two panels: The Information tab and the User tab. The Information tab deals with information pertaining to the server. Whenever a new action takes place it will be displayed in the information tab. The User tab displays all the users that are current joined to the server, their details and the top bid. The User tab is dynamic and will change according to amount of users and bid values. Both tab panes are scrolled enabled. This means if the viewing area is not sufficient to display all of the information it will automatically add a scroll facility to do so.

The Status Bar

The status bar is made up of three items. Those are, Auction Status, Auction Time and Memory Used. The status of the auction is updated to "running" when an auction is in progress, otherwise it displays "inactive". The Auction Time displays the time remaining till the end of the auction. This is updated every second by the AuctionNotify class. The final element on the status bar is the Memory used feature. This is updated every two and a half seconds by the AucMem class.

User Interaction

One will realise that all the buttons and menu items are not always available (greyed out). This is due to the state of the current application. Elements become unavailable or available due to the current state of the application. Refer back to *4.5 The Server* to understand all the states available.

The AuctionInterface class also listens and reacts to all the events that occur with regards to the user interface. All events are sent to a class called the AuctionHandler class. This class then deals with all user interactions and selections. This class will then call the required methods for the desired selection. For this reason the Auctioninterface class requires a reference to the AuctionHandler and Server classes; as it allows this conveyance of state change and the execution of the methods in the aforementioned classes.

The AuctionInterface class deals with all other graphical aspects, such as error message dialogue boxes, selection menu boxes and the HTML Help frame. However it must be re-iterated that this class does not contain any information regarding the actual auction state. It is the vessel by which the user may move from the one state to the next.

User Information

The tabbed panes show the user descriptive information with regards to the current state of the application. As the AuctionInterface class has no access to this information it would be impossible for the class to achieve this task by itself. Therefore the AuctionInterface class has made available methods for the classes that control the application to display information.

To update the Information tab one executes the method *displayResult(Message)*. The message will then be displayed on the information tab. To clear the Information screen one simply calls the method *clearScreen()*.

To update the User tab one simply has to execute the method *updateUserScreen(ClearScreen , Message)*. If the Clearscreen = one, it will clear the User screen and start afresh, otherwise it will simply add the message to the screen.

Application Termination

Hint: To terminate the application, Click on the Server | Exit menu items.
Or
Click on the close button in the top right hand corner of the application window.

When the application is terminated it follows a similar routine to that of stopping the server. It stops any auction running, then stops the server and finally it terminates execution. When the program terminates all references to the objects of the application are lost and the Garbage collector cleans up the memory.

At no stage should the application terminate execution with an error. If this does occur it is beyond the control of the application.

Known Problems

There are a few problems with the user interface. However these are due to the actual language. It is a well-documented problem that there may be refresh problems for the JFC/Swing classes. Another problem occurs when recreating keys. If the key is larger than 128 bits, the CPU workload becomes intensive. This prevents the screen from being refreshed immediately.

However these are minor issues and will not impact on the overall performance of the application. The benefits of using the Swing classes outweighs the few problems that do occur.

4.9) The Client Applet

The client's applet was designed to interact with the servers application and be as independent as possible. The aim was to make it as simple as possible for the client to participate in the auction process.

Applets are Java applications embedded in web pages. They are downloaded once one visits the web page they reside within. The benefit of this is that the client does not need to do any special downloading and installation. The client simply opens the web browser and proceeds to the correct web page. The applet is downloaded and then executes on the clients machines, assuming the client has a Java enabled web browser and the correct Java version.

There was some design criteria that had to be met. Specifically the size issue. Not all people have ISDN and satellite connections to the Internet. Slow modems and service providers are still prevalent around the world (and particularly South Africa). Therefore the applet had to be as small as possible to aid in the speed of the download to the client's machine. The applet weighs in at thirty-five kilobytes which is less than the average size of a web page. To streamline matters further the applet has been placed in a Single JAR (Java Archive) file. This JAR file compresses all the classes within it using the Zip compression algorithm. This reduces the size of the applet even more. In fact there is about a forty-five percent compression rate for the client applet. When a new class file is requested Java first search the local machine, JAR's included, before requesting from the originating server. This tends to speed up the performance of the applet, as no extra network requests will be needed to obtain all the required classes.

4.10) The Applet Files

All the files required for the client application are place in the JAR file called "aucjar.jar". The files required by the applet are as follows:

AucClient.Java

This is the main class for the Client applet. It starts all the required objects and threads.

AucUI.Java

This class controls the user interface for the client applet.

ClientEngine.Java

This class controls the communication and general program management of the applet.

AucImage.Java

This class controls the process of downloading the image files for the current auction elements.

AucTimer.Java

This class updates the time remaining field for the client.

CryptLib.Java

This is a common library of methods used for encryption and decryption processes.

MARSCrypt.Java

This class contains all the necessary algorithms and key schedules for the Mars encryption algorithm.

RC6Crypt.Java

This class contains all the necessary algorithms and key schedules for the RC6 encryption algorithm.

RSACrypt.Java

This class contains all the necessary algorithms and key schedules for the RSA encryption algorithm.

HTML File

The HTML file in which the client applet resides, contains embeded code to notify the web browser what to do with the JAR file.

The Client State

The AucClient.Java class is the file that runs initially. It executes the *start* method, as this is what executes in an applet. (As opposed to the *main* method in an Application).

```
//AucClient.Java – Start method
Get parameters (itemid, port number)
If (Auction Running) & Port number = ok)
        Initialise AucImage
        Set Initial Image = item
        SetupConnection
        Initialise AucUI (applet,AucImage)
        Initialise ClientEngine(AuctionUI,codebase(),AucImage,Itemid, port number)
        ClientEngine.start
End if
```

The applet first attempts to get the port number and itemid from the HTML page, which are passed in as parameters. The port number tells the client what port to connect to the server, as the port number is dynamic. The itemid value informs the client applet what item the client started bidding for.

If there were no problems, (such as corrupt parameters) the itemid exists and the port number is not invalid, then the auction continues. It initialises the AucImage class and sets the first image equal to the name of the itemid. The format specifies that item images be named the ID of the item. (I.e. Item ID = ITEMAA1, then Item Image = ITEMAA1.GIF) The AucUI and AucEngine classes are initialised and passed the required references of objects. The clientengine class is then started, which takes over control of the running of the client applet. The codebase() method returns the URL (Uniform Resource Locator) of the applet code on the web site.

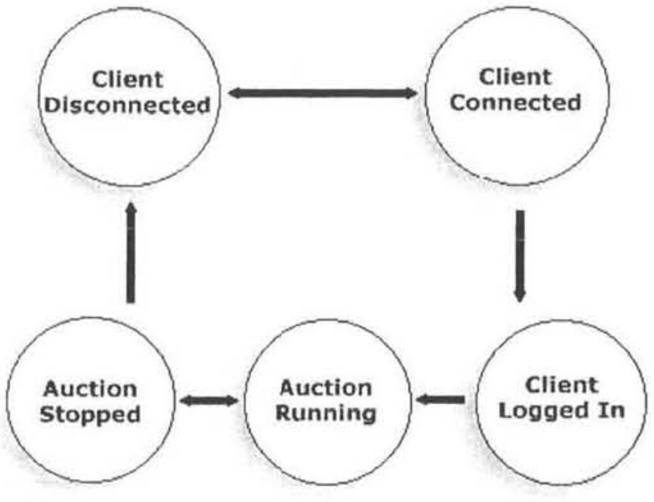


Fig 4.10.1 The Clients States

Client disconnected state

This is the state that the client starts in. However on the loading of the applet, the client attempts to connect to the server immediately. If the connection is lost at any stage to the server the client moves back to the disconnected state.

Client connected state

The ClientEngine class sets up the connection to the host as follows:

```
//ClientEngine.Java – SetupConnection method
Get Hosts address
New Socket = Socket (Host Address, Port number)
Set Buffered input/output streams
```

Initially the client obtains the host address of the applet. It then uses this address to open a communications socket to the host (Server application), using the port number that is passed in as a parameter from the HTML document. It may only open a network connection to the originating host that the applet came from. This security feature prevents unauthorised communication to other machines. Once the client is connected the application may proceed to the next state.

Client logged in State

Hint: To log into the server. Fill in all the required details in the relevant text fields and then press the "Connect" button at the bottom of the screen.

This state is achieved once the client has logged into the server with all the required personal details. This requires the client to type these details into the relevant fields on the applet screen. These include name, email etc. The client logged in state cannot be achieved unless the client has been connected to the server first. The details are simply forwarded to the server. Only once this state has been achieved may the client participate in any running auctions.

Auction running state

Hint: To participate in the auction, enter a bid in the text box. To submit a bid, click on the "Submit Bid" button.

This state is controlled entirely by the Server application. It notifies the clients about the status of the current auction and updates frequently. In this state the client may bid for the currently auctioned item.

Auction stopped state

Similarly for this state, the Server is in control. It will end the clients Auction when the Server changes its own state. The Clientengine class updates these state changes.

4.11) What happens when the Applet is started?

The Clientengine object is a reactive object. It responds to requests and calls from the server and other objects.

It is important to understand the make up of the ClientEngine class, as it is crucial to the applet control. When the ClientEngine class is initialised, it initialises the variables used to control the applet.

```
//ClientEngine.Java – ClientEngine method
Set local References to itemid ,port number, AucImage & AucUI
Initialise RSA & CryptLib
RSA = New RSA object
Set Key = 0
```

This method is called when the ClientEngine class is initiated for the first time. It sets all required references and variables. The ClientEngine also keeps a copy of all the clients' details (name, email etc).

Login page

The Login page is the first page loaded and displayed by the applet. This requires the users details. Without completing this section first the client may not proceed to the Bidding Page.

Bidding page

The bidding page is displayed once the client has logged in. It displays all the details regarding the current auction. This includes the clients Auction ID, the Item ID, the top bid, minimum bid and the time remaining. The client receives an auction ID number that is unique. This is used to differentiate between different clients and also maintains client anonymity. The Server generates and distributes the auction ID. This ID number is retained while the client stays logged in. If the client leaves the auction and rejoins they will be issued with a new auction ID.

When the client joins initially, and every auction thereafter the details are sent from the server to update the client with all the required details. The client then bids for the item and is notified when the auction is over and who won it. When the following auction starts the client is notified that a new auction has begun. To aid in the user friendliness of the application the client also receives an image (if available) of the item currently being auctioned.

If no auctions are running the applet will display no values and the client will not be able to bid. This auction running and stopped states are displayed, as long as the client is connected.

4.12) How does the user interface work?

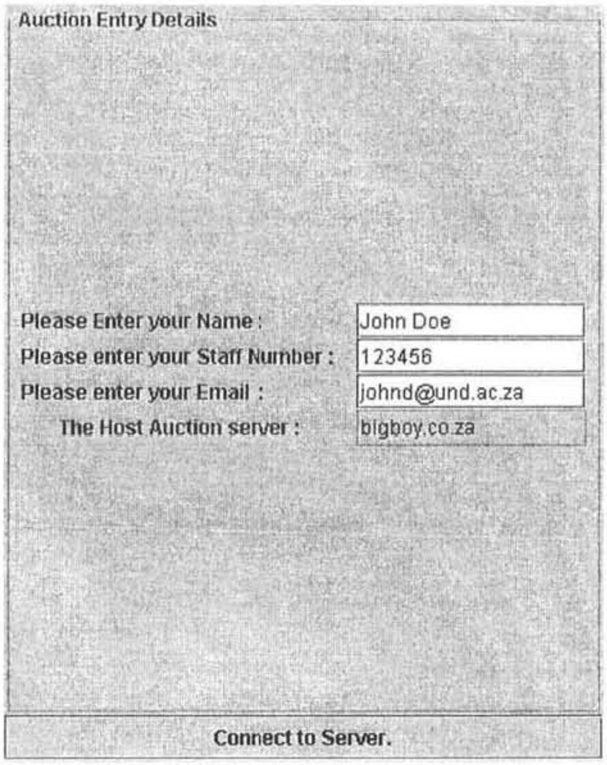
The AucUI class controls the user interface. As with the server application this class has no control of the actual information it displays. However it will display the information received from the server.

The applet is made up of two distinct screen shots (see below), the login page and the bidding page. The applet uses layered panes. A layered pane is a window with more than one pane within. Each pane or element has a z-value. This z-value determines what pane is displayed. The pane with the highest positive z-value will be displayed over all the other panes.

When the application begins the login pane has the highest z-value. Once the client has logged in the AucUI class changes the z-value of the bidding page so that it is higher than the login page. This causes the bidding page to be displayed. The reason that this is so useful is that both pages are prepared as the applet starts up but remain invisible. The applet takes a little longer to execute initially, but once its running things are quicker. When the applet is ready it displays the current pane and makes the perceived speed of the application seem greater.

The Login page

The login page is made up of one panel and a button. On this are four text boxes. They are in order, the Name, credit card (or staff number in the demo version), email and Hostname fields. The first three fields require the user to enter data. Any fields left blank will prevent the user from continuing. The fourth field displays the host name of the applet. If there is no connection to the server the applet will display an error here. Once the fields are all updated the user can continue by clicking on the "Connect to Server" button. This sends the data to the server. This page is hidden and the bidding page is then displayed.



Auction Entry Details

Please Enter your Name :	John Doe
Please enter your Staff Number :	123456
Please enter your Email :	johnd@und.ac.za
The Host Auction server :	bigboy.co.za

Fig 4.12.1The applet login page.

The Bidding Page

The bidding page is made up of four sections. The Auction bids, auction parameters, auction information and auction image. The auction bids section consists of a text field for placing a bid and a button to submit the bid. If the text value consists of non-numeric characters an error message will be conveyed to the client. To send the bid the client must click the "Submit Bid" button.

The auction parameter section displays the details of the auction. It consists of four text boxes. The Auction ID textbox displays the client's own Auction ID value. The auction item ID value displays the ID value used for the current item auctioned. The Top bid field displays the current top bid and the auction Id of the bidder. The time remaining text box displays the time left for the current auction. The AucTimer class updates this textbox every second. Due to the fact that the server sends the time remaining initially and if there are any changes, there may be a slight discrepancy between the clients time and the Servers time. However the server has the last word, and it indicates when the auction is over. The time remaining clock is meant to be a guideline for the client. To prevent unfair last second bids the auction will add a minute to the time remaining if anyone bids within the last sixty seconds. All of these text boxes are uneditable and are updated by information received from the server.

The auction information page informs the user on minimum bids, bids submitted, time extensions and other auction proceedings. The AucUI class has no control over the auction image section, instead the AucImage class updates the image section. This takes care of obtaining and displaying the correct item image. It downloads the item image off the server and promptly displays it in the image section.

In all instances the ClientEngine class instructs the AucUI and AucImage classes to update the information.

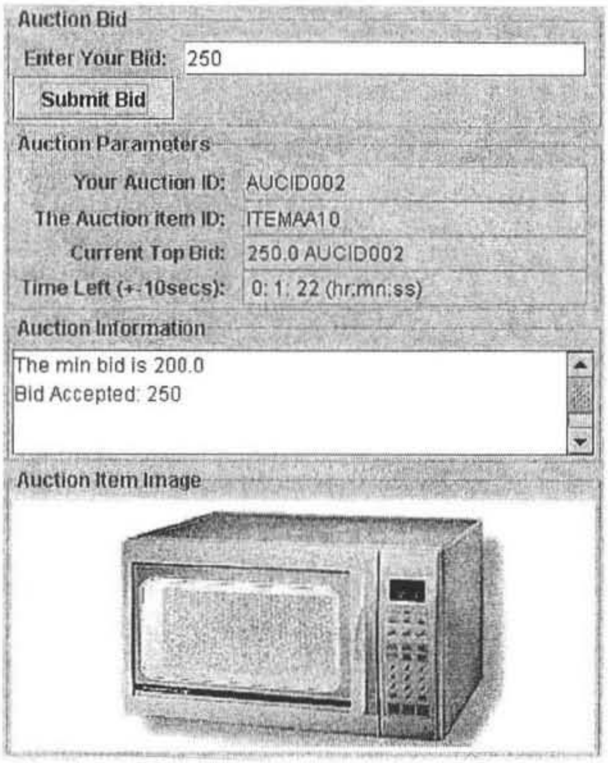

Auction Bid
Enter Your Bid: 250
Submit Bid
Auction Parameters
Your Auction ID: AUCID002
The Auction Item ID: ITEMAA10
Current Top Bid: 250.0 AUCID002
Time Left (+-10secs): 0: 1: 22 (hr:mn:ss)
Auction Information
The min bid is 200.0
Bid Accepted: 250
Auction Item Image

Fig 4.12.2 The applet bidding page.

User Interaction

User interaction only takes place on two occasions. When the client attempts to login into the server, and when the client attempts to place a bid. In both occasions the class that controls the interaction is the ControlAction class. When it receives this action it makes sure there are no abnormalities with the information being entered by the client before presenting it to the ClientEngine class to send to the server.

Applet Termination

The applet will only be terminated when the client moves to another web page. As long as the client remains on the original web page the applet will continuing running. Re-visiting the web page will cause the application to restart completely.

Known Problems

There are a few problems with the user interface. However they are well documented in the Java literature. The main problem that occurs is the repainting or refreshing of the client applet. On some occasions the applet forgets to repaint when it has lost and regained focus.

4.13) How are messages sent between parties?

One must first understand that the application uses the TCP/IP (Transmission Control Protocol / Internet Protocol) protocol. The reason for this is that TCP/IP has a number of useful features. It guarantees delivery in the correct order of all information sent or received. This means that all data sent would be received in the correct order, so no manual time stamping is required and information sent will not be corrupt.

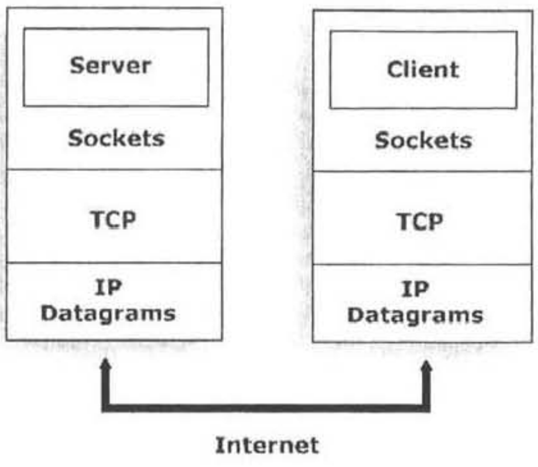


Fig 4.13.1 TCP/IP communication

IP is a non-guaranteed and connectionless layer that sends datagrams. TCP then controls these packets of information, re-sending when necessary, and placing in the correct order. The stack layer shows how the applications open virtual streams of communication via TCP, which send the packets using the IP protocol from the server to the client and back again. TCP/IP is far more complex than shown and a discussion is not needed in the context of this chapter [Hughes, Hughes, Hoffner & Winslow, 1998, p10].

Java uses streams to communicate, be it to files or networks. Both the client and server applications open input and output datastreams. A standard stream can only send and receive bytes. Data input and output streams allow a higher level of communication, such as string and data types. The problem though with these streams is that they are not buffered. As one places information in the stream it is automatically sent. This leads to two problems. One is that it takes a lot of resources to do this continuous stream handling, which slows down the operation of the application. Secondly it does not use the TCP/IP packets to its fullest, and one will be wasting bandwidth, as many packets containing few bytes will be sent. To combat this problem the streams are buffered. This allows for the messages to be completely created and then placed in the stream, when the buffer is flushed of its contents, thereby sending information less often in fuller TCP/IP packets [Hughes, Hughes, Hoffner & Winslow, 1998].

The usage of these wrappers gives the application more flexibility and allows easier communication. The way in which the streams are constructed is displayed in the following diagram. The original input and output streams are linked to the socket that the application has opened to the other parties involved. Both the server and client applications use the same send method.

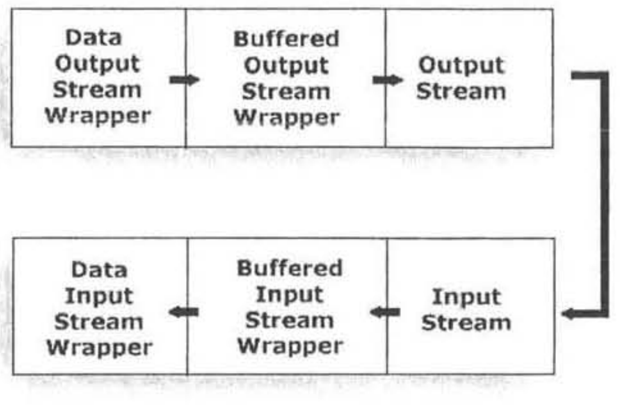

Data Output Stream Wrapper
Buffered Output Stream Wrapper
Output Stream
Data Input Stream Wrapper
Buffered Input Stream Wrapper
Input Stream

Fig 4.13.2 Stream wrappers

```
//SendMsg (String Message) method
If (encryption_set) then Message = Encrypted(message)
Outputstream.WriteUTF (Message)
Outputstream.flush
```

First the method checks for encryption. If this is in place the message is encrypted. The message is then placed in the output stream (which is buffered). The message is sent once the call to flush the output stream has been executed. If there are no stream problems the message will be sent. The message is sent in a UTF-8 format. This writes each element of the String message in one, two or three byte formats. Common ASCII text will use one byte and the more uncommon ASCII text will use three bytes. This decreases the amount of information sent, thereby saving bandwidth in an efficient manner. The length of the String is also encoded so the stream on the opposite end will know how long the message is and can separate messages. This format is the common format for communication between Java applications or programs.

The format of the messages sent between the client and server are quite simple and relatively small. The message is divided into two sections. The message descriptor and message information. The message descriptor is a space-delimited set of numbers. Each sequence of numbers leads to a different command. The message information contains all the information required for the current action.

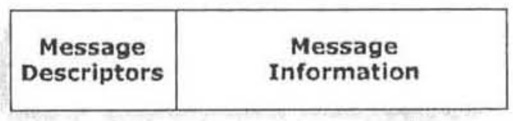


Fig 4.13.3 The Message format

The Server's messages

As has been indicated by the message format, there are conventions for sending commands. The following few paragraphs will explain what messages the server sends to the client and what results from them. The server sends five kinds of messages. The table below lists the format of the messages sent.

Message Number	Message Identifier	Message format
Security		
0		<Algorithm type> <Public RSA Key> <Public N value>
Registration		
1		<Client Auction ID>
Auction Details		
2	1	1 <Item ID> <Min Bid> <hrs> <mns> <scs>
2	2	2 <Top Bid> <Top Bidder ID>
Bid Response		
3	1	1 <Bid Value>
		(Error messages below)
3	2 1	*Bid must be Higher*
3	2 2	*Auction Not running*
3	2 3	*Bid lower than Minimum*
3	2 4	*Bid lower than Current Top bid*
3	2 5	*Encryption Error*
Extend Time		
4		<hrs> <mns> <scs>
Average Time		
9		

Fig 4.13.4 Message formats from server to client

Note: <Value> are values sent to the client.

The Security message sends, to the client, the type of algorithm that is being used for symmetric encryption, as well as the public key pair for RSA encryption. The registration message sends the client and Auction ID on confirmation of all their details. The auction details messages sends one of two messages. The first is the details of the current item auctioned. This is sent by default the instant a client logs into the system, or a new auction begins. The second is the current top bid and bidder for the auction. This is sent every time the top bid changes. The Bid response message replies to the clients bid. The response is either an acceptance of the bid or a rejection with an error message. The extended time message updates the client with the new finish time of the current auction. The average time message is very similar to what is known as an Echo request. On receiving this message the client responds immediately with the same message. This allows the server to measure the average time packets take to and from the client. All messages are space delimited.

The servers messages are sent by calling the method *Notify_Client(Message No, Message)* with the appropriate information.

The Client's messages

These are the message formats sent to the server from the client's machine.

Message Number	Message format
Security	
0	$<K_1>..<K_n>$ //RSA Encrypted Symmetric Key
Login	
1	<Name> <Number> <Email>
Bid	
3	<Bid value>
Average Time	
9	

Fig 4.13.5 Message formats from client to server

The client has a limited number of messages. The first one sends the client's key to the server. It has been encrypted using the Servers public key. The login message sends the clients details to the server. The bid message sends the clients current bid for the auction to the server. The average time procedure responds with the same message received.

The client calls the method *notify_Server(message number, message)*. This method then executes the correct procedures to send the message to the server.

Receiving messages

The client and the server threads sit in a non-blocking loop waiting for information to arrive in the input stream. As soon as it arrives both applications call a method called *readMsg()*.

```
//readMsg() method
String = (String) InputStream.readUTF()
return String
```

This method reads the message from the input stream using the UTF format. This UTF format is then cast (changed) to a string and returned to the caller. A method called the *MessageProcedure(Message Received)* takes this received message and deals with it accordingly.

```
//MessageProcedure (Message Received) method
New StringTokenizer(Message Received," ")
Message number = (integer) NextToken.StringTokenizer
Select (Message number)
        Case 1: Call Method(Rest of Message Received)
        " "
        Case n:
End Select
```

The method accepts the string received. It then creates a String tokenizer. The string tokenizer breaks the string up into tokens or different elements. The delimiter used to indicate tokens is a space. For instance "3 250" would have two tokens. The first is the message number. This determines what to do with the rest of the information sent. The message number retrieved is cast to an integer and used in a select statement. This select statement then calls the methods associated with the current command. It passes the remaining string to the method being called. Continuing the example of the Bid "3 350" to the Server, would mean that case 3 is chosen and the *BiddingProcedure(String)* would be called. The value "350" would be passed to the *BiddingProcedure* method.

When the message is received, at every stage the message is examined for malformity. This prevents any unwanted or incorrect messages affecting the application. If any poor messages are received they are simply ignored, and the application makes a note of the occurrence. The message, time and whom it is from is all recorded in the log file for examination at a later stage.

4.14) What happens when a bid is placed?

The client sends the bid in a string format. This is placed in the network stream, which is then flushed. The Clienthandler object for that current client receives the message. This message goes to the *messageprocedure* method, which determines the message type and calls the correct appropriate method (in this case it's the *BiddingProcedure* method). The rest of the string (The actual bid) is passed into the appropriate method where it is dealt with.

```
//ClientHandler.Java - BiddingProcedure(String Bid)
Bid value = (float) Bid
Result = SubmitBid(ClientID,Name,Email,Number,Bid value)
Switch (Result)
        Case 1: Notify_Client(Success)
        Case 2: .. Case 5: Notify_Client(Error)
End Switch
```

The string value is converted to a floating number. The bid is then submitted, along with the client's details. The reason for the details being passed along is twofold. Firstly it helps differentiate one bid from the next. Secondly no other copies of the clients details exist outside of their Clienthandler object. When the client leaves this object is destroyed, and so is the client's information. It was imperative to keep this information, so the clients bid and details is stored by the Auctionhandler. When the auction ends the AuctionHandler will know the details of the client who won without having to keep the all the Clienthandler threads alive. This small overhead allows for an even greater saving of resources later in the life span of the application.

```
//AuctionHandler.Java - SubmitBid(ClientID,Name,Email,Number,Bid value)
If (auction_status = stopped) return "Auction Not Running."
        Else
If (MinBid(Current Item) > Bid value) return "Bid lower than min bid."
        Else
If (TopBid(Current Item) > Bid value) return "Bid lower than top bid."
        Else
```

```
If (CheckBiddersExistence(BidList) = true) &(BidValue> LastBid ) then
                        BidList[BidderPosition].addBid(Client Details)
                        SortBids()
                        ValuesChangedNotify()
                        Return "Success"
        Else
                        BidList[LastPosition].addBid(Client Details)
                        SortBids()
                        ValuesChangedNotify()
                        Return "Success"
End if
If (timeLeft() < 1) then extendAuction
```

The procedure first checks to see if there are any problems with the bids. It makes sure the current bid is greater than the minimum, top. It also checks to see that the auction is running before hand. Once all these checks have been confirmed one of two cases can occur. The client has bid before or it is the client's first bid. If the client had bid before and the new bid is higher than the previous bid, the new bid is entered. It overwrites the older bid in the Bidlist and the bids are then sorted to get them in order.

If the client has not bid before, it is added to the bid list. The list is then sorted and the Auctionhandler is notified there has been a change in the bid list. On all occasions the procedure returns the result of the update.

Finally this method checks to see if the time remaining in the auction is less than one minute. If it is, a bid has been placed in the last minute of the auction, which is by all means unfair for all the other contestants. This results in the auction being extended by another minute. This extension time is defaulted to one minute.

Bid Storage

The bids are stored in the BidInfo class. The Auctionhandler class has a dynamic array of theses BidInfo objects. The default number of elements in this Bid list is five. However it can be extended. The reason for making it a dynamic array was the fact that not all auctions are based on the top bid (i.e. Dutch Auctions). The result is that the top n (n=5) bidders bids are kept. For each variable there is a get method to retrieve the information.

The BidInfo class stores the following details for each of the top bids.

Bid Variables	**Example**
BidderID	AUCID001
BidderName	Joe Soap
BidderEmail	SoapJ@email.com
BidderNumber	12345
BidderValue	350
BidTimeStamp	2000-01-13,12:37.38 am

When the client calls the setBidInfo(ID,Name,Email,Number,Value) method, these values are set to local copies of the object. However the BidTimeStamp is also set so that the AuctionHandler will know when the bid was placed. This means that if two bids are equal the person who placed the bid first will take preference.

The sortBids() method sorts the bids using a simple bubble sort algorithm, comparing bids, and if the bids are equal comparing the Bid time stamps. All the methods are synchronised so that any deadlocking problems are prevented. The Java system takes care of all the synchronisation of users placing bids.

4.15) How are the Items stored?

The items are stored in the ItemInfo object. An array of ten elements (or less) of these objects is created. The user indicates the amount of elements in the array. Each stores the following information.

Item Variables	**Example**
Item ID	ITEMAA1
Min Value	250
Item Desc.	A lovely UND emblazoned leather diary.
Winner Details	"AUCID001 Joe Soap,SoapJ@email.com

The Item ID is used to differentiate between different items. Each item has a reserve minimum bid price, stored in the Min Value variable. There is a string description of the each item. This is used to give the user a brief description of the item on the bidding web page. Once the auction is over the Winners Details are stored. This stores all the information held in the BidInfo object of the winning bid. When the auction is over it writes all the relevant information to log files so that the administrator has the desired contact details of the auction winners.

4.16) How is the Security implemented?

The security of an application cannot be viewed as just the security of its communications. The entire emphasis of this project was on creating an overall secure application. For this very reason the methodologies described in *chapter 3.9 Java Programming Security Techniques* have been closely followed in the development process thus far.

This means that the security of an application as a whole is based on the secure communications and programming techniques.

The security of the communications hinges on two very important features. The first is the cryptographic methods used, and the second being the way in which they are implemented. A discussion on the choice and implementation of the cryptosystem will be dealt with in the following four chapters, as well as the implementation. The system chosen uses RSA to set-up a secure communications channel and once this has been achieved the client and server share the same key for MARS or RC6 algorithms.

When the administrator starts the application, the RSA object automatically generates new RSA keys. One is public which is sent to the client and the other is private which the server uses. The default symmetric cryptosystem is RC6. The administrator can change the default symmetric cryptosystem. However this must be done before the server is started. Once the server has been started one may not change the cryptosystems.

Hint: To change the symmetric cryptosystem, Click Security menu | Select Algorithm

When a client joins for the first time the server sends the client the RSA public keys and informs the client what symmetric cryptographic algorithm to use. This message is sent as normal with no special encryption. The client has a RSACrypt class. Which is used for encryption and decryption of RSA. The RSA public key is used by the client to encrypt a random symmetric key, which is used for either RC6 or MARS encryption. The encrypted random key is then sent to server. The ClientHandler on the server decrypts the RSA message from the client using its private RSA key and obtains the symmetric key. The symmetric key is different for all clients.

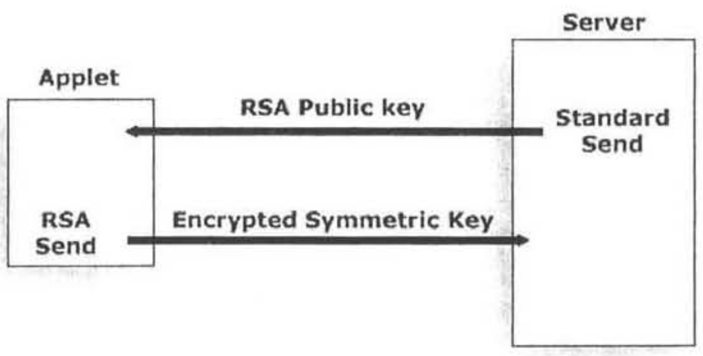


Fig 4.16.1 Exchange of public and symmetric keys

Thereafter all communication is secure. As the messages are sent they are encrypted and when they are received they are decrypted. The class MARSCrypt and RC6Crypt are used for encryption/decryption of MARS and RC6 algorithms respectively. The CryptLib class is used for encoding strings to and from bytes and integers, which are uses by both the encryption classes.

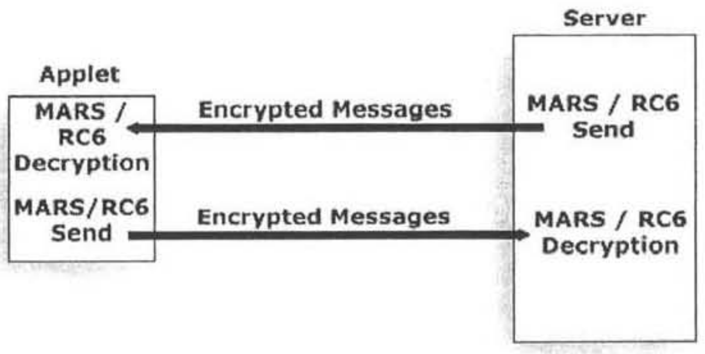


Fig 4.16.2 Secure Communication

CryptLib.Java

The client and the server have a copy of the Cryptlib.Java file. This contains useful methods for the encryption and decryption of information.

On initialization the CryptLib class creates instances of both the symmetric key cryptosystems (RC6 and MARS), as it is not known which one will be used. The Server decides which cryptosystem is going to be used. This is selected in the Security Menu. A variable called Algo_No in the AuctionInterface classes holds an integer number representing the current algorithm (0 = RC6, 1=MARS), this is passed to the CryptLib class which records what algorithm is being used in the encrypt_mode variable.

Thereafter whenever the client or server asks for interaction with current cryptosystem, such as setting the key, encrypting or decrypting, the CryptLib class knows what to do for each cryptosystem.

As an example, when the client sets the symmetric Key for the current auction, they call the *setKey* method.

```
//CryptLib.Java setKey(Users Key)
switch (encrypt_mode)
        case 0 : return RC6.generate_Keys(Users key, key length)
        case 1 : return MARS.generate_Keys(Users key, key length)
End
```

The class determines which algorithm is being used and then calls the correct generate_Keys, which generates the required key.

The decrypt and encrypt methods work in a similar fashion

```
//CryptLib.Java Encrypt(Mesg, Key)
switch (encrypt_mode)
        case 0 : return RC6.RC6Encrypt(Mesg,key)
        case 0 : return MARS.MARSEncrypt(Mesg,key)
End
```

The CryptLib class has a number of other useful functions. It can generate a random key using a seed from the user, a random number and the MD5 hashing algorithm

```
//CryptLib.Java randKey(Size, Random Values)
generate random bytes[size]
MD5.update(bytes)
Return MD5.digest(Random Values)
```

The user provides random values, which are used to encrypt the random data, using MD5 (See Appendix).

The CryptLib class provides some functionality for both the symmetric cryptosystems. There are methods for converting an integer to bytes and vice versa. One can convert strings to bytes, and bytes to strings.

As all messages are strings, they are converted to bytes, and then from bytes to equivalent integer values. These integers are then encrypted by the cryptosystem. When decryption is complete, the process converts decrypted integers into bytes, and then the bytes into the original string format.

4.17) What problems are associated with the threads?

An understanding of what threads are is probably a good place to start before anything else may be discussed. A thread is shorthand for a thread of control. This thread of control is the simplest element of a program that can execute independently, [Oaks & Wong, 1999]. Even the most simple of applications has a single thread of execution. However when one starts dealing with streams and user interfaces it becomes easier to break a program up into separate procedures to deal with asynchronous stream communications.

Ever since the inception of a multithreaded environment there has been competition for the available resources. When threads demand access to resources at the same time it can lead to confusion and incorrect results. As an example lets take two threads: Thread one and Thread two. Both are competing for the output stream to the monitor. Thread one attempts to output "Hello world", and thread two attempts to output "This is a test". As they are fighting for control of the single resources the result is a jumbled output of the two threads.

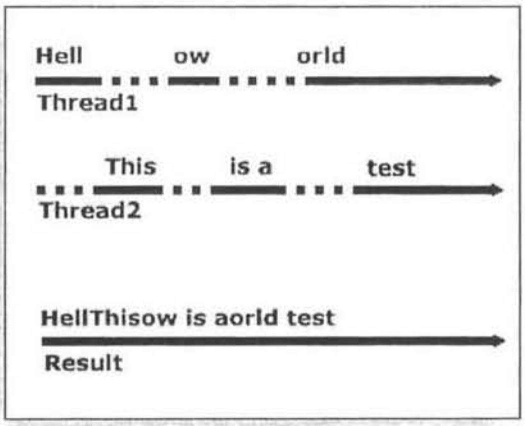


Fig 4.17.1 Threads fighting for the same resource.

If however there is some form of control of the threads, for access to the resources. this problem would be alleviated.

A semaphore is one way in which one may control the threading issue. A semaphore allows only one thread to gain access to a resource at a time. All other threads wanting to gain access are put to sleep until the current thread is finished. The next thread in the queue is then allowed access to the resource. To achieve this affect Java uses synchronization to control the paths of execution of threads. The way in which one may use this feature is quite simple. By declaring a common method synchronized, any threads attempting to use the method will have to wait until it is free.

Admittedly threading issues are far more complex than indicated here, and Java's support for a multithreading environment is just as complex. However the synchronization of methods and to control access to a resource is sufficient for this application and its demands.

4.18) How does the server use synchronization to update all the clients?

As has been mentioned beforehand the server has a list of all the clients currently connected. This is the client list vector, which is a dynamic array of object references. A Vector can dynamically grow and contract as required.

There are two methods that are used, the addElement(Object) and removeElement(Object). The addElement method adds a reference to the current object at the back of the Vector. The removeElement method removes the object in the Vector. It does not require its index as it searches and removes this reference. The power of Vectors arises at this stage, as the removed element would cause an empty index to exist in the array. However the Vector re-packs all the elements in the list to consume the empty index. When a client joins or leaves the server they are added or removed from the client Vector. These method calls are synchronized so only one update occurs at a time. The adding of a client takes place once they have logged into the server. The removal of a client occurs when the communications channel is closed between the client and the server.

The objects that are referenced in this list are the clienthandler objects for each client. When the Auctionhandler determines that an update is required, it calls the method updateAll(Message Type,Message). An update is required on three occasions, one when a new top bid is received for the current auction and the final two when a state change has occurred. (Start or stopping of an auction).

```
//AuctionHandler.Java - updateAll(Message Type,Message)
synchronized(client_list) {
        Enumerated = client_list
        While (Enumerated.hasMoreElements())
                (ClientHandler)Enumerated.nextElement .
                notify_Client(Message Type,Message)
        End while
}
```

The first thing that the method requires is the message type or number, and the actual body of the message. The client_list is then synchronized. This is done, as one does not want users joining or leaving as the AuctionHandler traverses the client Vector. Once the method has control of the client vector it first creates an enumeration of the Vector.

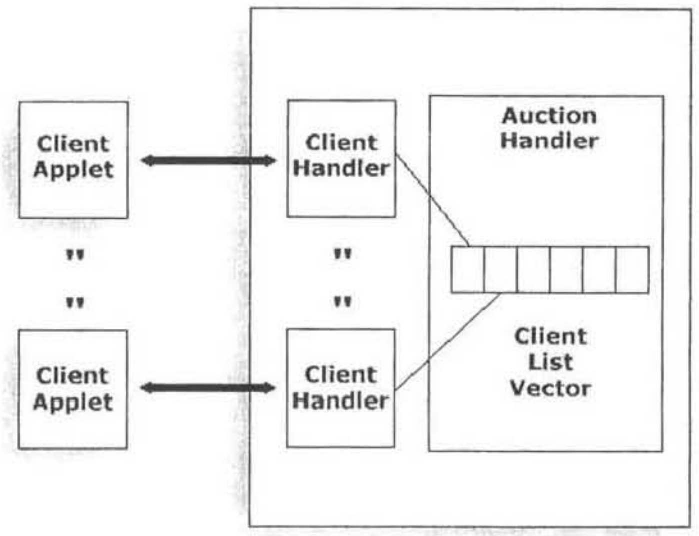


Fig 4.18.1 The Client Vector List, and its references to ClientHandler objects.

Once the method has control of the client vector it first creates an enumeration of the Vector. This enumeration generates a series of all the elements in the Vector. The method then extracts each element one at a time until there are no elements left to interrogate using the nextElement call on the enumeration. Once one has access to this element it is cast to a ClientHandler object. This gives one a reference to a client's Clienthandler object. Once this has been achieved the method notify_client(message type, message) is called, which deals with the physical communication to the individual client. The notify_client method has already been addressed in section 4.13.

In this way all clients that are currently joined to the server are updated with all the correct information regarding the auctions.

4.19) What logging takes places?

The server application performs logging on a continuos basis. It was originally envisaged to allow logging to be controlled, and stopped or started. However logging is such an important role for any server application that it was decided to force logging to be continuous. There are two log files, Aucserver.log and user.log.

The Aucserver log file stores information regarding the general running of the server. This includes start and stop times of the server and auctions. Any error messages generated by the server will be recorded within the file.

The user log file stores information about the past winners of auctions. This includes all their details, the bid made, the items involved and the date it all occurred.

As the application runs continuously it was decided to append information to the log files. If these files do not exist, they are then created and thereafter they are appended. This means that there is a continuous history about the server application as well as all the auctions run.

4.20) How are the directory structures set-up?

The directory set-up is straightforward. The server application can be installed anywhere, but the children directory structure must be as follows. The server root directory holds all the classes required for execution. This directory requires execution privileges for the user, but no write privileges.

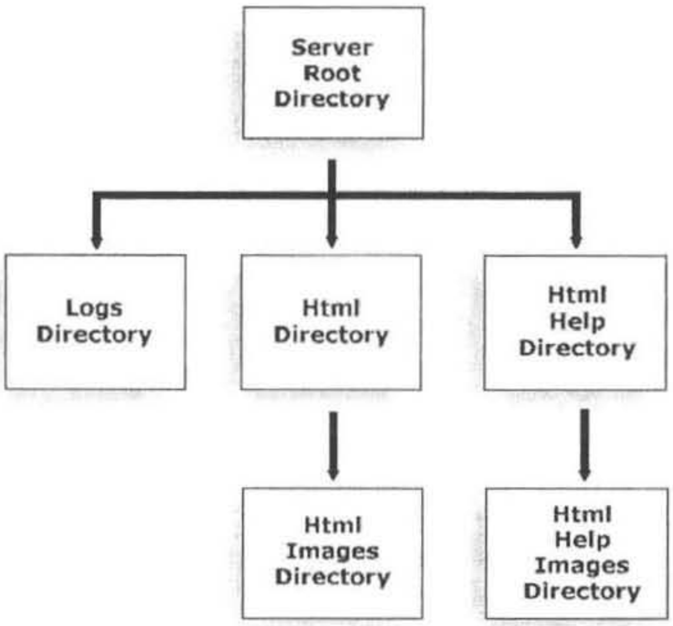


Fig 4.20.1 Directory structure

There must be a logs directory called "logs". This is used to hold the log files that the application writes to. This directory does not need to be readable or executable, however the server must have write access to this directory.

The "html" Directory holds all the HTML files for the web sever, as well as the applet JAR files aucjar.jar. The HTML "images" directory holds all the images for the HTML pages and the images of the items auctioned. This is where the applet retrieves the images it uses from. The server requires write access to the "html" directory to update the history and current auction files. A virtual web site should be set-up on this directory. The virtual web directory requires executable privileges to execute the ASP / CGI script and the Java applet file. One must not allow write privileges in this directory as this will compromise the security of the application.

The "htmlhelp" directory holds the help for the server application in HTML files. The required images are placed in the "images" directory. The application needs read access to this directory. There is no need for execute or write privileges. Note that in the Windows environment the directories are not case sensitive. In Unix all lower cases are used.

The server may be set-up anywhere on the system, and will require the directory structure as described above. If these requirements are not met it may lead to unpredictable results. The privilege requirements have been listed as read, write and execute as a guideline. Different operating systems have different or no policies regarding these privileges.

4.21) What files are used and Why?

There are four files that the AucServer uses continuously. Those are the two discussed log files and two HTML include files. These are files included when HTML pages are displayed. Specifically auchtm.inc and history.inc.

Auchtm.inc file

The auchtm.inc files includes a list of all the items to be auctioned. This file is generated dynamically by the server application. Every time an auction begins or ends, the server rewrites the file. It displays the file in the following format:

<Auction ID>	<Description>	<Minimum Bid>	<Finish Time>
ITEMAA1	A lovely UND emblazoned leather diary.	250	Wed Jan 12 13:58:26 GMT+02:00 2000
Actual HTML text <tr><td> ITEMAA1</td>	<td> A lovely UND emblazoned leather diary.</td>	<td>250</td>	<td> Wed Jan 12 13:58:26 GMT+02:00 2000</td></tr>

It displays the auction ID number, the brief description, the minimum bid and the finish time. The actual file includes all the required HTML text. The HTML file that includes this file is the auctions.asp page (auctions.html in Unix). It uses the HTML code:

<!--#INCLUDE FILE="auchtm.inc"-->

When a client views the auctions.asp file to get a listing of the available auctions, the auchtm.inc file is included into the current document. This means that the application does not impact on the design of the web pages whatsoever.

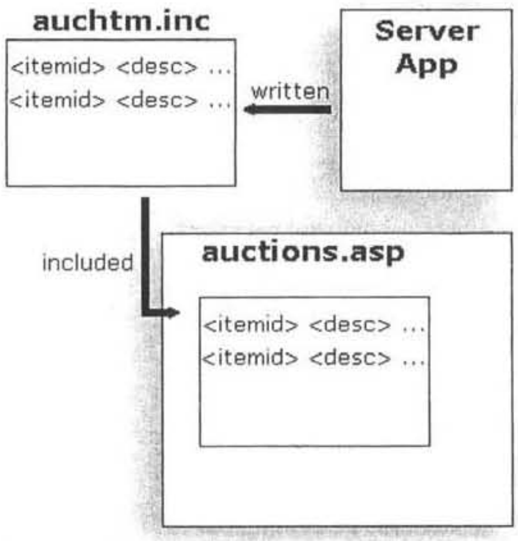


Fig 4.21.1 auchtm.inc file being included into auctions.asp.

If one of the items is currently being auctioned, this page displays an HTML hyperlink to the itemauc.asp page. As an .asp file is designed to work in the Windows environment using VBScript it will not work in a Unix environment. The server detects what system is being used and will appropriately use the correct auctioning page. This is itemauc.asp for windows, and itemauc.cgi for Unix, as both are used for scripting on their respective systems. The item id value is highlighted as the actual link to this page, which executes client applet. This link is generated dynamically by the server and is created in the following, manner:

<A href=itemauc.asp?port=<Port number>&itemid=<Item ID>> Item id </a>

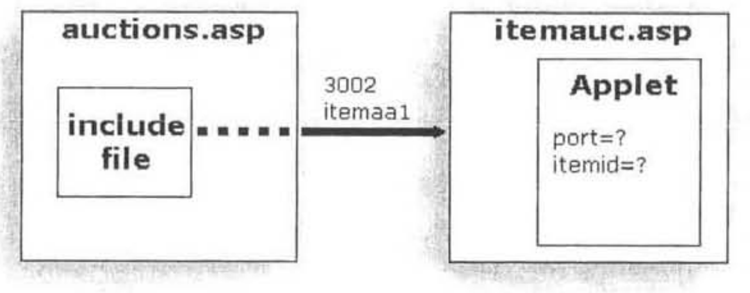


Fig 4.21.2 Passing parameters to applet.

This results in the itemauc.asp page being called. Two parameters are passed to into this web page. These parameters are then passed onto the client applet residing in the page. The parameters are Port Number and Item ID. The applet requires the port number parameter so it knows how to communicate with the server. The Item ID is for the item currently being auctioned. This is required by the client's applet as it must know the item the client originally bids for. The original way in which one added an applet to a web page was to embed it into the HTML code as follows:

```
//Itemauc.asp
<Applet Code = "AucClient" Archive="Aucjar.jar" Width="350" Height="450">
<Param Name = port          Value = <%=request.QueryString("Port")%>
       Name = itemid        Value = <%=request.QueryString("itemid")%>>
</Applet>
```

This would run the file AucClient.class that is found in the JAR Aucjar.jar. The width it would be displayed in is 350 and height 450. The parameters passed in are the port and item ID numbers. To extract the parameters from the querystring, VBScript is used. On a Unix system Perl can be used instead (A sample itemauc.cgi is included with the source).

However the with Java 2 available the type of embedded applet command has changed. Sun Microsystems have released an application ("HTML Converter") that will update the previous HTML code to the correct version implementation. The application can be downloaded for free from their web site (Java.sun.com). The reason for doing this is that the client will automatically try and update the Java version if it is not compliant with the current applet being used. This allows for a more robust system.

The auctions.asp file is reloaded every fifteen seconds so that the user will see any changes in auctions.

History.inc file

The final file that the server writes to is the history.inc file. This file gives a complete history of all the previous winners, their bids, the Item ID and the time it was achieved. The HTML file that uses the history.inc is the history.asp file. All it does is include the history.inc file, so that when the HTML file is processed so is the include file.

<!--#INCLUDE FILE="history.inc"-->

<Auction Number>	<Winners Details>	<Winning Bid>	<Finish Time>
ITEMAA1	AUCID001 (Joe Soap)	350	Thu Jan 12 13:01:15 GMT+02:00 2000

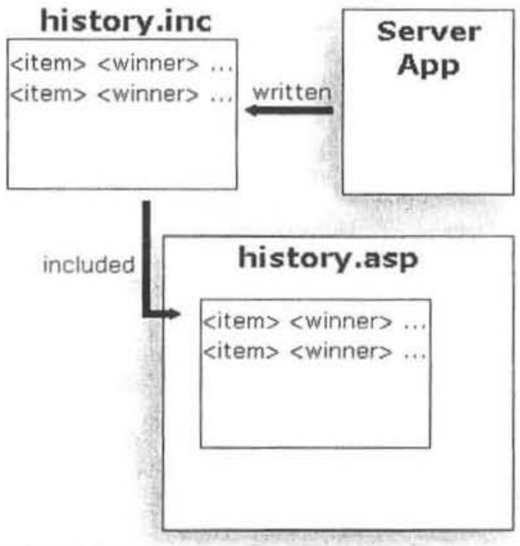


Fig 4.21.3 history.inc file included into history.asp

Only the ID value and name of the client is given and not any personal details. This file is appended unlike the auchtml.inc file. So it will not overwrite any of the previous information, thus giving a complete history of all the past winners. The way in which the server interacts with files has been developed such that it has no effect what operating system or file structure is being used. The development of the web pages needs only make one or two small concessions to incorporate the servers file usage to allow the auctions to take place.

Porting

Admittedly there is a bit of preparation required to set-up the server but the source code comes with the examples HTML files. These HTML files use ASP and reside on a Win32 web server. All the scripting is currently performed using VBScript. However it is not difficult to port the ASP VBScript to a language such as PERL or JavaScript if one desires. The only thing that is actually required is to replace the code below with the appropriate script to extract the Querystring parameters port and itemid. *<%=request.QueryString("Port")%> & <%= request.QueryString (" ItemID ")%>*.

The source has been included for the Windows and Unix versions.

Conclusion

The aim was to create a robust and secure Internet based application. This application must be system independent and extendable. The application surpassed what was expected performing admirably well in testing and development stages. Java showed its true potential in the application with its natural support for Internet based applications and applets. The seamless capability of Java to interact within a networked environment holds great promise and should bode well for its future as a development language of choice.

Using object orientated programming techniques and the power of the Java language I believe all goals were met admirably. Not only did it meet all the goals, but also, with the increasing popularity of Internet based auctions, the application has real world marketable potential.

In the next few sections I will discuss the security issues in greater depth.

5) Security

"...the security of information systems and networks (is) the major security challenge of this decade and possibly the next century...there is insufficient awareness of the grave risks we face in this arena."
[Joint Security Commission, www.nsa.gov]

The mathematical science of securing information is known as cryptology. Cryptology's rise in importance is due mainly to the advent of the digital age. The original Greek terms are Kryptos ~ graphia, essentially Kryptos meaning hidden, and graphia meaning writing. There are four goals that cryptology strives for:

Confidentiality

The content of the information and messages must be kept confidential.

Authentication

Determine the identity of the sender of information.

Integrity

To validate that the information has not been tampered with during transmission.

Non repudiation

Prevent the sender from denying that they sent the message, at a later date.

5.1) Terminology

The two parties involved in the communication are usually known as Alice and Bob. Eve is the undesirable attempting to eavesdrop on the communication channel. The original information to be encrypted is called the plaintext (P). The encrypted information is called ciphertext (C). Both plaintext and ciphertext are stored in the standard digital format (Binary, Hexadecimal etc.). Encryption is any method by which plaintext is converted into ciphertext. Similarly decryption is any method by which to convert ciphertext into plaintext

Encryption and Decryption require keys (passwords, PINS or suchlike) to perform the procedures. The keys (K) help lock (encrypt) the information, as well as unlock (decrypt) the information. The encryption key (K_1) and the decryption key (K_2) may or may not be the same. This depends on the cipher (algorithm) being used. There are two types of algorithms:

Symmetric Algorithms

The same key is used for encrypting and decrypting the cryptosystem. This is also known as a "symmetric key" system, e.g. DES (Data Encryption Standard). Both parties (Alice & Bob) agree on the same key before any communication can take place.

$E_k(P) = C$
$D_k(C) = P$

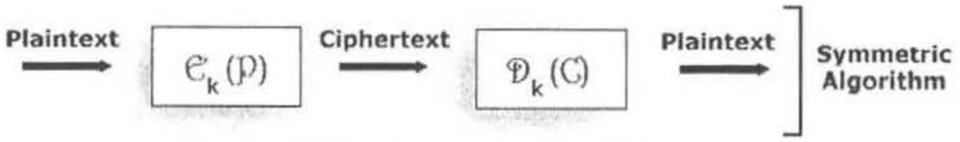


Fig 5.1.1 The Symmetric Algorithm

Asymmetric Algorithms

Different keys are used for encryption and decryption. The encryption key is usually known as the public key, and the decryption key is usually known as the private key. This is also known as a "Public key" system, e.g. PGP (Pretty Good Privacy). No information can be gathered from the encryption key about the decryption key.

$E_{k1}(P) = C$
$D_{k2}(C) = P$

The key K_1 is not the same key as K_2.

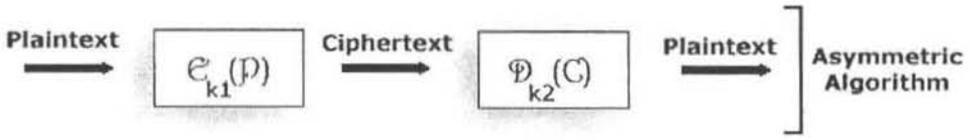


Fig 5.1.2 The Asymmetric Algorithm

Each algorithm attempts to have at least one of the following properties, but preferably both.

Confusion

Confusion is the process by which the algorithm hides all relationships between the ciphertext and plaintext. It makes it difficult to conclude anything about the plaintext from the ciphertext.

Diffusion

Diffusion attempts to hide any statistical characteristics in the plaintext. This is required as in most languages each letter has a different probability of occurring in any text.

Letter	Probability	Letter	Probability	Letter	Probability	Letter	Probability
A	0.083	H	0.161	O	0.075	W	0.023
B	0.015	I	0.070	P	0.019	X	0.001
C	0.028	J	0.002	Q	0.001	Y	0.020
D	0.043	K	0.008	R	0.060	Z	0.001
E	0.127	L	0.040	S	0.063		
F	0.022	M	0.024	T	0.091		
G	0.020	N	0.067	U	0.028		

Fig 5.1.3 Probabilities of individual letters in English. [Stintston, 1996]

As one can see from the table some letters are more popular than others in the English language. The letter "E" is the most popular, with it contributing approximately thirteen percent of any written English text. Using statistical evaluations of vast amounts of ciphertext of the English language could possibly yield results if the algorithm does not perform any sort of diffusion. One must bear in mind that most cryptoanalysts have some form of probabilistic idea about the plaintext, as they will understand what type of information is to be encrypted.

Cryptography is divided into two sets of practitioners, cryptographers and cryptanalysts. The former creates systems to secure information. The latter attempts to break these systems and find faults in them. There are a number of attacks that can be mounted on any cryptosystem they are as follows:

Ciphertext only attack

The cryptoanalyst only has a few ciphertext messages generated by the system.

Known plaintext attack

The cryptoanalyst has ciphertext generated by the system as well as its corresponding plaintext messages.

Chosen plaintext attack

The cryptoanalyst chooses the plaintext and is able to generate the corresponding ciphertext.

Chosen ciphertext attack

The cryptoanalyst is able to generate the plaintext for a given ciphertext.

Each of the attacks attempts to find the keys used by the cryptosystem. There are a number of assumptions for each of the attacks. It is assumed that the cryptoanalyst knows the cipher being used. This is known as Kerchoff's principle. It is generally considered a very poor option for the entire system to be based on the secrecy of the algorithm. This is security through obscurity, which history has proved never works. It has been shown that the strongest algorithms are the ones that have withstood the scrutiny of the industry.

5.2) A formal definition

The following definition is due to [Stinston,1996] formulates it as a mathematical concept.

Definition: A cryptosystem is composed of the following four conditions;

- P is a finite set of plaintexts ($p \varepsilon P$)
- C is a finite set of ciphertexts ($c \varepsilon C$)
- K is the key space, which is a finite set of keys
- For each $k \varepsilon K$, there is an encryption and decryption rule
- $E_k(P) = C$, $D_k(C) = P$,such that $D_k(E_k(x)) = x$ for every plaintext $x \varepsilon P$

An immediate consequence is that each encryption must be injective. If this was not the case one could end up with the following result:

$C = E_k(x_1) = E_k(x_2)$,$(x_1 \neq x_2)$

This is potentially disastrous, since not only would the cryptoanalyst not be able to find the correct plaintext, but neither would the valid parties (Alice and Bob).

The definition of an asymmetric, or public key cryptosystem as related by [Stinston, 1996] is as follows.

Definition: A cryptosystem is composed of the following conditions:

- P is a finite set of plaintexts
- C is a finite set of ciphertexts
- K is the key space, which has a finite set of key pairs (k_p, k_s), where (k_p is the public encryption key, k_s is the private decryption key)
- For each (k_p, k_s) ε K, there is an encryption and decryption rule
- $E_{kp}(P) = C$, $D_{ks}(C) = P$,such that $D_{ks}(E_{kp}(x)) = x$ for every plaintext $x \varepsilon P$
- For each $x \varepsilon P$ and (k_p, k_s) εK , $E_{kp}(x) = y$ should be feasible
- Given y, it should be infeasible to calculate x with only the knowledge of y and k_p
- Given y, it should be feasible to calculate $D_{ks}(y) = x$

One of the main criteria with public key cryptosystems is that no information must be gathered about the plaintext with just the ciphertext and encryption key. Most of the public key cryptosystem are based on what are believed to be **one way** functions.

$F(x) = y$, with $F^{-1}(y) = x$ infeasible to calculate.

The entire field is usually based on this type of premise. However there are currently no proofs that one way functions actually exist.

The security of cipher can be classified as computationally secure or unconditionally secure. Almost all ciphers fall into the former category.

Computational security

Computational security means that the resources required to break the cryptosystem are relatively infeasible at this time. The amount expended attempting to break the cipher, in time and fiscal terms, would be unprofitable. This does not mean that the cryptosystem is absolutely secure and that it cannot be broken. This measurement is something that needs to be re-evaluated constantly due to the huge advances in computing power. A cryptosystem that is computationally secure today may be thoroughly insecure in five, ten or twenty year's time. An example of this is the DES cryptosystem. In 1977 the NSA (National Security Agency) supported it, however by 1996 the NSA had withdrawn all support of DES. A year later DES had been broken in a measly fifty-six hours, a year later it was done in twenty hours on a hundred thousand-dollar machine. In 1977 it would have taken years of computing time, as well as millions of dollars to achieve this feat.

Unconditional security

Unconditional security places no bounds on the amount of resources and time that may be used to mount an attack on the cryptosystem (An impossible situation). Nonetheless even with all these resources the cryptosystem would never be broken. There is only one pure unconditional secure cryptosystem. This is the "One Time pad", which has a random key as long as the actual message. For this reason it is a totally infeasible cryptosystem. It is difficult to quantify the security of algorithms, as one deals with infinite resources, and hence one uses probabilities to measure a cipher.

5.3) The Protocols

The protocol is the actual implementation of the ciphers in a secure and efficient manner. When one implements a protocol all aspects of the ciphers must be looked at first. Once all points have been weighed up, all parties involved must know what protocols are being used and have an identical implementation to avoid any conflicts.

What is required?

The auctioneering application requires secure communication over the Internet. This needs to be relatively quick and efficient so it can run on your average home processor, as this is the targeted market of users. It also needs to be invisible to the user and must not hamper in any way the actual running of the auction system. The client and server need to communicate securely and quickly. It is important to point out that most of the messages sent from the client to the server are relatively small and this will have an impact on the cryptosystems used. The lifetime of messages is not long, as the auctions are relatively short. All keys used will be dynamically and randomly chosen. The application regenerates keys on instruction and there are no hard coded keys.

There are four things that need to be targeted on the Internet to enhance security; Improved cryptographic algorithms, efficient cryptographic processing, routing and multi-party cryptography [S.M.Bellovin, 1998]. The scope of the latter two is outside of the current topic.

Currently there are only a few individuals working on algorithms that will be successful in the future. It is relatively easy to throw together a block cipher, but this does not mean that it is going to be any good at all. The science of cryptography has been described as more of an art. Even though an algorithm may seem to be secure it may have inherent weaknesses. It does not take much for one to realise why people with a strong background in theoretical mathematics, number theory and information theory, not to mention extensive experience, do relatively well in the industry. This is the primary reason why I decided not to implement my own algorithm, but to look at a mix of proven and new algorithms available. Even with all the experience and support available one's algorithm is not tried and tested until it has stood up against the industry for a number of years. Each of the algorithms discussed in later chapters has a number of benefits that will be highlighted in the text that follows.

The efficient processing of cryptographic algorithms is where a number of problems occur, and, as one will see later where most applications can fail. The incredible advancement of technology is facilitating the improvement of algorithm implementation. The difficulty arises when one attempts to implement an algorithm that was never intended for use on any form of network. Attempting to place an implementation of an algorithm into a twenty-five year old protocol is no easy feat. There are varying arguments for and against different implementations of algorithms and the problems that may be faced but these will be dealt with later.

Each of the symmetric and asymmetric algorithm types have benefits and disadvantages.

Symmetric Algorithm	Asymmetric Algorithm
Fast	Slow
Single key	Separate keys for encryption / Decryption
Smaller key	Larger keys

The biggest problem with symmetric algorithms or ciphers is the fact that the single symmetric key needs to be securely shared with Alice and Bob. If Alice and Bob already have this secure communications channel for sharing keys what is the reason for implementing the cipher? As one can see symmetric cryptosystems would be ideal for Internet applications and hence the current problem faced.

As symmetric ciphers are on the whole about one thousand times quicker than asymmetric algorithms it makes sense to use them for bulk encryption of information. Even though processing power has increased considerably over the past few years, and will continue to do so in the future it is still not feasible to use public key cryptosystems for bulk communication due to their general sluggishness. However as the question has already been posed it is difficult to exchange the single key discreetly and securely. A solution would be for Alice and Bob to physically communicate the key via telephone, courier or some other physical means, but in many cases this is impractical, especially over the Internet which has released client the from physical limitations. Not only is it impractical, but if any person listens to the conversation or views the message as it is been couriered then the entire security of the algorithm is under threat.

The answer lies in the fact that one may use the asymmetric cryptosystem to resolve some of these issues. Using the best of both worlds as it may be. The asymmetric cryptosystem as has been pointed out is rather slow. However it does not need a single key. There are invariably two keys, a public and private key. The public key is distributed to everyone that wants to speak to Alice, in other words it is made public. As the algorithm has been agreed on, anyone who wants to send Alice a private message need only encrypt the messages using the encryption algorithm with the public key. Alice is the only one that can decrypt this message with the associated secret key. The solution then is to use the asymmetric cryptosystem to share, secretly, the symmetric key between Alice and Bob. This is known as a hybrid cryptosystem.

Hybrid Cryptosystem

In a Hybrid cryptosystem more than one cryptosystem is used. The public key cryptosystem is used to create a secure communication between Alice and Bob. Once this has been achieved, Alice and Bob can then generate and send the symmetric key to one another. Once they have shared this key using the asymmetric cryptosystem, the cryptosystem is then changed for the remainder of the communication session. The symmetric cryptosystem is then used for encryption as both Alice and Bob have the same symmetric key.

Initially one has to bear with the slower performance of the asymmetric algorithm to take advantage of its key sharing ability. Thereafter one may switch to the faster symmetric cryptosystem for the bulk of communication.

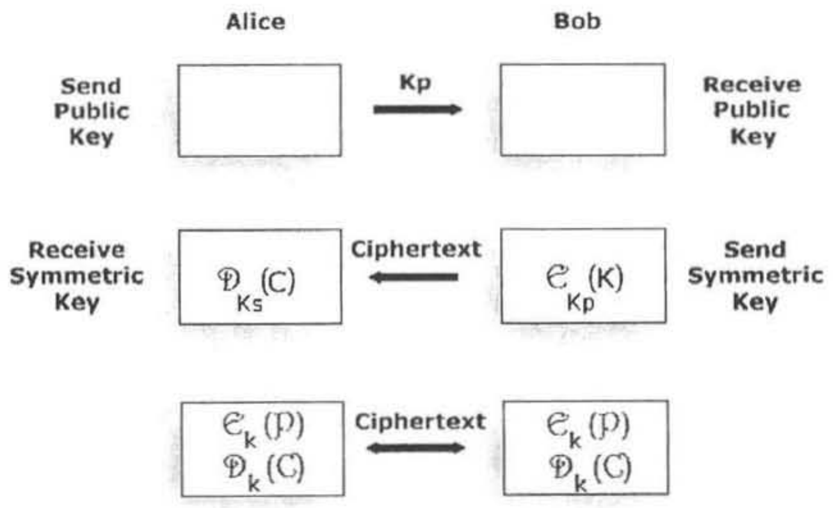


Fig 5.3.1 The process of key exchange

There are a number of problems that exist. One may ask the question, why doesn't public key cryptosystem become more popular? If speed is an issue it will be resolved as technology evolves. Within the next ten to fifteen years technology will be fast enough for it not to matter about the strain that encryption places on the system. However public key systems are also weaker, relative to their symmetric counterparts. As the encryption function and key are available, the system is susceptible to a plaintext-mounted attack. Therefore most public key cryptosystems require larger keys to make the system relatively stronger against this line of attack. However this does allow an exhaustive search technique by which the cryptanalyst can attempt to determine the private key.

However all is not lost as public key cryptosystems are designed with this fault in mind. Its security is based on two things, the unfeasibility of determining the private key with knowledge of the public key, as well deducing the plaintext given the ciphertext [B.Schneir, 1996].

Martin Hellman and Whitfield Diffie developed public key cryptography in 1976. Their idea was to remove this key exchange problem altogether. Each person has their own set of public and private keys, they release their public keys, and use one another's public keys to encrypt and send information. Similarly they use their own private key to decrypt the message sent. Using this method no forms of secret key exchange takes place.

RSA, RC6 and MARS

The three cryptosystems used in this project were RSA, a well known and tested asymmetric (public key) cipher, RC6 which is a symmetric cipher developed and advanced upon RC5, and finally MARS which is the latest symmetric cipher. It must be mentioned that RC6 and MARS are part the Advanced Encryption Scheme (AES), which is controlled by the American National Institute of Standards and Technologies (NIST) to develop a replacement for DES. The usage of these algorithms in this project has a number of implications that will be mentioned later.

RSA is used for the key exchange, and either RC6 or MARS are used for secure communication.

5.4) Provable Security

The problems that most security applications face these days are vast. It has been shown that in most security breaches it is not the actual algorithm that is at fault, but it usually is a problem with the implementation of the algorithm, usage of the algorithm and the keys selected in the algorithm. This is usually due to a poor understanding of the algorithm and essentially the ineptitude of the implementers. Any problem with implementation is going to give a cryptanalyst another possible point of attack, and will generally weaken the system as a whole.

The idea is to find a fine balance between the practical implementation and the mechanisms of the cryptosystem. The hybrid cryptosystem discussed above using

RSA and RC6, or RSA and MARS is the current secure protocol for message passing systems on the Internet.

However what is required of a good secure cryptosystem as related by [M Bellare, 1998] is good primitives. The atomic primitives are the foundations of the protocols used. The idea is to base the protocol on good primitives, resulting in a good protocol. One needs to question how good these primitives are? How secure are they? Are they suited to the specific protocol? This will be done in each of the following chapters concerning the relative algorithms.

How do you prove "Provable Security"?

This task was formerly proposed by Goldwasser and Micali [M Bellare, 1998]. The idea is based on a mathematical notion and there are a few steps. First one needs to model a secure system and conceptualise the points required for this secure system. In a similar fashion one models the current protocol. Using reduction, and breaking down the protocol one looks at all its weaknesses. If a weakness is found in the protocol then the entire protocol needs to be re-assessed. Once the protocol has been reduced to its primitive elements, the security of the protocol then rests entirely on the security of these primitives.

The only way then to break the protocol is to break the primitives it is based on. Then cryptoanalysts will focus on these primitives and if they are secure then the entire protocol is secure. Needless to say one has to prove the security of the primitives, and before this can be done one must understand what the primitive is designed to do. Just like a matchstick bridge would not serve the purpose of supporting lorries, nor will a cipher work in an environment it was not developed for.

To prove that a protocol is sound or secure, one uses the Complexity theory approach. A discussion on this follows in the next section. However this theory does not actually look at secure practical physical implementation of the protocols, rather it examines a theoretical approach. The practice orientated provable security methods attempts to quantify the security of the physical protocol as discussed by [M Bellare, 1998], thereby giving it a measure of security. This is made up of two measurements; the degree of insecurity, which is measure by how easy it is to attack the protocol and the degree of security which is measured by the reduction of the protocol and primitives. The reduction of the elements allows one to break the protocol down to its most basic elements and examine the security of each. Admittedly one must realise that when one model's a protocol there are a number of assumptions that need to be made. Due to the fact the real world cannot be entirely modelled, all circumstances cannot be foreseen. The models of the algorithms discussed will show how each of the cryptosystems is implemented, where their strengths and weaknesses are and where possible lines of attack may be seen. There are always the attacks that are not seen, which can brutally destroy a protocol, but bear in mind the security model derived does not prove that it is secure against the unknown, just the known with a number of assumptions. If any of the assumptions are wrong and the unknown attack occurs, the entire security model collapses.

5.5) Complexity Theory

Complexity theory has been developed to examine the complexity of mathematical formulae. It hopefully leads to some measurement of the formulae, and hence its security.

One needs to determine the complexity of an algorithm to deduce any further information about it. This is measured by the time and space the algorithm requires. The resultant value is expressed as the order of magnitude of the complexity of the algorithm. Both time (T) and space (S) are functions of n, with n being the size of the input of the algorithm. The order of magnitude (O) is measured as the biggest value of n, as n increases. This is because as the n tends to infinity, the highest order of magnitude will count the most. This order of magnitude then shows one, the effects of a bigger input, on the time and space required for the algorithm. [B.Schneier, 1996]

$T = O(n^2) < T = O(n^3)$

Algorithms can be of a number of different equations, depending on O. These range from constants, to polynomials to exponential order of magnitude. Algorithms are usually classed as some time complexity equasion (i.e. Polynomial-time equations $O\ (n*a))$. The aim is to find some function that will be able to break the cryptosystem. New algorithms are designed with this in mind, and attempt to make this breaking algorithm of exponential-time complexity, thereby making it incredibly hard for one to find a short method for breaking the cryptosystem. However this theory does not prove that there is no existence of a trapdoor algorithm that can quickly and easily find a solution. The following table shows why only the higher order of magnitude of n is considered. Once the order of magnitude becomes a polynomial equation, breaking the system becomes even more difficult.

Class	Complexity	No. of Operations, $N = 10^6$	Time of operation
Constant	$O(1)$	1	1/1000 sec
Linear	$O(n)$	10^6	1sec
Quadratic	$O(n^2)$	10^{12}	11.6 days
Cubic	$O(n^3)$	10^{18}	32000 years
Exponential	$O(2^n)$	10^{301030}	10^{301006} age of universe

Fig 5.5 Running times of different algorithms [B.Schneier, 1996]

There are a number of problems with this method as discussed by [E. F. Brickell & A. M. Odlyzko]. As it is a measurement of the actual algorithm and the time taken to break it, it cannot deduce what may occur once the cipher is implemented and has external influences. Secondly this theory is based on the experimental evidence about the algorithm. The way in which this is done is by showing how quickly the algorithm is broken with fewer rounds and then extrapolating this figure out for all rounds. These are really just assumptions.

Complexity of Problems

This theory is used to discern the complexity of the actual cryptosystem. All this is based on the work of Alan Turing, who invented a non-deterministic finite state machine, commonly known as the Turing machine.

Problems can be divided into two groups as discussed by [B.Schneier, 1996], those that are feasible, and those that are infeasible. The feasible problems are usually solved in some polynomial time by the Turing machine. Those that are infeasible may or may not have a solution. These problems are divided into a number of classes: P,NP, P-Space and EXP-Time. If the algorithm is breakable, the cryptanalyst will find a P class solution. In other words it executes in polynomial time and is a feasible solution. If the problem can be solved in NP time then things get more difficult. NP time means the solution is solvable by a non-deterministic Turing machine in P time. A non-deterministic Turing machine makes the correct guesses required to solve the problem. This does put an upper limit on the time required to solve the problem. P-Space problems have been shown to be provable in polynomial space (The machines would have enough memory to handle them), but cannot be done in polynomial time. The EXP-Time class of problems can be solved in exponential time

The interesting thing here is that P class problems fall under NP as a subcategory. In other words if a solution can be found in polynomial time on a Turing machine, it can be found on a non-deterministic Turing machine in Polynomial time. Most cryptosystems fall into this NP class of problems. Thus far no one has managed to prove that the class P = NP. In doing so they would prove that any cryptosystem problem is solvable in polynomial time and all algorithms are breakable. The solutions to the algorithms would still need to be found, but one would know they existed. The world would fall into anarchy as chaos ruled supreme and all economies collapsed.

The cryptosystems that are being dealt with in this book are suspected to be of type NP hard problems. However as there is no proof of one way functions (especially when dealing with RSA), this is conjecture and thus there is always a chance that a cryptanalyst could find a solution to these problems in P, thereby destroying the algorithm.

5.6) The Process

As has been mentioned previously, RSA, RC6 and MARS are the three algorithms that are examined. However what has not been discussed is some of the preliminaries regarding the algorithms. This includes the key generation, authentication, and verification of the auction system. As is mentioned in [T Aura, 1997], "..we stress that the techniques should be implementable at a reasonable cost in computation and bandwidth. This is essential because designers of concrete protocols often struggle with tight performance constraints." The aim of this project has been to achieve a relatively secure system, with as little impact on CPU and network usage as possible.

There are two things that are done. The first is to set-up and share a public key cryptosystem with the use of RSA. The server generates RSA public and private keys on starting, or when the administrator issues an explicit command to re-create the keys. The server sends the client the public key, and information regarding the symmetric algorithm to be used. (See Chapter 4.13 and Figures 4.13.4, 4.13.5). This is sent in plaintext, thus anyone can read it. The client receives the server's public key. Once the client has this key, all messages sent to the server can be encrypted with this public key. The client can then generate the symmetric key for the symmetric cipher decided upon. Once this key has been generated (See *Keys and Key management* later in this chapter) it is then encrypted using RSA and the servers public key. This encrypted symmetric key is sent back to the server. The server receives the message and decrypts it using RSA and the private key. No other action or request from the client will be accepted until secure communications are set-up. The server extracts the symmetric session key from the message. This session key is only for the current client concerned. Clients have their own session keys for the duration of the auction. The Clienthandler object keeps the session key for the respective client. If no symmetric key is shared or there is a problem with the key the server will request the key up to a maximum of three times. Thereafter if there are still problems communication is ignored from the client. This prevents malicious flooding and denial of service type attacks.

At this stage the RSA cipher is not used anymore. Thereafter, whenever communication takes place between the client and the server it is with either the RC6 or MARS algorithm. As both the client and the server have the same symmetric key they may use the quicker symmetric cipher system. All communication is automatically encrypted on sending and decrypted on receiving. The client only uses this key for the duration of the auction. If the client leaves the auction and then joins again, a new session symmetric key is generated. It should also be mentioned that if any communication between the client and server has errors or attempts to perform illegal operations, it is ignored, the message is logged and the administrator notified. This will catch any message tampering, thereby guaranteeing message integrity for the messages accepted.

AES

In 1997 NIST made a formal call for new algorithms to replace the current ageing standards. There were a number of stipulations for these algorithms. They must be symmetric ciphers of 128-bit block size and 128 to 256 bit key sizes. Within a year fifteen AES ciphers had been submitted from all over the world.

Since then there has been analysis of these algorithms from the best cryptoanalysts around the world. A number of the algorithms displayed weakness that caused them to fall by the wayside. At the end of 1999 5 candidates still stood standing meeting all the requirements laid down by NIST. These requirements are quoted as the following:

"***Security*** is the most important factor in the evaluation. Security encompasses features such as resistance of the algorithm to cryptanalysis, soundness of its mathematical basis, randomness of the algorithm output, and relative security as compared to other candidates.

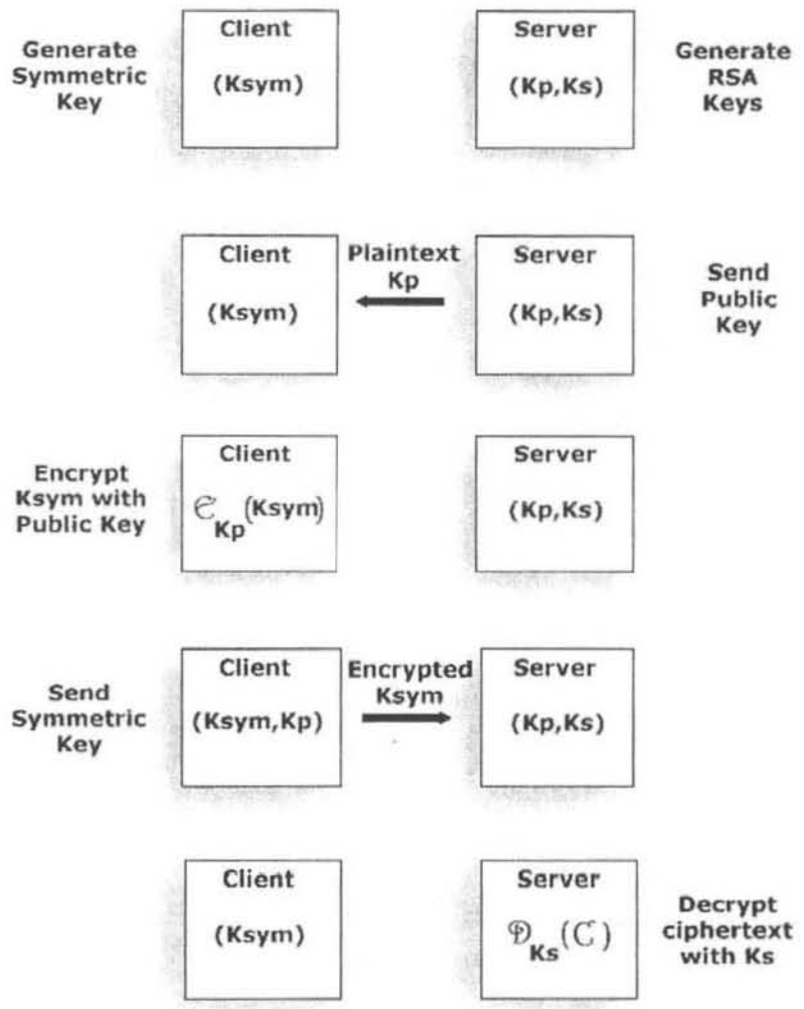


Fig 5.6 Key Exchange

Cost is a second important area of evaluation that encompasses licensing requirements, computational efficiency (speed) on various platforms, and memory requirements. Since one of NIST's goals is that the final AES algorithm(s) be available worldwide on a royalty-free basis, intellectual property claims and potential conflicts must be considered in the selection process. The speed of the algorithms on a variety of platforms must also be considered. During Round 1, the focus was primarily on the speed associated with 128-bit keys. Additionally, memory requirements and constraints for software implementations of the candidates are important considerations.

The third area of evaluation is ***algorithm and implementation characteristics*** such as flexibility, hardware and software suitability, and algorithm simplicity. Flexibility includes the ability of an algorithm:

- to handle key and block sizes beyond the minimum that must be supported,
- to be implemented securely and efficiently in many different types of environments, and
- to be implemented as a stream cipher, hashing algorithm, and to provide additional cryptographic services.

It must be feasible to implement an algorithm in both hardware and software, and efficient firmware implementations are advantageous. The relative simplicity of an algorithm's design is also an evaluation factor."

[ITL Bulletin, Aug 1999, online]

The finalists of round two were MARS, RC6^{TM}, Rijndael, Serpent, and Twofish. However, before these results were published I decided to examine and use MARS and RC6 because of all their strengths. A discussion comparing the algorithms would be out of the scope of this thesis.

Due to export restrictions granted by NIST to myself, I may not distribute or disclose any of the digital code of the two AES algorithms. However I have developed my own Java implementations which I will be discussing and comparing to the optimal Java implementations by the actual creators. Hopefully this MSc will be completed before a winner of the AES algorithms is chosen in late 2000, or early 2001.

5.7) Keys and Key Management

Keys are a highly contentious issue in the world of cryptology. It has been shown that weak keys or poor choices of keys can turn a secure system into a redundant system. The aim is to create and share secure keys, which do not put the primitives of the algorithm at fault for any security breach.

As described by [S.Halevi, H.Krawczyk] any key sharing technique that is required cannot be based on symmetric ciphers alone. This is the reason for using RSA to share the keys. As the current auctioning system has a number of requirements it should be shown what is done to protect and generate the best keys possible.

PRNG

The most common method for generating a key is to use a Pseudo random number generator (PRNG). It has been realised recently that a PRNG is actually a form of cryptographic primitive. In most cases the PRNG is the weakest point in the entire system and most cryptoanalysts attack this first [J.Kelsey *et al*]. There is no true random number generator on a computer, since as one knows, there is nothing random in a computer. New products are being released as add-on features to computers that will perform some random functionality. This includes measuring the heat of hard drives, spin speeds positions, using oscillating crystals, or pushing circuitry to its breaking point and outputting the results its spews out. The only thing that is truly random is nature. However this makes it rather difficult to monitor in a non-living machine.

Key attacks

The methods used for attacks on PRNG's and the keyspaces that they generate can be classified as follows: [J.Kelsey, *et al*]

1) Direct cryptanalytic attack
   This occurs when the cryptanalyst can see the results of the PRNG, and determine whether it's random or not. By deducing this one may reduce the keyspace and hence make it easier to guess the correct key.
2) Input based attack
   The cryptanalyst has knowledge of the inputs to the PRNG, or may control it thereby compromising the keys. This attack is divided into three parts: The known input attack means the cryptanalyst has confirmation of the information placed in the PRNG. The replay-input attack allows the cryptanalyst to force the same input state to be used. The chosen input attack allows the cryptanalyst to modify the input state.
3) State Compromise attack
   The cryptoanalysts manages to obtain some internal state values of the PRNG. This results in the cryptanalyst being able to guess and extrapolate what the future results are of the PRNG.

These are the kind of attacks that may be mounted on the PRNG. If one of them is successful the cryptographic primitives that the protocol is based on will be compromised, as they are only as strong as the keys used.

PRNG Guidelines

When working with PRNG it is best to take the advice of the industry. As A.E.Neuman once said "Learn from the mistakes of others; you'll never live long enough to make them all yourself". [J.Kelsey, *et al*] recommends some methodologies to follow to make the PRNG reasonably secure.

One should base the PRNG on something secure. This means that the cryptanalyst has to break the cryptographic primitive used in the PRNG. The internal state of the PRNG should change over time. This prevents the cryptanalyst from guessing the current state of the PRNG, which allows one to generate following information. It should resist chosen input attacks. If the cryptoanalysts guesses the correct internal state, but not the input, they should not be able to determine the result. The PRNG should recover quickly from compromises. Each change of input should result in a drastic change in the output.

Key Generation – The Server

Keys are generated in two places, the server and the client. The server generates the RSA keys to enable the key exchange. The client generates the session key for the symmetric algorithm.

The server uses the BigInteger class to generate a large prime number. This class first uses the random function of the Java class generate a large odd number. The

BigInteger class then uses the Solovay Strassen primality test to test the primality of the generated number. The Solovay Strassen test algorithm is as follows.

1) $n>1$ and is a random odd number , a is a positive integer < n.
2) If $\gcd(a, n) \neq 1$, then n is composite.
3) If $\text{Jacobi}(a, n) \equiv a^{(n-1)/2} \pmod{n}$ then n is composite
4) else n is prime
5) Probability = Probability / 2
6) Repeat from 2, using a different value of a until the probability that n is not prime is infinitely small

If n is composite it cannot be prime and the test fails. This process continues until the probability of n's primeness exceeds the minimum limit as specified by the user. Doing this allows one to be as certain as one wants of the primeness of the random number. The Miller-Rabin primality test may also be used and is a quicker algorithm. However the reason for using the system class function is that it will be faster than anything I could write. The default setting for the probability that the random number is prime is ninety-five percent. When dealing with such large numbers one can be reasonably confident that the number is prime. If there are any errors a new set of RSA keys will be generated and the problem will be resolved.

It has been shown that standard random number generators are not very good at generating random numbers. However the reason that I have used the standard Java random number generator is that one takes two of these prime random number and multiplies them together to get the base of the modulo arithmetic, which is still difficult to determine. This is the basis of RSA security, and in fact as long as the prime numbers are large enough and prime, the result of multiplying them together will make it just as hard to factorise and should not impair the system. It may slightly decrease the key space that the random numbers are being chosen from, which at 512 bits up to 4096 bits will be 2^{512} to 2^{4096} keys to search.

To prevent the random number from being too static the PRNG has been seeded with two seed values. These are then multiplied together to give a large initial seed. The seed chosen is the time the application takes to execute, multiplied with the current memory used by the system. Both of these will be different on all servers, depending on the following factors: the speed of the CPU, the current amount of applications running simultaneously, the load on the CPU, the amount of memory and the amount of memory currently used.

Key Generation – The Client

The client application needs to generate the symmetric key. The Java Sandbox prevents one from obtaining random values from the client machine, and therefore another method would be required to generate the key. It also needs a different type of key that does not need to be as long, but is required to be far more random. A few steps are followed to obtain a relatively random number from a PRNG. The client is asked to enter thirty characters into the keyboard, these must be as random as possible. The client need not remember this information, and all characters are allowed upper, lower, alphanumeric and special characters. This is then converted to a byte array (4 bytes/ Char, 30 Chars = 120 bytes). Then a number of random bytes are generated using the random number generator. The client's characters are used as a key to encrypt the randomly generated values using MD5 algorithm (See Appendix). MD5 is a hashing algorithm, and its output is used as the symmetric key between the client and the server.

MD5 is based on MD4 and has been around since 1991. The reason that I use this algorithm is that it can be construed as a PRNG. A single change in the key or the random input values will result in a huge change in the result. The RSAREF 2.0 is based on the MD5 hashing algorithm and addition modulo 2^{128}. MD5 has proved itself rather useful over the past decade in this function and there have been a number of attempted attacks mounted. It has been shown to be vulnerable against a chosen input attack that can force the algorithm into repeating information. This allows the cryptanalyst to work out its internal state at a certain time. This works in conjunction with a timing attack that monitors how the CPU works and can deduce what some of the internal state is. MD5 also has another problem. This occurs because MD5 results are order independent of the input. It does not matter if one sends A, then B into MD5, or B then A, one still gets the same result. [J.Kelsey, *et al*]

However there are ways of counteracting these attacks. One should prevent the cryptanalyst from choosing the input. This is achieved, as the client is the only one to actually select the characters to be inputted as the key for MD5. The cryptoanalyst must guess this (relatively hard), and guess the random number generated by the computer (relatively easy). As the client machines' information may not be easily deduced over the Internet, and not much information can be discovered about the CPU usage. The timing attack falls away. The order independence problem is resolved because only one set of values is ever generated and this makes it difficult for the cryptanalyst to deduce any statistical or frequency information.

The MD5 algorithm has proved itself over the last few years, and there are some problems. However due to its design and usage in the protocol it manages to avoid these weaknesses and hence generate keys that are random enough for usage with the RC6 and MARS algorithms.

Required Key Lengths

It has been shown that the security of an algorithm lies in the key, assuming that there are no major faults with the algorithm. A system will therefore be considered strong if the only attack that can be mounted on the system is a brute force method, whereby every key in the key space is tried until the correct one is found. Thus given a strong system it is imperative to have a large key space.
The auctioning system is dealing with sensitive data, but at most this will be staff cards, id numbers or credit card numbers. The flow of economic transactions could not be compared to that of a large corporation and hence it can be concluded that the type of attack on the system would not come from such a large organization. As has been mentioned before the benefits of the attack mounted on the auctioning system must outweigh the costs. Bear in mind that this was included in the design and development of the application, and therefore some issues that would be contentious (such as authorization and verification) in other protocols and systems are not so here.

Symmetric Key length

There are a number of ways in which this brute force method can be achieved. It can be implemented as distributed software that can be placed on all the machines on a university campus. These machines could run the application trying different keys when they are not in use. This type of attack would not cost any money, as the machines are free to use and usually idle. Some dedicated students could mount this attack over a weekend. At the other end of the scale trans-national corporations and governments could create specific hardware to perform the task of finding a key. It essentially boils down to two resources, time and money. They are indirectly proportional in mounting an attack on a system. The table below, from [B.Schneir, 1996] shows one the estimated cost of a brute force attack for symmetric keys of increasing length.

Cost / length	40 bits	56 bits	64 bits	128 bits
\$100 000	2 seconds	35 hours	1 year	10^{19}years
\$1000 000	.2 seconds	3.5 hours	37 days	10^{18}years
\$100 000 000	2 milli-seconds	2 minutes	4 days	10^{16}years
\$10 Trillion	0.02 micro-seconds	1 millisecond	0.3 second	10^{11}years

Fig 5.7.1 Average Time for brute force attack in 1995

As one can see the, the more money one has to spend, the easier it gets to mount a hardware brute force attack. These estimates were made in 1995, so they are now redundant, as no one can predictably measure the speed increase in hardware, with its associated drop in price for more than two to three years. This means it has become considerably cheaper to break larger key lengths over the past five years. Another paper by [M.Blaze, *et al*], which is about a year older than the figure above shows how quickly these values must be re-assessed.

Type of Attacker	Cost	40 bits	56 bits
Civilian	Nil to \$400	1 week	Not feasible
Small business	\$10 000	5 hours	38 years
Corporation	\$300 000	12 minutes	18 months
Big Company	\$10 000 000	24 seconds	19 days
Government	\$300 000 000	0.7 seconds	12 seconds

Fig 5.7.2 Revised average time brute force attacks

In 1997 a machine costing one hundred thousand U.S dollars was used to break the DES cryptosystem with a brute force attack. It took fifty hours, a year later, the price of the system had halved and it took twenty hours. DES was using a fifty-six-bit key. This is an indication how quickly technology changes.

The facts that concern the key length are simply this, what information is being secured, how long does it need to be secured for, and who would attack it. Bearing this in mind one can apply it to the current problem. The auctioning system has data that will only be valid for the lifetime of the current session and auction, which is measured in minutes and hours. Secondly the information is not extremely important so the kind of attacker that would be attracted would be a small corporation or less. The auctioning system uses 128-bit key encryption (32 byte) for the symmetric ciphers RC6 and MARS. The reasoning has already been provided, but to keep up with advances in technology these can be upped to 256-bit keys. This should maintain the security of the current session keys for a long enough period, against advances by the intended attackers.

Asymmetric Key lengths

As public key algorithms release their public keys, they are susceptible to a plaintext attack. For this reason the keys in general have to be longer than symmetric algorithm keys. In fact public keys need to be exponentially longer to attain the same level of security. Most of these keys are also based on the fact that factoring large numbers and finding their prime decomposition is rather difficult at the present time. So one does not need to find the key, one is required to discover the primes that the key has been created with.

Symmetric Key Length	Asymmetric Key Length
56 bits	384 bits
64 bits	512 bits
80 bits	768 bits
112 bits	1792 bits
128 bits	2304 bits

Fig 5.7.3 Key Length comparisons for equivalent security [B.Schneier, 1996]

This had lead to the rise in special factoring algorithms: the quadratic sieve, and the number field sieve are but a few. Currently the research in this field is exploding and this has resulted in large numbers being factored at quicker speeds. Currently 130-digit to 140-digit numbers are being factored, which means that RSA 512 bit keys are at risk. Admittedly it takes a lot of computation and resources, but it has been shown that these keys are breakable. Therefore one should not be complacent with current key lengths. The level of security should be determined by the information being encrypted, and how long does it need to be secured for. As the key will be used to distribute session keys it has to last the length of the session and auction. The default is a 128 bit key, however one has the option to increase up to and including 4096 bit keys, which does take quite awhile, but will guarantee security for at least the next five years.

[B.Schneier, 1996] has recommended that in the year 2000, one should have the following public key lengths, 1024, 1280 and 1536. These are against individual, corporation and government attacks respectively. This would have increased dramatically to 1280,1536,2048 against individual, corporation and government attacks respectively by the year 2005. These are just estimates but they do show the huge strides that are expected in the respective fields of technology and cryptanalyst.

In the light of all this one should not forego the unknown. We may be on the verge of a monumentous discovery involving the factoring of numbers that may yield public key algorithms defunct.

5.8) Protocol Attacks

There are a number of techniques that could be employed to attack the current protocol. They are as follows:

Eavesdropping

Eve or the undesirable listens to the communications channel and tries to deduce some information about the information being sent. This can be performed in a number of ways. A network sniffer is a utility that catches all packets along a network line even if they are not destined to the current machine. Another method, which is rather simple, is a technique by which an electronic device is placed around a section of the network cable. This then makes a copy of the packets being sent and diverts them to the attacker.

Replay

The attacker has some record of the messages sent along the communication channel. The attacker attempts to place these copied messages in the communication channel at a later stage and determine what occurs when the message is received.

Man in the middle

The attacker posses the power to divert the communications channel and add or replace messages in this channel. The attacker usually replaces the current message with his own message. The attacker sets up a block between the client and the server. To the client the attacker acts as the server, and vice versa for the server.

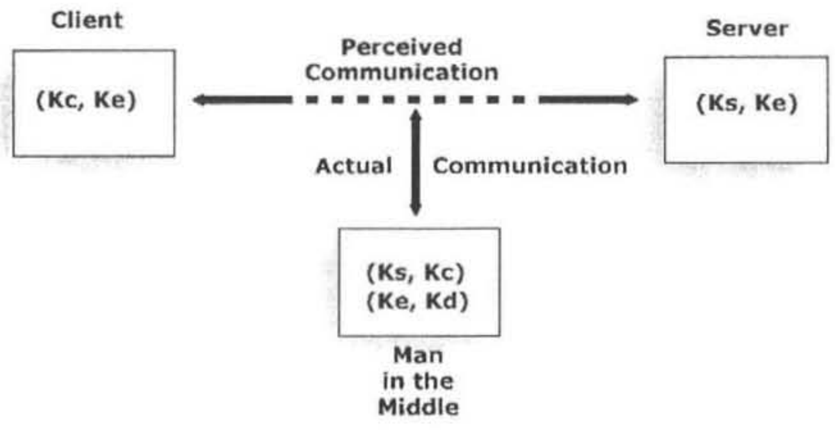


Fig 5.8.1 Man in the Middle attack

The server sends a message to the client (K_s); the man in the middle (Eve) intercepts this. This is replaced by (K_e), and forwarded to the client. The client receives (K_e), and uses it to encrypt (K_c), as it is assumed to be the servers key. The client sends this message back to the server. However this message is also intercepted by Eve, whom decrypts this with the (K_d) key. Eve has now got a copy of (K_s) and (K_c). Eve encrypts (K_c) with the server's real key (K_s) and forwards it onto the server. The client and the server don't realise there is a problem. The man in the middle continues this process as he may decrypt and encrypt all messages between the client to the server.

Miscellaneous

The attacker may use other techniques to get keys to the algorithms from the client. For instance there are devices that can be pointed at computers from a distance away and can reproduce the state of the monitor or internal information. Another method would be to bribe or threaten a user for keys.

5.9) Safeguards

Now that one has a brief understanding of the attacks that may be mounted on the protocol, we will discuss some measures that may be taken to avert such attacks.

Authentication

Authentication schemes provide the user's of protocols with authentication and confidentiality of the information sent and received. This results in the non-repudiation property of the cryptosystem being met. Authentication is usually achieved by use of digital signatures and certificates, implemented through public key cryptosystems.

In the current problem there are only two parties the server and the client. If one were to use digital certificates one would have to introduce a third trusted party known as the CA (Certificate Authority). This means that the problem space would now have grown. The server would have a certificate that supports its claim to be who they are, this has been distributed to the server from the CA. The client then requests the CA certifies the certificate, acknowledging the server's claim. All this extra work is not a necessity in the current problem and would place an extra burden on network and computational resources.

One should understand that authentication and key exchange are two separate problems that most people seem to lump together. Authentication is about convincing the client that they are communicating with the intended server. Key exchange is the sharing of a single key between the two parties. As [M.Bellare, Progaway, 1995] stated, "Most of the time the entity authentication is irrelevant...". What is really a concern is making sure that one can securely share a key between two parties. Authentication will help in this matter, but it will not achieve this goal. It will go a long way to resolving the man in the middle attack.

Averting Replay

Replay attacks occurs when the attacker copies some messages from the client or server and then uses them at a later date. The messages will fall into two categories, those to the client, and those to the server. If the messages are to the client or server it does not really matter as the session key only lasts for the duration of the auction. Thereafter it is changed and the message will be ignored, as it cannot be decrypted. An alternate client cannot use the current client's message, as they will have a different session key and hence the message will not be decrypted. This prevents the hijacking of sessions from currently joined clients. The message can only be a bid message and if it is replayed it will be exactly the same information as contained in the last bid, in which case it will have no affect on the auction.

To prevent replay attacks it is suggested that one uses time stamping or a counter in the message so that the host can differentiate from the previous messages. There are a number of problems with time stamping. The first is that it compromises the system as the attacker will now know a partial extract from the message as it is not too difficult to determine the time on a machine. This will make it easier to mount an attack on the encryption protocol. Secondly, even if the counter or time stamp is hashed it is going to increase the size of the message, thereby using larger bandwidth for communication. The aim is to use as little resources as possible, and to do this the messages are as small as possible. To avoid having to use time stamps or counters the auctioning system communication was developed so that a client may place a bid, which is time stamped once it is received. All messages thereafter, even if they are a repeat, will not be valid unless the bid is higher than the previous bid. The use of threads and unique session keys has averted the risk of replay attacks.

Replay and authentication are both issues that need to be resolved when dealing with protocol development. However for the current problem I believe that these matters are not an issue.

5.10) Hardware vs. Software

The implementation of this problem has been software approach, whereby the algorithms have been manually coded and executed. These days with the huge increase in digital communication many organizations find that the software implementation is not efficient enough. This has resulted in algorithms being implemented in hardware. These come as chips or add on cards that allow one to plug them straight into the system. Each of the algorithms that will be discussed in the following chapters gives an indication of the software implementation speed at which they operate.

It is obvious that anything specifically created in a hardware format is going to out perform the software implementation. There are many companies moving into this territory and I assume that it won't be long before hardware encryption takes over from software.

As one can see from the table most hardware implementations will be designed for specific hardware platforms or operating systems. However they are incredible quick, and the cost to the user of encryption will be less than a software implementation which will slow the computer. This will result in more memory and CPU's for the server involved. The disadvantage of hardware implementations is that the are not easily upgradeable, as opposed to the software implementation which can be changed and simply recompiled.

Hardware Encryption	Software Encryption
Fast	Slow
CPU independent	CPU intensive
Cost effective (Cheap)	Expensive
System dependant	Portable
Fixed	Modifiable

Fig 5.10.1 Advantages / Disadvantage of hardware and software encryption

The PIJNENBURG Company deals in these kind of add on cards, as example it is interesting to see the specifications and feature attained by such a card.

PCC-ISES features:
ARM7TDMI® Microprocessor with 128KB RAM
Embedded Cryptographic Accelerators
RSA up to 4096 bits, DES, 3DES and SAFER, MD5, SHA-1 and RIPEMD
True Random Number Generator, Tamper Security Circuit

Function	Speed
RSA - 1024 bit	300 / sec
DES (ECB)	400 Mbps
Triple Des (ECB)	400 Mbps
MD5	375 Mbps
SHA-1	300 Mbps
Random Number	50 Kbps

Fig 5.10.2 Example of speeds of encryption in a hardware implementation.
[http://www.pijnenburg.nl/, online]

As one can see, the performance of this card is exceptionally fast in all cryptographic aspects. The card price is estimated at approximately four hundred US dollars. This is far cheaper than the cost of a new computer, which would deal exclusively with encryption features of a server.

These types of add-on cards will become more prevalent in the future and will invariably become programmable. This will enable any bugs or problems to be fixed. This will increase the ease of use of encryption and make it a common place feature for all network communication in the future.

5.11) Conclusion

This chapter has discussed the types of problems and issues that face the designer of a security protocol. The requirements of the current protocol were discussed and the resultant solution investigated. This included the types of algorithms used, what they required and how they interlocked to create the finished cryptographic protocol.

The major weakness of the protocol is its vulnerability to the "man in the middle" attack. However as this type of attack is very hard to mount and the rewards are small it was decided not to persue an expensive third party authentication method, which is currently the only known way to defend against such attacks.

In the sections that follow I will discuss in detail the RSA algorithm, its implementation and its problems

6) RSA

W.Diffie and M.Hellman first brought public key cryptography to light in the paper "New Directions in Cryptography". In nineteen seventy-six they patented (U.S Patent #4,200,770) the idea of public key cryptography. In their system all users have a public and a secret key. The public key is made available to all, and the secret key remains private. Anyone may encrypt a message using the public key, but only the secret key of the recipient of the message, will perform decryption.

The RSA cryptosystem is based on this theory and was publicised in 1977 by R.Rivest, A.Shamir and L.Adelman. It was patented in 1983 (U. S. Patent #4,405,829), and it is patented in a number of countries. The security of RSA is based squarely on the difficulty of factorising large numbers. "In Scientific American, Martin Gardner described the RSA scheme as "A New Kind of Cipher That Would Take Millions of Years to Break." Oddly enough, just 60 years earlier in 1917, the same journal published an article touting the Vigenere ciphers as "impossible of translation". [D.Denning, 1996]

An interesting aside is something brought to light in "The Code Book" by S.Singh. It has been shown that RSA and public key cryptosystems were actually invented in the late sixties by the British Government Code and Cypher School (GCCS). Three gentlemen by the names of M.Williamson, C.Cocks and J.Ellis had discovered this technology and due to the secrecy act, were forbidden to talk about it. It was also for this very reason that their ideas were not patented, as this would mean the cryptosystem would fall into enemy hands. In 1997 this department revealed the truth at a press release.

This tragedy is best described by one of the men involved. As related by J.Ellis in a secret document, "Cryptography is a most unusual science. Most professional scientists aim to be the first to publish their work, because it is through dissemination that the work realises its value. In contrast, the fullest value of cryptography is realised by minimising the information available to potential adversaries" [S.Singh,1999]

6.1) Overview

The RSA cryptosystem has been in existence for over twenty years and has withstood large amounts of cryptanalysts. Admittedly it will not always be the de facto standard in asymmetric cryptosystems, but currently there is no better solution to the key exchange problem. RSA is part of the following international standards; International Standards Organziation (ISO), the Consultative Committee in International Telegraphy and Telephony (CCITT) X.509 security standard, the Society for Wolrdwide Interbank Financial Telecommunications (SWIFT) and the ANSI X9.31 standard for the U.S banking industry [RSA FAQ].

This is the explicit reason that I used the RSA cryptosystem. It has been the dominant force over the last twenty years and its impact on cryptology will be felt for years to come. The interesting thing about this algorithm, is the fact that it has never been proven absolutely secure. It has been based on a difficult problem that is commonly known as a one way function. A one way function is very easy to evaluate. However it is difficult to find the inverse of the function. A trap door one way function uses the same principle but with a back door. It is easy to calculate $F(x)$, and difficult to find $F^{-1}(y)$, unless one has some extra information which makes the inverse possible. Almost all public-key cryptosystems are based on this premise of assumed existence of one way, trap door functions.

The major flaw in the thinking is that is has not been proven mathematically that these functions exist. If a method is found to calculate $F^{-1}(y)$ easily the cryptosystem would be rendered useless. However thus far no major breakthroughs have been discovered for inverting RSA's supposed one way trap door function. The key here is that it is "almost impossible", but not purely impossible. This fact will be revealed later in the cryptanalytic attacks that are mounted on the RSA cryptosystem.

Even when RSA lies a broken cryptosystem, the way in which it has been used, and the amount of effort that has been utilized in the examining of the system still makes it worthy of studying. This is the prime reason I included the most popular and simple asymmetric cryptosystem currently in use on a global scale.

6.2) Theory

The RSA cryptosystem is based on a number of mathematical premises, which need to be examined before one can discuss the cryptosystem.

GCD

An integer a is known as common divisor if it divides two integers x and y, i.e. where

$a|x$ and $a|y$

The Greatest Common Divisor (GCD) is expressed as the following.

$a = \gcd(x, y)$

where a is the greatest common divisor of the two integers x and y. For instance,

$x = 20$, the divisors of x are : 1,2,4,5,10.
$y = 28$, the divisors of y are : 1,2,4,7,14.
The greatest common divisor, gcd(20,28) = 4.

Note that when another value z divides x and y, but is not the greatest common divisor, then z divides the greatest common divisor. In the current example

$z = 1$ or 2

The GCD is relatively simple to determine when the numbers are small, however when the numbers being dealt with become large it becomes difficult to determine the GCD. The best solution to this problem can be found in the Euclidean algorithm.

Euclidean Algorithm

The Greek mathematician Euclid published the Euclidean algorithm in approximately 300B.C.

The algorithm is a repetitive loop of division until the end result is attained. Its formal definition for (x, y) is as follows:

$$
\begin{array}{rll}
x &= yq_1 + r_1 & 0 < r_1 < y,\ y < x \\
y &= q_2r_1 + r_2 & 0 < r_2 < r_1 \\
r_1 &= q_3r_2 + r_3 & 0 < r_3 < r_2 \\
& \text{" "} & \\
& \text{" "} & \\
r_{m-2} &= r_{m-1}q_m + r_m & 0 < r_m < r_{m-1} \\
r_{m-1} &= r_mq_{m+1} + 0 &
\end{array}
$$

This repetitive division yields the greatest common divisor of x and y as the value r_m. The initial value of q_1 multiplied by y is the greatest whole number less than x, and r_1 is the remainder.

An example to find the gcd(8246,2326) :

$$
\begin{array}{rll}
8246 &= 2(2326) + 1268 & 0 < 1268 < 2326 \\
2326 &= 1(1268) + 1058 & 0 < 1058 < 1268 \\
1268 &= 1(1058) + 210 & 0 < 210 < 1058 \\
1058 &= 5(210) + 8 & 0 < 8 < 210 \\
210 &= 26.8 + 2 & 0 < 2 < 8 \\
8 &= 4.2 + 0 &
\end{array}
$$

$\therefore$ gcd(8246,2326) = 2

Note: The running time of this algorithm is determined by $O((\log n)^2)$

Now that one has this method for determining the greatest common divisor, what can be achieved with it. Before one may continue it is necessary to describe a field.

Definition: A field is a commutative ring in which all non-zero elements have multiplicative inverse [D.Stintston].

Z_5 is a field, in which all operations on the field are modulo 5.

Ve^-	-5	-4	-3	-2	-1
Z_5	0	1	2	3	4
Ve^+	5	6	7	8	9

Fig 5.2 Z_5 field

One can see from the table above what values outside the field Z_5 equate to inside the field. I.e $-4 \bmod (5) = 1$. Returning to the definition, which states that inverses must exist in the field. The extended Euclidean algorithm determines these inverses in the current field.

Extended Euclidean Algorithm

The extended Euclidean algorithm is based on the following theorems:

If $x \in Z_n$,there exists a $y \in Z_n$ which is a multiplicative inverse of x. i.e. $x.\ y = 1$ in Z_n.

It follows that if gcd(x,n) = 1 (No other elements in the ring are a divisor for x), then the inverse of x in Z_n is unique.

The extended Euclidean algorithm is as follows:

Calculate t_0 to t_m as follows:

Calculate the Euclidean and Extended at the same time.

$r_i = r_{i-2} - q_{i-1} . r_{i-1}$
$\equiv t_{i-2} . r_1 - q_{i-1} . t_{i-1}\ r_i \pmod{r_0}$
$\equiv (t_{i-2} - q_{i-1} . t_{i-1})\ r_1 \pmod{r_0}$
$\equiv t_I . r_1 \pmod{r_0}$

Calculate the Euclidean algorithm for (x, y), substituting the values of q_i determined above into the following formulae. Calculate t_0 to t_m as follows:

$t_0 = 0,\ t_1 = 1,$
$t_2 = (t_0 - q_1 . t_1) \pmod{n}$
$t_3 = (t_1 - q_2 . t_2) \pmod{n}$
" " "
$t_m = (t_{m-2} - q_{m-1} . t_{m-1}) \pmod{n}$

If $r_m \neq 1$, then there is no inverse. If this is not the case then the inverse is

$t_m \pmod{n} = a^{-1}$ in Z_n

An example (47563,31387), calculate Euclidean first to see if an inverse exists,

Euclidean	Extended Euclidean
47563 = 1.31387 + 16176	$t_2=(t_0- q_1 . t_1) \pmod{n}$ = (0-1.1) (mod 47563) = 47562
31387 = 1.16176 + 15211	t_3 = (1 – 1.47562) (mod 47563) = 2
16176 = 1.15211 + 965	t_4 = (47562 – 1.2) (mod 47563) = 47560
15211 = 15.965 + 736	t_5 = (2– 15. 47560) (mod 47563) = 47
965 = 1.736+ 229	t_6 = (47560 – 1.47) (mod 47563) = 47513
736 = 3.229 + 49	T_7 = (47- 1.47513) (mod 47563) = 197
229 = 4.49 + 33	t_8 = (47513-4.197) (mod 47563) = 46725
49 = 1.33 + 16	t_9 = (197-1.46725) (mod 47563) = 1035
33 = 2.16 + 1	t_{10} = (46725-2.1035) (mod 47563) = 44655
16 = 16.1 + 0	

(47563,31387) = 1, therefore an inverse exists, so continuing with the extended Euclidean, the inverse is t_{10} = 44655

Therefore 31387^{-1} in $\mathbb{Z}_{47563} = 44655$. This algorithm is one of the core building blocks of RSA. However there is still a little more theory required to cement the process.

Primes

A Prime numbers has the special property, which is that, it may only be divided by itself or the number one. Relatively prime numbers are two integers x,y such that gcd(x,y) = 1. They are prime to one another, this does not mean they are prime.

The Euler function, $\varnothing(n)$, denotes the number of integers in the interval 1..n, which are relatively prime to n. If p is prime then $\varnothing(p) = p-1$. The Euler function is also multiplicative, so that $\varnothing(mn) = \varnothing(m)\,\varnothing(n)$.

The Chinese Remainder Theorem

The Chinese remainder theorem states that if one has a number of relatively prime positive integers ($p_1..p_k$), which are the prime factorization of n ($n= p_1\,p_2..p_k$), one can use this to solve the set of equations,

$a_1 \equiv x \pmod{p_1}$
" "
$a_k \equiv x \pmod{p_k}$

Which have a unique solution x, with $x < n$. So, for an arbitrary $a < p$ and $b < q$ (where p and q are prime), there exists a unique x, where x is less than pq, such that $x \equiv a \pmod p$, and $x \equiv b \pmod q$.

The Euler Fermat Theorem

If some integer $x \in Z^*_n$, where ($Z^*_n = \{x \in Z_n, x^{-1}$ exists in $Z_n\}$) , then

$x^{\varnothing(n)} = 1$ in Z_n, and for every gcd(x,n)=1, $x^{\varnothing(n)} \equiv 1 \pmod n$

Fermat's Theorem

If p is a prime number and x a positive integer, then

gcd(x,p) = 1, and $x^{p-1} \equiv 1 \pmod p$, and $x^p \equiv x \pmod p$.

Its also interesting to note that if p is prime then Z^*_p is a group of order p-1.

This theory is what the RSA algorithm is based, and all will be much clearer once it has been discussed.

6.3) RSA Algorithm

The following steps are used to implement the RSA algorithm
1.) Alice generates two large random prime numbers p and q.
2.) $n = pq$
3.) $\varnothing(n) = \varnothing(pq) = (p-1)(\ q-1)$
4.) Alice then picks a random integer a such that $\gcd(a, \varnothing(n)) = 1$

The public key is then (a,n). This is used to encrypt information using the following encryption algorithm

$E_{(a,n)}(m) = m^a \bmod n$

This raises the message value to the power of a in the field Z_n. The Chinese remainder theorem guarantess a unique solution. This means that only the current message will result in the corresponding ciphertext message.

5.) Alice then determines the inverse of ($a^{-1} = b$) in $Z_{\varnothing(n)}$.

The secret key is then (b,n). This is used to decrypt in the following fashion:

$D_{(b,n)}(c) = c^b \bmod n$

The decryption algorithm raises the ciphertext to the inverse of the key, thereby reverting it to its plaintext status.

The entire security of RSA lies one thing, which is the factorising of n to find the two primes. Without these primes one is unable to find the Euler function, $\varnothing(n)$. The Euler function is the field in which Alice discovers the inverse of a. This inverse, b is the key to decrypting the encrypted messages. Please note that $||m^a||_n$, denotes m^a mod n.

Theorem:
$D_{(b,n)}(E_{(a,n)}(m)) = m$

Proof:

$$\begin{aligned} D_{(b,n)}(E_{(a,n)}(m)) &\equiv ((m^a) \pmod n)^b \pmod n \\ &\equiv (m)^{ba} \pmod n \end{aligned}$$

Now $b.a = 1(\bmod \varnothing(n))$, as $b=a^{-1}$,where $\varnothing(n)=(p-1)(q-1)$
$= 1 + t(p-1)(q-1)$, where integer t>1.

$$\begin{aligned} &\equiv (m)^{1+t(p-1)(q-1)} \pmod n \\ &\equiv m\ m^{t(p-1)(q-1)} \pmod n \\ &\equiv m\ (m^{t\varnothing(n)}) \pmod n \end{aligned}$$

As m exists in Z_n, by Fermat $m^{\varnothing(n)} \equiv 1$

$$\begin{aligned} &\equiv m.(1^t) \pmod n \\ &\equiv m \pmod n \end{aligned}$$

$\therefore D_{(b,n)}(E_{(a,n)}(m)) = m$

It would probably easiest to show the working of RSA in an example.

1.) Generate two primes p and q (211,251)
2.) $n = p.q = 211.251 = 52961$
3.) $\varnothing(n) = (p-1)(q-1) = (210)(250) = 52500$
4.) Choose a, such that $1 < a < 52500$, $a = 139$

$K_p = (139,52691)$ is the public key. Now one has to determine the private key.

5.) Calculate the inverse of, using the extended Euclidean algorithm.
    Use $Z_{\varnothing(n)}$ as the field, in other words determine the inverse of (52500,136).
    $a^{-1} = b = 36259$.

The secret key is then $K_s = (36259,52691)$.

The public key is given to Bob and anyone else who wants to communicate with Alice. Bob receives the key and decides to send Alice a message "Hello World!".

Now that this has been determined let us perform the encryption of the string "Hello World!". One needs to convert these characters to numerical characters. Where, a=00,b=01,..z=25,space=26,!=27. For the sake of the example we will ignore the case of the letters.

H	E	L	L	O		W	O	R	L	D	!
07	04	11	11	14	26	22	14	17	11	03	27

To make things easier to work with, combine the letters into group of two. The numbers however must be smaller than the values of n (n= 52691), to prevent any problems. This results in the following sequence of numbers that may be encrypted separately.

0704,1111,1426,2214,1711,0327

$E_{(139,52691)}(0704) =$ $(0704)^{139}$ (mod 52691)= 13400
$E_{(139,52691)}(1111) =$ $(1111)^{139}$ (mod 52691)= 14823
" " "
$E_{(139,52691)}(1711) =$ $(1711)^{139}$ (mod 52691)= 24843
$E_{(139,52691)}(0327) =$ $(0327)^{139}$ (mod 52691)= 40232

The resultant encrypted message is:

13400, 14823, 38303, 43817, 248483, 40232

This is the message sent from Bob to Alice. On receiving it Alice decides to decrypt the message with the secret key K_s.

$D_{(36259,52691)}(13400) =$ $(13400)^{36259}$(mod 52691)= 704
$D_{(36259,52691)}(14823) =$ $(14823)^{36259}$(mod 52691)= 1111

and so on until the message received is

0704,1111,1426,2214,1711,0327

This is encoded back into a string, and one gets

07	04	11	11	14	26	22	14	17	11	03	27
H	E	L	L	O		W	O	R	L	D	!

which is the original message sent by Bob to Alice.

6.4) RSA Implementation

The RSA cryptosystem that has been implemented in the auctioning application is in the RSACrypt.Java file. As the number dealt with can get very large I have decided to use the new Java implementation of the BigInteger Class, as opposed to coding up my own class. The BigInteger class has been reworked in Java two and is incredibly fast and efficient. It would be unproductive not to use the native support for large numbers.

The BigInteger class will represent as large a number as one requires and allows all the required mathematical functions to be performed. "Immutable arbitrary-precision integers. All operations behave as if BigIntegers were represented in two's-complement notation (like Java's primitive integer types). BigInteger provides analogues to all of Java's primitive integer operators, and all relevant methods from java.lang.Math. Additionally, BigInteger provides operations for modular arithmetic, GCD calculation, primality testing, prime generation, bit manipulation, and a few other miscellaneous operations." [Java 1.2 Help files].
The class has a number of functions and methods. The class stores the clients information, this includes, p, q, n, ∅(n) and the public and secret keys. They are all of type BigInteger.

When the class is intialized for the first time it calls the method to generate keys

```
//RSACrypt.Java - generate_Keys(size,error,seed) method
p = generate_prime(size,error,seed)
q = generate_prime(size,error,seed)
n = get_N(p,q)
phi = get_phi(p,q)
publickey = get_A(phi)
privatekey= get_Inverse(publickey,phi)
```

The p and q values are generated, once this has been done the value of n is then determined by calling the *get_N(p,q)* method. The value of phi is determined by calling the *get_Phi(p,q)* method. The public key is determined by calling the *get_A(phi)* method, and the private key is determined by calling the *get_Inverse(publickey,phi)* method.

Generate_prime(size,error,seed)

This method generates the random large prime numbers for RSA.

```
Random Number (seed)
if (Error > 99) then Err = 99
else if (Err <1) then Err = 1
return Random BigInteger(Size,Err)
```

The method makes sure the error value is not out of range, and then generates a random prime number. The error value determines how sure one is that the random number is prime using the Solovay Strassen test.

It is interesting to note that the number of primes five hundred and twelve bits long or less is equal to 10^{150}, which is more than the number of atoms in the known universe [RSA FAQ].

Get_N(p,q)

This method multiplies the two BigInteger values and returns the result. As has been mentioned before the BigInteger values have explicit functions for mathematical functions.

```
Return p.multiply(q)
```

Get_Phi_N(p,q)

This method subtracts one from p and q respectively and multiplies them together to get the euler function value known as phi. This is the value that is returned.

```
p.subtract(one)
q.subtract(one)
Return p.multiply(q)
```

Get_A(phi)

This method generates a random public key value between 1 and the phi value. This value is then checked to make sure that the greatest common divisor of the value and phi is one. The key generated is also a prime number, which should mean that the gcd test succeeds.

```
Do {
        Public key = generate_prime(sizeOf(phi)-1,error)
}while Not (gcd(public key,phi)=1)
```

Get_B(phi, a)

This method returns the value b, given a and phi. The value of b is simply a^{-1}.
Return get_Inverse(phi,a)

Get_Inverse(publickey, phi)

The inverse function is inverts a value in the phi field. It calls the function *get_Power(x,a,n)*. This function is also used by the encrypt and decrypt methods. The public key value is simply raised to the power of negative one, thereby determining the inverse. This is then returned.

```
Return get_Power(public key, BigInteger(-1), phi)
```

Get_Power(x,a,n)

This uses the functionality of the BigInteger class to perform all the hard work. It uses the *mod-pow* function of the BigInteger class. This will raise a value to a certain power and then mod it by a mod value. In doing so it meets the requirements of the RSA algorithm, as all operations have to be modulus the n value.

Retrun x.modpow(a,n)
X^a (mod n) is the result.

RSAEncrypt(plaintext,public Key,n)

At the heart of the encryption is this method. Which accepts a string called plaintext, this is then converted into a BigInteger. This plaintext value is then raised to the power of the public key modulus n. The result is returned as a string value

Return get_Power(BigInteger(plaintext),public key,n)

RSADecrypt(ciphertext,secret Key,n)

The decryption method works in exactly the same way, but to make things clear, it has been coded up with a different name.

Return get_Power(BigInteger(ciphertext),secret key,n)

There are a number of methods for storing and retrieving the public key, private key and n values as strings and integers.

6.5) RSACrypt Usage

Each of the client applets and the server creates an instance of the RSACrypt class as it is central to the encryption process. This is done in the server's main thread in the *serverlistener* method. A copy of this object is passed to the auctionhandler and each of the clienthandler threads. On initialization the keys are generated for use between all the clients. This is only changed at the end of an auction or on the command of the server application administrator. When a client connects to the server, the server sends a message with the public key (K_p,N) to the client (See Chapter 4.13).

The client creates an instance of the RSACrypt class. On receiving the message containing the public key, it extracts these key values and places them in the RSACrypt class variables, through the get/set methods. The client then generates the symmetric key. This key is of type byte, and is made up of thirty-two bytes by default. The key is divided into pieces of four bytes each, these four byte values are then converted to single integer values. (i.e 32 bytes = 8 integers) . The client then individually encrypts each integer using RSAEncrypt and the public keys. Once all the integers have been encrypted they are sent back to the server. The CryptLib.Java library performs the converting of Integer to bytes and vice versa, as well as a number of other useful cryptographic functions.

```
//ClientEngine.Java – setEncryption() Method
For (loop=0;loop<sizeOf(Symmetric Key)/4;loop++) {
        Integer = ConvertToInteger(SymmetricKey[loop*4])
        Message += RSAEncrypt(Integer, Public Key, N)
        }
```

The server receives the encrypted integer values. Each integer value is decrypted using the private key, until all the integers have been dealt with. The integer values are then converted back to a byte format. All the bytes make up the resulting symmetric key, which is used for the symmetric algorithm. Only once the symmetric key has been received, does the usage of RSA stop and the symmetric cryptosystem takes over. If the server recreates RSA session keys they are distributed to all new clients as the start, but not clients that are already communicating using the symmetric cryptosystem.

6.6) Timing

The timing of the algorithm is interesting to look at because it shows one how fast the encryption and decryption is and the kind of pressure it is placing on the CPU. It is also useful for one to compare against hardware implementations of the RSA encryption which show how much faster it could be (Chapter 5.10, Fig 5.10.2)

	32 bits	128bits	256 bits	512 bits
Encrypt	220ms	5820ms	32370ms	217610ms
KBits/Sec	142.04	21.47	7.63	2.29
Decrypt	220ms	5830ms	32790	214050
KBits/Sec	142.04	21.48	7.61	2.33

Fig 6.6.1 Times for 1000 encryption and decryption's of RSA.
Using equal sized bits for P and Q and large A.

As one may see from above table the encryption and decryption figure are similar for the different keys. All get exponentially slower as the size of the key increases. One can see however how slow in general the actual operations are. As related by [B.Schneier,1996], RSA 512 bit is attaining up to sixty four kilobits a second in a hardware implementation, which is significantly faster than the software implementation.

	32 bits	128 bits	256 bits	512 bits
Encrypt	60ms	220ms	490ms	1980ms
KBits/Sec	520.80	568.17	510.19	252.51
Decrypt	280ms	6310ms	32680	218380ms
KBits/Sec	111.60	19.81	7.64	2.28

Fig 6.6.2 Times for 1000 encryption and decryption's of RSA.
Using equal sized bits for P and Q, but a very small e.

The timing of the RSA encryption and decryption functions was performed on a Pentium II 450MHZ CPU with 196MB ram. Each encryption and decryption is performed one thousand times to get a good average.

Optimization

The classes used for the operation are all system classes and therefore are as quick as possible. The only other speed-ups will occur from smaller numbers and hence quicker operations. On average public key encryption is of the order $O(n^2)$, private key decryption takes time $O(n^3)$, and the key generation takes the longest with an order of magnitude of $O(n^4)$ [RSA FAQ]

One method suggested by Shamir [H.Gilbert, *et* al] is by fiddling with the public key exponent. One chooses equal size p and q values, but the value of a (x^a(mod n)) is chosen to be small, and it meets gcd(a,(p-1)(q-1)=1. In some cases this a has even been chosen as small as three. This will mean that the encryption of the message is incredibly quick. However this does not decrease the speed of decryption as the inverse (secret key) will invariably be a large number.
Another method is the use of different sized p and q's. Whereby the value of p is half the size of q. However this method is susceptible to attack as discussed by [M.Joyle & J.J.Quisquater, 1997] and will not be discussed.

6.7) Attacks on RSA

RSA has been checked, analysed, pulled apart and attacked in many different ways, but none of these attacks have actually crippled the cryptosystem. There are one or two main lines of attack, but most seem to attack the improper implementation of the cryptosystem.

There have been many attacks over the past twenty years and all are not relevant to this discussion. Only a few of the main attacks will be discussed here.

Factorization

The non-factorization of n (n=pq) into its component values of p and q is the basis of the security of the algorithm, and also its main point of attack. One must realise though that the difficulty of this factoring problem is actually a conjecture, and furthermore it has never been proven that one needs to factor n to find the decryption exponent.

One asks how hard is it to discover the factors of the modulus n. Once these have been discovered, $\varnothing(n)$ is easily determined and $b = a^{-1} \pmod{\varnothing(n)}$ can be found. Determining these factors is commonly known as a brute force attack as one tries a huge number of solutions until the correct one is found. The factoring of large numbers is related by [D.Boneh] as "...one of the most beautiful problems of computatonal mathematics."

The factoring of large numbers has been getting better every year due to a combination of things. The technology used to implement the factoring of numbers is getting quicker, as well as cheaper, and the algorithms used are becoming more efficient and powerful. The Quadratic sieve the fastest-known algorithm for numbers less than 110 digits. The fastest algorithm for digits above this is the Number field sieve (NFS). As an indication, the General number field sieve, which has factored 132 digit numbers (Approximately RSA 512 bit), has a running time of exponential $((c + O(1))n^{1/3}\log^{2/3}n)$, where c<2, on n bit integers. [A.K.Lenstra, et al], relates a discussion of the Number field sieve. However the interesting thing to note here is that the NFS is generally considered ten times faster than the QS for larger numbers.

"In March 1994, a 129-digit (428-bit) number was factored using the double large prime variation of the multiple polynomial QS [66] by a team of mathematicians led by Lenstra. Volunteers on the Internet carried out the computation: 600 people and 1600 machines over the course of eight months, probably the largest ad hoc multi-processor ever assembled. The calculation was the equivalent of 4000 to 6000 mips-years" [B.Schneier]. This means that for anyone willing to spend a few million dollars it is within the realm of possibility to break a 512-bit RSA encryption. Each year the price of the hardware drops, the speed increases and the size of the numbers to be factored increases.

For the purpose of the current protocol it would take a rather long time to factorise the required values. The length of the auction is measured in hours and minutes, and hence so is the RSA session key. The time taken would require days and months to determine, which means that by the time the decryption exponent was discovered it would be out of date for the current auction.

Revealing decryption exponent

This may seem trivial at first, but it is relevant to show how the revealing of the secret key would result in the cryptosystem failing. This is described in [D.Boneh, Pg3], but if one does have the decryption exponent b, it seems pointless determining and factoring n, as one will be able to decrypt all messages.

However if the encryption and decryption exponents (public and private keys) change often and the value of n is constant it is shown that by using the Chinese remainder theorem and by guessing one can find the factors (p and q). However this will take the order of $O(n^3)$. The attack on a common modulus is described in [D.BleichrnBacher, *et al*][D. Boneh] and [M.Joyle & J.J.Quisquater, 1997].

The only way that one could get the decryption exponent from the auction server would be to have some control over the system it is running on. As is indicated in most security specifications, if an intruder has access to the machine the system is executing on no security may be guaranteed. In other words, all bets are off. The entire protocol is designed at maintaining secure communication between the client and the server. The server is assumed to be a relatively secure environment. If the server is unfaithful and reveals this information then nothing may be done.

Discovering ∅(n)

It has also been suggested that one guesses the values of (p-1) and (p-1), one could then use the extended Euclidean algorithm to find the inverse of the encryption exponent. To find these values though would be just as difficult as factoring n. There are two equations available to finding the factors of n.

$n = p.q \quad (1)$
$\varnothing(n) = (p-1)(q-1) \ (2)$

with q= n/p from (1) substituted into (2), we have

$\varnothing(n) = (p-1)(n/p-1)$

$\varnothing(n) = n - p - n/p + 1$ *Expanding the right side of the eqn.*

$p^2 - np + n - p + \varnothing(n)p = 0$ *multiply by p, and take to the left side.*

$p^2 + (\varnothing(n) - 1 - n)p + n = 0$

The roots of this equation would give one the values of p and, as n is known, the value of q drops out. However before this can be solved one needs the value of $\varnothing(n)$. As it has already been stated the value of $\varnothing(n)$ is therefore just as difficult to determine as the factorising of n, and one would need to discover $\varnothing(n)$ to solve the above equation.

The same idea holds for this attack as it does for the factoring of n. It really boils down to the feasibility of the attack, the time required and the general lifetime of the key. As it would take huge fiscal sum, a large amount of time and a lot of effort, it would be pointless mounting an attack where the keys last hours and the information encrypted would not be sufficient to recoup the costs.

Small Encryption Exponent

To increase the speed of encryption the size of the encryption exponent is made small ((m^a) where a very small). In some cases a=3 is used, but $2^{16}+1=65537$ is the most common value (This is even used in the SWIFT implementation). As this encryption exponent is small it will be easier to encrypt information, as (m^a) where a is small, is easy to calculate. However the decryption not be as quick as the value of b (decryption exponent) will not be small.

The attack for this method is based on D.Coppersmith Theorem. "Let N be a integer and Z_x be a monic polynomial d. Set $X = N^{1/d-e}$ for some e>0. Then given <N,f>, one can find all integers $|x_0| < X$ satisfying $F(x_0)=0 \bmod N$." [D.Boneh].

This attack shows in some instance the attack can be quicker than a brute force attack, but it still takes on exponential running time to determine d [D.Boneh, G.Durfee]. The theorem proves that a small exponent is susceptible to an attack, but due to the fact a small exponent is not used in the auction systems implementation of RSA this is not an issue.

Brute Force

It is certainly possible for a cryptanalyst to try every possible b until he stumbles on the correct one. Alternatively one could guess the plaintext and encrypt plaintext guesses until the exact same ciphertext appears. This would take a lot of time and effort, as it has been shown that a brute-force attack is even less efficient than trying to factor n.

The implementation of the protocol is used in such a manner that the plaintext encrypted by RSA on the clients side cannot be guessed. As this is a key created by the MD5 algorithm (See Appendix). This means that someone would have to first break MD5, guess the correct plaintext and key for MD5, and only then mount the plaintext attack on RSA. This is really infeasible and certainly not efficient.

Attempting all the decryption exponents has two problems, the first being that the exponent may not be found as one does not have $\varnothing(n)$, and secondly even for the next few years guessing and attempting all these numbers is beyond the power of computing.

Timing Attack

An attack that this algorithm is susceptible to is known as the Timing attack. One can measure the time it takes for the computer to decrypt messages received. This along with the information pertaining to the system can lead one to determine the suspected length of the decryption exponent. Armed with this information one will be able to guess the range in which the decryption exponent lies. Knowing this range one can simply try all the elements in this range until the correct decryption exponent is discovered.

There are a lot of assumptions made here, and essentially the cryptanalyst needs to be able to record minute timing details from the decryption object. This is rather an impractical attack for the current protocol as a lot of the information required would not be discernible from the server running the auctioning system.

6.8) Conclusion

RSA has shown over the last twenty years of its existence that it is a relatively secure asymmetric cryptosystem and should remain for a few years yet. This is assuming that no major breakthroughs in factoring, or that any other attacks, are developed in the near future. One should not be complacent, but nothing has indicated that this will happen in the near future, and for the time being the primitive used in the asymmetric cryptosystem is secure.

It has been shown how RSA has been implemented and what attacks may be mounted upon the protocol. The protocol was developed so that it minimised the effects of the attacks discussed on RSA. It should be impractical, infeasible in monetary and time constraints, to mount an attack on the auctioning system.

The next chapter will examine the first symmetric key cryptosystem RC6.

7) RC6

RC6 is a block cipher, or symmetric cipher based on the five-year-old RC5 system invented by Ron Rivest, one of the designers of RSA. There are four operations that the algorithm depends on; XOR, addition, multiplication and rotation.

RC6 was created to meet the AES requirements, the improved the security and efficiency of the algorithm. The RC series of ciphers has been undergoing revision of the algorithms. RC2 was the first algorithm released, and even to this day the workings of it are unknown, as a non-disclosure agreement is signed by anyone that uses it. The RC supposedly stands for "Rivest's cipher".

7.1) Overview

The reason I decided to use RC6 as one of the two symmetric ciphers is that it has been based on a relatively secure foundation. The RC algorithms have been in existence for awhile, and are used to this day in many commercial applications. The RC5 algorithm has been around for a number of years and has proven to be relatively secure, with no serious flaws. As RC6 is a natural progression of RC5, one hopes that it is stronger than its predecessor.

RC6 is what's known as a block cipher. Block ciphers encrypt blocks of plaintext at a time. Stream ciphers, which are also symmetric cryptosystems, encrypt or decrypt single bits. The RC6 algorithm is also known as RC6 w/r/b. Where w stands for the size of the words being encrypted, r is the number of rounds the algorithm is for executed, and b is the size of the key in bytes. I have used the algorithm default here as it speeds up the encryption and decryption process. The default values are w=32 and r =20. Essentially the algorithm uses thirty-two bit words and twenty rounds of operations.

However before one can examine the algorithm, its implementation, and attacks against it, it is pertinent to look at the theory behind symmetric cryptosystems.

7.2) Concepts

These background concepts behind the block ciphers are useful for both of the algorithms used in the auctioning application. As was discussed in chapter 5, confusion and diffusion is the aim of the encryption process and serves to hide statistical information.

Bayes Rule

This is based on what is called conditional probability. This in essence is the occurrence of event A, based on the occurrence of another event B. As is standard notation, the probability of A occurring by itself is P(A), and the probability of A occurring given that B has occurred is written as P(A|B).

Bayes rule is as follows,

$P(A|B) = P(A)P(B|A) / P(B)$, and $P(B)\neq 0$

For a good discussion of this and all the required proofs refer to [L.Bain & J.EngelHardt,1991]. This assumes that A's probability is dependent on B's probability. Another variation of this is if A and B are independent of one another, which would mean that $P(A|B)=P(A)$. This is known as independent events

The reason that Bayes rule is important is that it relates directly to the secrecy of the cryptosystem. What this boils down to is that the secrecy of the encryption algorithm is based on the plaintext and the key. If any probability can be concluded from the ciphertext about the plaintext without knowing the key, one then has a weak cryptosystem.

Entropy

Entropy is the measure of the amount of information in a message. The way in which it is measured is by the probability distribution formulae. The formula $\text{Log}_2\, m$, is commonly used where m is the total amount of meanings for the current message.

The entropy of a message is denoted by the function H(X). One has to determine the probability distribution of all the possible messages, where $x_1..x_m$ are all the messages. The probability of them occurring is $P(x_1)..P(x_m)$. H(X) is expressed as follows:

$$H(X) = -\sum_{i=1..m} P(x_i) \log_2 P(X_i)$$

This is useful as it gives an average amount of bits needed to encode the message. As an example consider three events that can occur with the following probabilities,

$x=½, y=z=¼$

The entropy of this would be as follows:

$$\begin{aligned} H(X) &= -x\log_2 x - y\log_2 y - z\log_2 z \\ &= ½ \log_2 ½ + ¼\log_2 ¼ + ¼\log_2 ¼ \\ &= (½)1 + (¼)2 + (¼)2 \\ &= 1.5 \end{aligned}$$

Now that we have a definition of the entropy, what can we use it for? If one expands that equation to include something called Conditional Entropy it will become clear.

Conditional Entropy

Conditional entropy is the entropy of some X, given some value of Y, where $x_1..x_m$ and $y_1..y_m$ are the random variables.

$$H(X|Y) = -\sum_{j=1..m} P(y_j) \left(\sum_{i=1..m} P(x_i|y_j) \log_2 P(x_j|y_j)\right)$$

This formula determines the uncertainty of some value X, given Y, where $y \in Y$. This is averaged for all the values in the finite set $y_1..y_m$. So if X was our plaintext, and Y our ciphertext, what is the uncertainty of the plaintext given that one knows the ciphertext. Conditional entropy allows one to measure how difficult it is to determine the plaintext once it is encrypted.

Key Equivocation

After examining the entropy of the plaintext and the ciphertext it is pertinent to discover the entropy of a key from the ciphertext. Using the standard notation where P is the finite set of the plaintext, C the finite set of ciphertext and K the keyspace

$H(K|C) = H(K) + H(P)\ \ H(C)$
Proof: See [D.Stinston]

The uncertainty (entropy) of the key given the ciphertext is, the uncertainty of the key and plaintext minus the uncertainty of the ciphertext (as its known). The security of the cryptosystem lies on the entropy of the key. The higher the entropy the harder it is to determine the key, and thus harder to discover the plaintext. This is the reason that larger keys invariably mean better security as the entropy increases.

Entropy of Languages

The entropy of a language measures the information per letter that the language holds. This is denoted by H_L.

$$H_L = \lim_{n \to \infty} (H(X^n) / n)$$

Where X^n is a random set of letters of length n in the language L. This is an approximation, and studies have shown the English language to have an average of approximately 1.5 bits per a letter. With the knowledge of the entropy of the language encrypted, one has a good idea of the message type, and one can then ignore keys that result in gibberish once the ciphertext is decrypted. These keys are known as spurious keys.

The reason the natural entropy of a language is important is that it allows one to work out the redundancy in the language.

Redundancy of Languages

This is the excess information in a natural language. In other words in an intelligent string of a language, the measure of letters that are not actually adding to the information in the string. This is denoted by R_L.

$R_L = 1 - H_L / (\text{Log}_2 |P|)$

P is the probability of a random string in the natural language. As the H_L of English is between $1.0 < H_L < 1.5$, one can approximate that R_L

$0.5 < R_L < 0.75$

This means that English is a highly redundant language, but as has been shown, so are all natural languages. As the ciphertext gives some information about the plaintext it is important to reduce this redundancy in the plaintext. The redundancy gives the cryptanalysts information regarding the plaintext encrypted.

Unicity Distance

Unicity distance attempts to find the minimum length of ciphertext for which a cryptanalyst will be able to determine an intelligible plaintext solution. Unicity distance is given by the following formula

$U = H(K) / R_L$

This is the entropy of the cryptosystem, which is based on the key entropy, divided by the redundancy of the language. A good cryptosystem should aim to remove the redundancy of the plaintext by using some form of compression. Since in the above formula, as R_L tends to zero, the Unicity distance tends to infinity and it becomes almost impossible to determine the correct plaintext for the current ciphertext.

Confusion & Diffusion

As has been mentioned before a good cryptosystem incorporates the two properties of confusion and diffusion. Confusion hides all links and relationships between the plaintext, ciphertext and the key. Diffusion spreads the statistical information held in the plaintext over the entirety of the ciphertext.

Using these two properties, statistical, linear and differential cryptanalysis becomes difficult. How does one hide the information that can be determined by unicity distances, entropies and statistical examination? One of the most powerful methods is by the use of a Feistel network.

Feistel Networks

In this methodolgy one takes the plaintext and divides it into two equal halves. These are commonly known as L and R, which stands for left and right respectively. Each round in the algorithm (or loop of operations) involves a process whereby the left and right values are swapped.

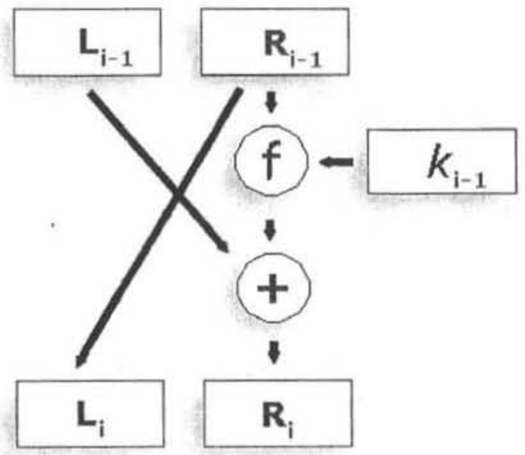


Fig 7.2.1 Feistel Network

$L_i = R_{i-1}$
$R_i = L_{i-1} \oplus F(R_{i-1},K_i)$

This process is repeated from i=1 to the max number of rounds. The function F, which is reversible, encrypts half the information with the current round key. The trick comes with the XORing of L_{i-1} with the function $F(R_{i-1},K_i)$ as this is a XOR process it is reversible and as the algorithm proceeds the data originally held in R and L is spread repeatedly after each round, as well as encrypted, making it exceptionally hard to determine any statistical information about the original plaintext.

S-Box

S-Boxes or substitution boxes help in strengthening the security of an algorithm. An S-Box is usually an array of predefined numbers. The S-Box takes input from a function and maps this input to an element in the S-Box, resulting in a new output obtained from the S-Box. The input is used as the index value in an S-Box array. This step is a non-linear step, as the S-Box is some random data. The resulting values returned are then used in the next round of the algorithm, and so forth.

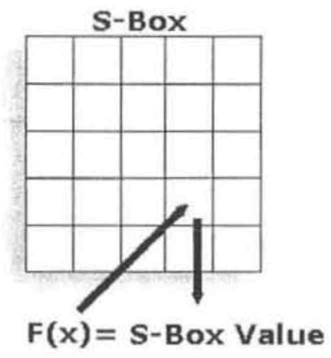


Fig 7.2.2 Extracting S-Box elements

The larger the S-Box, the harder it is for someone to cryptanalyze the system. S-Boxes are especially strong if the information contained within them is random, and independent of the key.

ECB

Electronic codebook (ECB) mode encrypts a block of plaintext at a time. If the message is too big it is broken into correct sized chunks, which are then, encrypted separately with the same key.

The benefit of this method is that each block of ciphertext is independent of the rest so an error in one block will not propagate throughout the ciphertext.

CBC

Cipher block chaining (CBC) mode encrypts a block of text, which is first XORed with the previous block of ciphertext. Each block encrypted will be based on the plaintext and the previous block of ciphertext.

The big problem with this method is that if just one bit of information is lost in transmission, all following ciphertext will not decrypt correctly and the message will be ruined.

Other Modes

There are a number of other methods including streams, output feedback (OFB) mode and cipher feedback (CFB) modes. However for a longer and more involved discussion please see [B.Schneier,1996]. The only mode encryption that is of interest in this application is the ECB mode that is used.

7.3) RC6 Algorithm

The RC6 algorithm has two important parts, the actual encryption/decryption and the key schedule. It is easier to discuss the key schedule first as it creates the S-Box used for encryption and decryption based on the users key. As RC6 is an AES candidate the amount of information regarding the algorithm is limited and hence most of the information comes from the paper [R.Rivest, *et al*, 1998].

The operations

The are a number of common operations performed in the key schedule, encryption and decryption. The are as follows:

A+B	Addition modulo 2^w
A-B	Subtraction modulo 2^w
A*B	Multiplication modulo 2^w
A⊕B	XOR modulo 2^w
A<<B	Rotation left of A, by B
A>>B	Rotation right of A, by B

[R.Rivest, *et al*]

Whenever one sees these operations in the following algorithm it is assumed that it is performed as described above. These operations are usually quick and relatively efficient on most operating systems. As the algorithm is made up of these primitive operations it is designed to take advantage of the speed benefits offered by them.

The Key Schedule

The key schedule is composed of three steps. The copying of the user key into an array (L), the initialisation of the S-Box array, and finally the mixing of the two arrays into the resultant key used for encryption and decryption.

Step 1

The user-supplied key, which is of size b bytes, is placed in the array L. The array L has c elements in it, L[0]..L[c-1]. The array is of type integer, and therefore the bytes of the user key need to be converted to integers. The first four bytes are placed in the array element L[0], and so forth.

The value of c is determined by the closest number of words derivable from the number of bytes of the user key (Word = Bytes / 4). If any extra bits are required, zero are appended to make up the missing terms.

c = (Length User Key) / 4, if Length Key = 0, then c= 1
Place user key into L[0] .. L[c-1]

Step 2

The next step is to initialize the S array. There are recommended starting values, P_w and Q_w. These values are mixed into the initial S array creating random values, which are independent of the key. To get as accurate as possible values for these two numbers, the word size binary numbers are used

$P_w = ((e - 2)2^w)$ (e=natural log base) = P_{32} = b7e15163 = 2.718281828...
$Q_w = ((\varnothing - 1)\ 2^w)$ (golden ratio) = Q_{32} = 9a3779b9 = 1.618033988...

The initial value of $S[0] = P_w$. The S array has 2r+3 elements, where r is the number of rounds used for the algorithm. (S[0]..S[2r+3]).

The S array is then modified with the following formula, initialising all its values. The S array elements are words.

$S[i] = S[i-1] + Q_w$

This process is repeated from i=1 to 2r+3. The operations are performed modulo 2^w, where w is the size of the words being used in the algorithm. The bigger the word size (w) of the system, the more accurate the values of P_w and Q_w, as specified in the paper [R.Rivest,1995]. "These values are somewhat arbitrary, and other values could be chosen to give custom or proprietary versions of RC6" [R.Rivest, *et al*]. This essentially means any values that are difficult to determine can be used to initialise the S array as long as they are independent of the key.

Step 3

The final step mixes the L and S array together. The operations are designed to be relatively one way, so it is difficult to determine the user key from the S array. The mixing is repeated by the maximum of c or (2r+4) multiplied by three.

LoopLimit = 3 * Max(c,2r+4)

Four temporary variables are used for the mixing process, A, B, i, j. All of which are initialised to zero before the operations are performed.

```
For loop = 0 to LoopLimit do
        A =S[i] = (S[i] +A+B) <<3
        B =L[j] = (L[j] + A+B)<<(A+B)
        I = (i+1) mod (2r+4)
        J = (j+1) mod c
End For
```

The result of x << y, means that x is shifted to the left by y places. All operations are carried out modulo 2^w. The S-Box array is the same for encryption and decryption. The key schedule algorithm is almost exactly the same one used in RC5, as it has shown to be quite successful in the past.

The Encryption Algorithm

Encryption with RC6 w/r/b, (word size/rounds/key size in bytes) is a block cipher so the plaintext is treated as such. Thus far the key has been generated and it resides in the S array, or the expanded key table. The S array is not quite an S-Box as the values in it are used sequentially as opposed to the lookup method used in an S-Box. The encryption encrypts four w bit words at a time. These words are placed into the variables A, B, C, D. They are also known as registers.

The plaintext is placed in the word registers using little endian notation, whereby the first byte of the plaintext takes up the low order byte of the word A. The fourth byte of the plaintext is the high order byte in the word A and the fifth byte of the plaintext then takes its position in the low order byte of B, and so forth.

The S array is initially added to the plaintext and thereafter it is mixed in with the encryption function. This helps in distributing the key and the plaintext, and increasing the entropy of both. The algorithm is as follows:

```
B = B + S[0]
D = D + S[1]
For (i=1;i<r;i++)
        t = (B * (2B+1) <<$\log_2 w$
        u = (D * (2D+1) <<$\log_2 w$
        A = ((A $\oplus$ t)<<u) +S[2i]
        C = ((C $\oplus$ u)<<t) +S[2i+1]
        (A, B, C, D) = (B, C, D, A)
End For
A = A + S[2r+2]
C = C + S[2r+3]
```

The values of S[0] and S[1] are added to the B, and D respectively. The main operation then begins and is repeated r times. The user specifies all the values of r, w and b. The variables t and u are temporary for holding the information as it is modified in each round. The interesting aspect about this algorithm is the fact that it is so short and simple. This leads to efficiency in computation. The aim of the designers of RC6 was to meet three requirements, security, simplicity and good performance.

The values of A, B, C, D are modified in a Feistel network fashion. The actions performed by (B * (2B+1) <<$\log_2$ w and (D * (2D+1) <<$\log_2$ w, are actually the transformation function being applied to half of the values each round. The one to one function that is used to change each register is:

$F(X) = x * (2x+1) \bmod (2^w)$

Theorem:

$F(X) = x * (rx+s) \bmod (2^w)$
Is one to one over $\{0,1,2,..,2^{w-1}\}$, whenever r is even and s odd for any integer w>=0

Proof: by contradiction

Assume $F(x) = F(y)$ for $x <> y$
Then
$x(rx+s) = y(ry+s) \bmod 2^w$
$r(x^2-y^2) = s(y-x) \bmod 2^w$.

Thus $r(x+y)(x-y) = -s(x-y) \bmod 2^w$.
But 2 divides the left side more times than the right side. QED [R.Rivest]

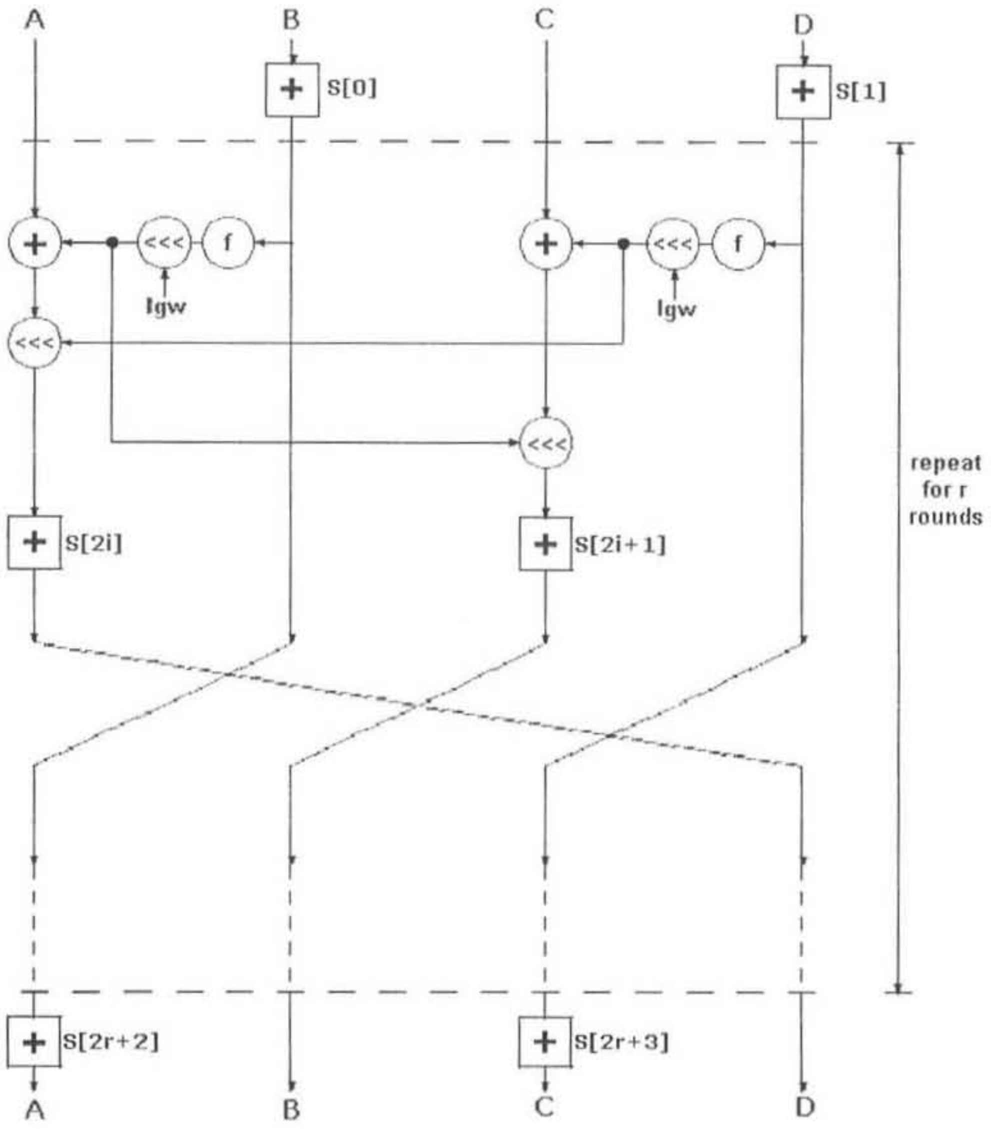


Fig 7.3.1 Encryption with RC6-w/r/b. F(x)=x (2x+1) [R.Rivest, *et al*, 1998]

The function's design is to diffuse the information being encrypted, thereby hiding all relationships and statistics. After the transformation function is applied the values are XORed with previous values and then rotated. Once these values have been rotated the next elements of the key are added. The left rotations are dependent on the values transformed in u and t.

One can see that this plaintext is split up and values are swapped at the end of each round. The values of A and B are not simply swapped with C and D, instead they are mixed, (A, B, C, D) = (B, C, D, A). This means that the values are equally shifted to the left every round and each element will be changed accordingly. It will obviously take four rounds for each element to move full circle back to its original register.

As r and b increases so does the security of the algorithm. This is because the plaintext is distributed more evenly as the rounds increase, and a larger key will make it more difficult to guess (increased key entropy). However this tends to slow down the speed of the encryption and decryption. The S array is not like other algorithms that can access any element of the array at once, which requires the entire array to be in memory. It uses the elements sequentially, and therefore only one element at a time needs to be dealt with. This saves vast amounts of memory, and is a big reason why RC6 is targeted at applications that use minimal memory (Smartcards, etc)

The addition of the key to the registers before the rounds begin

B = B + S[0]
D = D + S[1]

and after the rounds end,

A = A + S[2r+2]
C = C + S[2r+3]

is known as the pre and post whitening stages. The reason for these stages is that if it is not done the plaintext will reveal the input in the first part of the encryption, and the ciphertext will reveal the part of the input of the last part of the encryption. This is something that can be attacked so the key is added to hide this information. This forces the cryptanalyst not only to guess the key used in the algorithm but also to guess the values used for whitening.

The Decryption Algorithm

As the encryption algorithm makes use of left rotations, XOR's, multiplication and the swapping of values, all the decryption algorithm does is invert each stage of the encryption algorithm. This includes right rotations, XOR's, divisions and reverse swapping of values.

The ciphertext is placed in the same registers A, B, C, D, in the exact same format little Endian format the encryption algorithm uses. The same number of rounds r are used, and the same keys in the S array are used.

The algorithm is as follows

```
C = C - S[2r+3]
A = A - S[2r+2]

For (i = r ; i> 0 ;i--)
        (A, B, C, D) = (D, A, B, C)
        u = (D*(2D+1)) <<$\log_2 w$
        t = (B*(2B+1))<<$\log_2 w$
        C = ((C-S[2I+1])>>t) $\oplus$ u
        A = ((A-S[2I])>>u) $\oplus$ t
End For

D = D-S[1]
B = B-S[0]
```

The decryption starts at the last round and essentially undoes what the encryption algorithm performed. Reversing all the results round by round until the original values are attained.

An Example

As an example, using w/r/b=32/4/4. Please note that some values may be negative, as this is due to the rotations of the integer values. Integer values must be in the range $(-2^{32}+1)$ to 2^{32}, any value exceeding this will simply wrap around, hence the negative values.

Let the user key be,

0 1 2 3

This is then given to the key schedule algorithm, which then expands the key schedule.

C = 1, therefore there is only one element in L

L[0] = 50462976

The initial values of S are then calculated for 2r +4 elements of S.

S[0] = P_w = 0xb7e15163
S[0 .. 12] = {1209970333 1444465436 -196066091 -1836597618 817838151 -822693376 1831742393 191210866 -1449320661 1205115108 -435416419 –2075947946}

The values of L[] is mixed into S[], which results in the final S array value of

S[0..12] = {1089557206 -2003891483 2122675648 -172460913 -1170572592 955928541 -1506510775 -129986502 -524075534 1310761307 -1839897398 1162524834}

Now to encrypt the message "Hello World!", first one converts it to integer values, this is done using the CryptLib method ConvertAnyString(String), that converts the string to bytes and then to integers.

Hello World!
1819043144 1867980911 560229490 0

When the values are passed into the encryption algorithm they are placed in the registers A, B, C, D where

A=1819043144, B=1867980911, C=560229490, D=0

A bit of pre-whitening takes place and the values of B,D are increased by S[0] and S[1] respectively.

A=1819043144 B=-1337429179 C=560229490 d=-2003891483

The algorithm then starts performing the rounds. It determines t, u and then the resulting A and C values. The values A, B, C, D are then all swapped.

Round=1
t=-1227714832
u=621646576
A=1782468734

```
C=1782468734
Before
A=1782468734 B=-1337429179 C=-197494536 d=-2003891483
After
A=-1337429179 B=-197494536 C=-2003891483 d=1782468734

Round=2
t=-374657264
u=-578187064
A=-1651060695
C=-1651060695
Before
A=-1651060695 B=-197494536 C=-752376605 d=1782468734
After
A=-197494536 B=-752376605 C=1782468734 d=-1651060695

Round=3
t=-1134760272
u=-2038191764
A=261684431
C=261684431
Before
A=261684431 B=-752376605 C=-246186761 d=-1651060695

After
A=-752376605 B=-246186761 C=-1651060695 d=261684431

Round=4
t=-184732878
u=1361801790
A=716825574
C=716825574
Before
A=716825574 B=-246186761 C=-696265647 d=261684431
After
A=-246186761 B=-696265647 C=261684431 d=716825574
```

After the rounds are complete, the post whitening takes place.
A=-2086084159 B=-696265647 C=1424209265 d=716825574

Finally our ciphertext is
-2086084159 -696265647 1424209265 716825574

To decrypt, the ciphertext and key are passed to the decryption algorithm, and the first thing that takes place is the de-whitening

A=-246186761 B=-696265647 C=261684431 d=716825574
This is where C = C – S[2r+3] and A = A – S[2r+2]. The rounds then begin starting at the highest round and decreasing there after

```
Round=4
A=716825574 B=-246186761 C=-696265647 d=261684431
t=-184732878
u=1361801790
A=-752376605
```

```
C=-246186761
```

Until we get to the last round,

```
Round=1
A=1782468734 B=-1337429179 C=-197494536 d=-2003891483
t=-1227714832
u=621646576
A=1819043144
C=-1337429179
```

After the rounds are complete the values left in the registers are the following:
A=1819043144 B=-1337429179 C=560229490 d=-2003891483

The final values of the key are then subtracted (D = D-S[1] and B = B-S[0]) which results in

A=1819043144 B=1867980911 C=560229490 d=0

These are the same values as our original plaintext, which is "Hello World!"

7.4) RC6 Implementation

Due to the non-disclosure agreement that was signed with NIST, I cannot display any of the actual code for the algorithm. However I will discuss pseudo code for the RC6 algorithm.

The Class has a number of predefined variables, these are the S array and the values of r and w. All of these are stored as integers. The RC6Crypt initialise method performs the following:

```
//RC6Crypt.Java RC6Crypt method
CL = New CryptLib
W=32
R = 20
User_key[32] = 0
S = generate_Keys(User_key,32)
```

A new object of the CryptLib class is created, as a few of the methods are used by the encryption and decryption functions. A default user key is created which is set to zero. This enables a default S key array to be created, when generate_Keys is called. The values of w and r are set to 32 and 20, but these may be changed to increase the security of the algorithm at a later date.

There are two encryption and decryption functions, and the method for generating the keys.

Generate_Keys(userKey, size)

This method takes a user key of bytes and uses it in the key schedule to expand the key into the S array. It returns the S array of type integers.

```
S = new integer[2*r+4]
S[0] = 0xb7e15163
For (loop=1;loop<(2*r+4);loop++)
        S[loop] = S[loop-1] + 0x9e3779b9
End For
```

This section initialises the S array as described in step two of the key schedule.

```
c = key_length / 4 + Key_length % 4
L = new integer [c]
For (loop=0;loop<key_length;loop+4)
        L[loop/4] = ConvertToInteger(UserKey[loop..loop+3])
End For
```

This section determines the length of c (Amount of words in the bytes), and then converts the bytes to integers which are placed in the L array, as described in step one of the key schedule.

```
A=B=i=j=v=0
v = max(c,2*r+3)
for (loop=1;loop < v;loop++)
        A = S[loop] = (S[i]+A+B)<<3 or (S[i]+A+B)>>29
        B = L[Loop] = (L[j]+A+B)<<(A+B) or (L[j]+A+B)>>(-1*(A+B))
        I=(I+1) % (2*r+4)
        J=(j+1) % c
End for
```

The third step of the key schedule initialises all the variables used to zero. Then it determines the maximum of c or 2*r+3. The S array and L array manipulation then take place. The reason that I have ORed both of these steps with the two rotations (<<3, and >>(32-3)) is that this stops the timing attack that is discussed later.

Finally once the key schedule is complete it returns the S array.

RC6Encrypt(plaintext String, userkey)

There are two version of the encryption, one accepts a string and the other four integer values in an array.

```
Tempint[] = CryptLib.ConvertAnyString(Plaintext)
For (loop=0;loop<TempInt.Length;loop+4)
        BlockPlaintext[4] = Tempint[loop,loop+1,loop+2,loop+3]
        BlockPlaintext[4] = RC6Encrypt(BlockPlaintext[],use key)
        Ciphertext += BlockPlaintext
End for
```

The current method accepts the plaintext, and converts it to an array of integers, which is of uniform length that is divisible by four. The ConvertAnyInteger pads the end with zeros if it is not long enough. Then the code copies four elements of the plaintext integers into an array of four elements. These four elements are then encrypted using the other RC6Encrypt method, passing the array of four plaintext integers and the user key. The result of this is then added to a string called ciphertext. The loop continues looping until all elements in the plaintext integer array Tempint have been encrypted. The ciphertext string is returned.

RC6Encrypt(plaintext[] Integer, userkey)

This method accepts the four plaintext elements and the key. This method then proceeds to encrypt the four integer (also known as word) values. The plaintext elements are copied to variables a, b, c, d.

```
a=b=c=d=t=u=0
a=plaintext[0], b=plaintext[1], c=plaintext[2], d=plaintext[3]
S= user key
```

The algorithm then proceeds with the whitening of the input.

```
b+= S[0], d+=S[1]
```

Once this has been achieved the rounds begin,

```
For (loop=1;loop<=r;loop++)
        t= (b*(2*b+1)
        t = (t<< $log_2w$) or (t>>(w- $log_2w$))
        u = (d*(2*d+1))
        u = (u<< $log_2w$) or (u>> (w-$log_2w$))
```

This is done as a two step process, as then it is more efficient to calculate the value of x*(2*x+1) once, and then performs the data dependant rotations. Wherever possible I have tried to speed up the encryption by hard coding elements that would not change in the application.

```
        a = (a Xor t)
        c = (c Xor u)
        a = ((a<<u)or(a>>-u)) + S[2*loop]
        c = ((c<<t) or (c>>-t)) + S[2*loop+1]
        (a, b, c, d) = (b, c, d, a)
End For
```

To optimise calculations the same procedure is followed in the calculations of a and c initially. Thereafter the shifting, left and right, of a and c is performed. Finally the values of a, b, c, d are all swapped.

The post whitening then completes the encryption

```
a+= S[2*r+2]
c+=S[2*r+3]
Plaintext[0]= a, Plaintext[1]= b, Plaintext[2]= c, Plaintext[3]= d
```

The plaintext array of four integers is then returned.

RC6Decrypt(ciphertext String, userkey)

The same process is used in this method, as the RC6Encrypt(String, key) method. The string is converted to a uniform length integer array. A loop is then executed taking four of these integers and calling the RC6Decrypt(Ciphertext[] integer, Key) method. The results are then appended to a string called plaintext.

```
Tempint[] = CryptLib.ConvertAnyString(CipherText)
For (loop=0;loop<TempInt.Length;loop+4)
        BlockCipherText[4] = Tempint[loop,loop+1,loop+2,loop+3]
        BlockCipherText [4] = RC6Decrypt(BlockCipherText[],use key)
        Plaintext+= BlockCipherText
End for
```

The results are then returned in the plaintext string variable.

RC6Decrypt(Ciphertext[] Integer, UserKey)

The same registers and variables are used in this method as the RC6Decrypt (Plaintext[] Integer,UseKey) method. They are all initialised as the algorithm proceeds.

```
a=b=c=d=t=u=0
a=Ciphertext[0], b= Ciphertext [1], c= Ciphertext [2], d= Ciphertext [3]
S= Userkey
```

The removal of the whitening then takes place

```
a-= S[2*r+2]
c-=S[2*r+3]
```

And the rounds begin from the r down to the last round one.

```
For (loop=r, loop>0;loop--)
        (b, c, d, a)= (a, b, c, d)

        t= (b*(2*b+1)
        t = (t<<log2w) or (t>>w- log2w)
        u = (d*(2*d+1))
        u = (u<< log2w) or (u>>w- log2w)
        a -= S[2*loop]
        c -= S[2*loop+1]
        a = ((a<<u)or(a>>-u)) Xor t
        c = ((c<<t) or (c>>-t)) Xor u
End for
```

This reverses the operations performed by the encryption process. The same optimising methods have been used for the decryption method as were used for the encryption method.

```
b -= S[0], d -=S[1]
Ciphertext[0]= a, Ciphertext[1]= b, Ciphertext[2]= c, Ciphertext[3]= d
```

The ciphertext integer array of four elements is returned, containing the recently decrypted ciphertext.

7.5) RC6 Usage

The client and server application has a copy of the RC6Crypt class. This is initialised on the initialisation of the CryptLib class. As has been mentioned before the client generates the user defined key. The client sends this key, first encrypting it with RSA, to the server. The server then decrypts the RSA message and obtains the key. The key is specific to each client, and therefore the copy of the key is kept in each clienthandler class. Therefore each client has their own session key which will be different from any other client key. Once the key is received it is then given to the generate_Keys method which executes the key scheduler which will create the correct key for encryption and decryption.

The Cryptlib class deals with the administration of the algorithms and their uses. It monitors which algorithm is being used, it also executes the corresponding key scheduler method. This generates the key in the S array. The client and the server perform the same key schedule method and hence they will have the same key for the current client's session. Once the key has been generated, and there are no problems, the encryption then changes from RSA to the specified encryption algorithm (in this case its RC6).

```
//ClientHandler.Java - send method
...
        If (encrypt_mode>0)
                        CurrentMesg= CL.Encrypt(CurrentMesg,Key)
        EndIf
...
```

Thereafter whenever the SendMsg method is called by the client, and Send method by the server, the messages are first passed to the Cl.Encrypt() method. This method passes the message to the RC6Encrypt(String,UserKey) method, which encrypts the message and returns a set of numbers in a string format. This is the ciphertext, and it is then sent to the recipient of the message.

```
//ClientHandler.Java - run method
...
        CurrentMesg = ReadStream
        If (encrypt_mode>0)
                        CurrentMesg= CL.Decrypt(CurrentMesg,Key)
        EndIf
...
```

When the client or server receives a message, it is first passed to the CL.Decrypt method which passes it to the correct decryption method (RC6Decrypt(String, UserKey)). This method decrypts the message, and then the message is dealt with. If any errors occur while decrypting messages they are ignored, and the error is reported to the administrator.

7.6) Timing

Speed tables are used to show the reader how much quicker a symmetric cryptosystem over asymmetric cryptosystems is. The algorithm has been optimised in one or two places, but for the sake of readability it is as close as possible to the pseudo code. The Optimised implementation results of RC6 are from [R.Rivest, *et al*] performance figures.

Language	Scheme	Blocks/Sec	Mbits/Sec
Auction RC6			
JDK 1.2	Encryption	4545.45	0.554
JDK 1.2	Decryption	6250.0	0.762
Optimised Implementations			
JDK (1.1.7)	Encryption	12100	0.19
JDK (1.1.7)	Decryption	12300	0.19
Java (JIT)	Encryption	197000	3.15
Java (JIT)	Decryption	209000	3.35
ANSI C	Encryption	325000	5.19
ANSI C	Decryption	353000	5.65

Fig 7.6.1 Encryption and Decryption figures of RC6 32/20/32

The Auction version of RC6 has been implemented and tested on an Intel PII 450. The encryption's and decryption's were performed one thousand times each to obtain a good average. It can be seen that the Auction implementation on the whole is slower than most of the optimised figures quoted. These optimised implementations were executed on an Intel Pentium Pro 200. In the article by [B.Schneier, *et al*] the performance implementations are discussed, as it is noted that the performance of RC6 on a Pentium Pro is three times faster than on any other Pentium class chipset. The reason for this is that multiplication and variable rotation are not naturally supported by these chipsets.

RC6 also has huge differences in speed depending on the type of Chipset (ie RISC, Intel, etc). The main reason is because of the integer multiplication that takes place. It is interesting to note though, that the optimised version of RC6, when executed with a JIT compiler is not that much slower than the ANSI C version. As Java has always been hailed as much slower than C on all platforms this is an encouraging sign. [B.Schneier, et al], relates a more in depth discussion of the performance of RC6 and the other AES candidates on different platforms (32-bit and 64-bit).

The results of the timings do indicate however that the current implementation used for the auction application will more than suffice. The average message size is only a few bytes (approximately 32 bytes, 1 block = 16 bytes) long, and approximately five blocks need to be encrypted or decrypted per a client. The server application could handle easily over one thousand clients.

7.7) Attacks

As RC6 is based on RC5, many of the attacks will follow the current trend of attacks on RC5. There are two ways in which the algorithm is going to be broken. Breaking the key schedule, or breaking the algorithm, and both will be discussed.

Key Schedule Attack

The RC6 algorithm can use up to 2040 bit keys, which means the key space is very large. It has been suggested that when the key size is very large there is a good chance that there are similar keys. These are known as equivalent keys, which are keys that will perform very similar encryption [M-J. O. Saarinen]. If this is the case then the key space is not as large as initially expected. There is no proof that different keys yield the same round of keys. As an example, the chance of two 256-bit keys existing and yielding a table of 44, 32-bit round keys is approximately

$2^{2*256-44*32} = 2^{-896}$ (Highly unlikely in other words.) [R.Rivest, *et al*]

RC6 uses forty four keys for a twenty round implementation. Two are used for pre-whitening (S[0] & S[1]), forty are used for the rounds and two more are used for post whitening (S[43] & S[44]). If the keys used for encryption of the same plaintext, differ by S[43] and S[44] (two key elements), the ciphertext will be constantly different as well. This means that one can start to look for matches when using different keys. By guessing just two key elements, one can decrease the key space considerably. Instead of 44 elements, only 42 are required. This can be used to mount an attack on the system, and it will certainly decrease the amount time required for a brute force attack.

Using this idea, it has been suggested by [M-J. O. Saarinen], that one can guess round values when generating the key. It has also been shown that if one selects a value of L[i], it will only change the values from S[i+1,..,43]. If one is able to determine S[43] and S[42], one can get an approximation of L[43-i]. The problem here is that the key scheduler is a one way algorithm, in as much as the rotational value (A+B) is lost once the rotation has been completed for the rounding, making it difficult to reverse the process. This is what makes the attack impractical.

As this is the same key schedule that has been used for RC5, it has been extensively cryptoanalyzed. What has been concluded is that due to the nature of the scheduler and the algorithm that uses it, there are no weak keys. In other words all keys should be equally as strong.

Timing Attack

The timing attack comes about because of the data dependant rotations of the encryption algorithm. "Rivest notes that on 'Modern microprocessors, a variable rotation...takes constant time'"[H.Handschuh, H.Heys]. As this is not always the case with modern hardware it can lead to some susceptibility. The bits have to be moved or swapped bit by bit. Each movement takes time, and one can determine from this what the rotational value is. This attack was original noticed in RC5, but the same idea may be extrapolated out to RC6.

A full discussion on the attack on RC5 is discussed in [H.Handschuh, H.Heys], and some interesting ideas can be drawn from this attack. The attack measures the time of the entire rounds of encryption. It then be easily determined what the time is for one round of operation. As all the operations are known in each round, one can break it up into the fixed time operations and the variable time operations per round. This fixed time operations do not change, and hence if once subtracts it from the total round time one is left with the variable operations time. This is the time used for rotating $((A \oplus t)<<u)$ and $((C \oplus u)<<t)$. With work and approximations one is able to determine important values for each round which may allow reconstruction of the plaintext.

This attack is aimed mostly at the likes of smart cards, etc. It would be difficult to gather the timing information in the current implementation due to the fact that the cryptanalyst does not really have access to the machines being used, as well as the timing information. The current implementation is not immune to this attack, so a solution has been put in place to nullify the attack.

As recommended by [R.Rivest, *et al*], a modification to the implementation of the algorithm which does not change the encryption is used. To do this the left shift of x bits is ORed with a right shift of (w-x) bits. The time of the rotation now becomes data independent, and the timing attack will not succeed.

Linear and Differential Cryptanalysis

To mount any of the Linear or Differential attacks on the system it needs large amounts of corresponding ciphertexts and plaintexts. This in practise is difficult to obtain, as the auction system will probably not generate the kinds of plaintext/ciphertext that is required, and even then the attacker will need to know the values of the plaintext/ciphertext pairs.

Differential cryptanalysis examines the evolution of similar plaintexts as they are encrypted with the same key, giving probabilities for each iteration of encryption. This allows one to make approximations of the plaintext as it is encrypted and the keys used to encrypt it. The most powerful differential attack on RC6 uses differences in values (i.e A-B) to help obtain the probabilities of plaintext values. However the main problem with this is that it requires large amounts of plaintext. As mentioned in [R.Rivest, *et al*], RC6 32/8/b (8 round) requires 2^{76} of chosen plaintext it mount such an attack. When one expands this attack for greater rounds it does not fare as well as it becomes exponentially harder on each rounds iteration to determine the keys and plaintext.

Linear cryptanalysis uses known plaintext to approximate the actions of the algorithm. This is done incrementally with large amounts of plaintext, and one can then start determining information about the key. The attack uses a linear approximation over a number of rounds regarding the quadratic function. This method works relatively well over a short number of rounds (6 to 8), but thereafter the numbers dealt with and required amount of plaintext to mount the attack becomes incredibly large. The suggested amount of plaintexts to mount a linear attack is of the order 2^{182}, which is more than current auction application will encrypt in a lifetime of operations. (To encrypt 2^{64} and 2^{80} blocks of data on 50 machines capable of encryption rate of 10^{12} bits/sec would be 1 and 98000 years respectively [R.Rivest, *et al*])

These techniques of cryptanalysis will work, but there are really three problems that prevent them from doing so. The amount of plaintext, and ciphertext required is not even attainable. The computing resources required would be enormous, and the time it takes to perform all these attacks would invariably be an exponentially large number. Bearing in mind that the auction application does not deal with large fiscal values and the life span of the messages is relatively short, barring any huge advances in cryptanalysis and computing power the algorithm is secure enough.

Brute Force

It has been shown that to mount a brute force attack on the system, the exhaustive key search difficulty is estimated to be the minimum of (2^{8b}, 2^{1408})[R.Rivest, *et al*], where b is the number of bytes of the key. If the user key were larger than the table of keys it would be easier to guess this latter.

The problem with the brute force attack is that it gets exponentially difficult to perform as the size of the key increases, as there are more variations to attempt. This means that the algorithm does not really have a weakness and the only other alternative is to try all keys. This shows that the algorithm is strong, and unless a weakness is found it will take a large effort in computing, fiscal and time resources to determine the ciphertext. As the life span of the keys and messages is not all that long, it would be improbable that the ciphertext messages examined would be broken within the required time. The opportunity cost of breaking the ciphertext message is not quite enough for someone to mount this exorbitant attack on the system.

7.8) Conclusion

The RC6 cipher is based on a proven secure algorithm, and has been improved to take advantage of 32-bit systems. It was decided to stick with the suggested implementation of the algorithm RC6 32/20/b as it has been shown in the literature available that anything less than this will weaken the system.

One has to choose between speed and security trade-offs. The higher the number of rounds the slower the algorithm, but the more secure it becomes. At the moment the algorithm more than meets the requirements of the auction application in both the speed and security departments.

With current trends and technologies it is not feasible or viable to attack the auction system, and unless some inherent weakness in the algorithm is discovered it will protect the information more than adequately.

The next section will examine the implementation of and attacks against MARS.

8) MARS

The MARS algorithm was designed and created by IBM as a replacement for the DES cryptosystem. IBM originally designed DES in the seventies. MARS is a result of a group of eleven IBM cryptographers, and majority of information comes from the paper by [C.Burwick,*et a*,1998].

8.1) Overview

The MARS algorithm was designed with a number of predefined specifications in mind. The choices as related by [C.Burwick,*et a*,1998] are as follows:

Operations

The algorithm was designed to take advantage of the current computing technology. This involved the usage of strong operations, which are operations that can be easily and quickly performed by computers.

Structure

The structure of the algorithm is very important, and it has been shown that different parts of the structure perform different security actions. MARS was designed to use a mixed structure.

Analysis

All the components of the MARS have been designed so that extensive analysis may take place. This enables one to examine each component and note is strengths and weaknesses.

As an introduction to symmetric ciphers has been given in the previous chapter we may now proceed to the MARS Algorithm itself.

8.2) The MARS Algorithm

MARS is a symmetric key block cipher, which supports variable key sizes. The block size used is 128-bits, and the key size varies from 128 bits to 1248 bits. As has been indicated the MARS algorithm was designed with a number of prerequisites in mind.

The operations used in the MARS algorithm are XOR's, multiplication, addition, subtraction and rotations. The XORing of keys and information mixes the two together. All XOR's have additional operations included, as it is not a strong cryptographic primitive operation by itself. It is however a reversible operation which makes it very useful for undoing the operations initially performed.

Addition and subtraction are standard operations on any processor, and one will find that they are incredibly quick to perform. Hence the reason they are so common in most algorithms today. Multiplications were initially very expensive operations on computers, however this has changed as technology has developed, and it now only take 2 machine cycles to complete on modern architectures.

The rotations used are of two types, fixed and data dependant. The fixed rotations are used to obtain the required bits out of data. The data dependant rotations are used because of their strong cryptographic results, which prevents certain types of attacks. There are a few weaknesses but these will be addressed later. On the whole rotations are relatively quick operations on modern computer architecture.

Another operation that takes place is that of the table lookup of the S-Box. This is the basis of security in the MARS algorithm. It has been shown that S-Box's resist differential or linear attacks, as they are non-linear in operation. The problem with the S-Box (512, 32 bit words in MARS) is that the operations are quite costly in resources. It require memory lookups, and the S-Box is usually memory intensive as this is where it resides.

The development of MARS was aimed at complementing the current strengths of computer architectures today. The algorithm requires an S-Box, an expanded Key and the encryption/decryption functions that will be discussed next, and are shown in the paper by [C.Burwick, *et al*].

The S-Box

The S-Box for MARS has been developed to resist linear and differential attack by carefully picking the entries used. The elements have been generated using SHA-1 (A hashing algorithm). The three inputs were π, e and a constant c, which was changed for a number of iterations. The S-Box that was developed went through 2^{26} different values of c until a value was found that met the requirements below.

Each S-Box and its elements was checked for certain properties. Some of the differential properties were as follows

- No all zeros or all ones in a word
- Every entry differs by at least four bits
- S has no entries where $S[i]=S[j]$ or $-S[i]=S[j]$ or $S[i]=-S[j]$ with $i \neq j$

The linear properties:

- Two Consecutive bits have a probability of 1/30 of occurring.

The generated S-Box has been checked for strengths of its values as they make it difficult for analysis to take place. This S-Box is recommended if one wants to keep MARS secure. Other S-Box's may have weaknesses or inadequacies that could be exploited.

The Key Schedule

The key schedule expands the user key, which is comprised of n 32 bit words, into an array K[] of forty words. The user key requires no predetermined structure, all structure is created in the key generation. The key elements created have two provisions, the last two bits are set to one, and there are no elements that contain all ones or all zeros. There are four steps in the key schedule algorithm.

Step 1

The first step initialises an array T[], which has forty seven elements. The T[] array is indexed as -7..39. The first seven entries are then filled with the first seven entries of the S-Box.

$T[i] = S[i+7]$, $T[-7]=S[0]$ .. $T[-1]=S[6]$,

Once this has been completed the user key is then mixed in using the following formulae,

$T[i] = ((T[i-7] \oplus T[i-2]) << 3)$, $i=0..38$
$T[i] = T[i] \oplus k[i \bmod n] \oplus i$

This XOR's two of the current T[I] values, and then shifts these values to the left by three positions. Once that has been completed, the value of T[I] is then XORed with the user key k. The index to determine which user key is used is given as (i mod n), where n is the length of the user keys in words (n = UserKey Length). This means that if the index i is out of the user key index range, it then starts repeating the keys used so that all the values of T[i] will have user keys mixed in. The aim is spread the user key throughout the expanded key. The key is then XORed with the value of the index.

$T[39] = n$

This is used so that no two keys of different lengths generate the same key, If one has two different length user keys of all zeros, it will force the resultant keys to be different.

Step 2

Once the array has been created it is then mixed using a Feistel network. There are two operations that are performed seven times.

```
For (rounds = 1 to 7)
    For (i = 1 to 39)
        T[i] = (T[i] +S[Low 9 bits of T[i-1]]) <<9
    End For
    T[0] = (T[0] +S[Low 9 bits of T[39]]) <<9
End for
```

The inner loop adds the current T array elements to a S-Box element. The index used to determine which S-Box element to use, is the lowest nine bits of T[i-1]. Once this loop has finished the T[0] operation is performed manually as when i=0 the T[i-1] element equals T[-1], and one wants wrap around of the elements.

Step 3

The words in the T[] array are then placed in the 40 element key array K.

```
For (i=0 to 39)
    K[7*i mod 40] = T[i]
```

The words are then placed in the K array in the order specified by seven multiplied by the loop variable i modulus forty. The elements will be placed such that

K[0] = T[0] , K[7] = T[1], K[14]=T[2].., K[35] = T[5], K[2] = T[6]

and so forth. This means that the elements are reordered.

Step 4

The final step checks the keys that are used for multiplication in the encryption and decryption. The check is to prevent the formation of weak keys, such as zero or one, as this will cause the algorithm to be susceptible to cryptanalysis. The keys that are checked are K[5],K[7]..K[35]. Any key is considered weak if it contains more than nine consecutive zeros or ones.

```
For (i=5 to 35, i+2)
        j = least 2 bits of K[i]
        w = K[i], with least two bits set to one.
        M = bit mask of K[I],
        If M = 0 then K[i] ≠ weak
        End if
        If M = 1 then K[i] =weak
                Else
                        r= least five bits of K[i+3]
                        K[I] = w⊕((B[j]<<r) OR M)
End For
```

If there are any ten consecutive zeros or ones in the value of w, the key is weak. To resolve this, whenever there are two or more consecutive zeros or ones in w, the last bit of the consecutive values are changed to zero.

If w = $0^3 1^{15} 0^{12} 01$(superscript indicates the amount of consecutive ones or zeros), then M consider consecutive runs of zeros or ones a one, so

M= $0^3 1^{27} 01$,

where $1^{15} 0^{12}$ are considered by the mask as twenty seven ones. Now one must change the zeros to ones at the end of each consecutive list of ones or zeros. So M becomes

M =$0^3 0 1^{13} 0 0 1^{10} 0 0 1 = 0^4 1^{13} 0^2 1^{10} 0 0 1$

One uses the Key elements K[i+3] to determine a data dependant rotation r. The array B[] contains the elements in the S-Box at position 265..268. They have non - repeating values in them, where no large consecutive zeros or ones exists and are well suited for the rotation by r. This is then ORed with M, and the result of this is XORed with w, resulting in the new K[i]. As the values in B are known the new values of K[I] will not be a weak key.

Encryption Algorithm

There are a number of arrays that are used. S[] is the S-Box that has five hundred and twelve 32bit words (S_0 = first 256 elements, S_1 = second 256 elements). K[] is the key generated by the key schedule algorithm, which consists of forty 32bit words. D[] is the input array consisting of four 32bit words.

There are three phases in the encryption process, forward mixing, key transformation and backward mixing.

Forward Mixing

The forward mixing performs pre-whitening. This starts with the first four elements of the key K being added to the plaintext elements in D[].

```
For (i= 0 to 3)
        D[i]= D[i] + K[i]
```

Once this has been performed the mixing commences in an eight round feistal network process.

```
For (i = 0 to 7)
        D[1] = D[1] ⊕ $S_0$[lowest byte of D[0]]
        D[1] = D[1] + $S_1$[Second byte of D[0]]
        D[2] = D[2] + $S_0$[Third byte of D[0]]
        D[3] = D[3] ⊕ $S_1$[Highest byte of D[0]]
        D[0] = D[0] >>24
        If I=0 or 4 then D[0] += D[3]
        If I=1 or 5 then D[0] += D[1]
        (D[0], D[1], D[2], D[3]) = (D[1], D[2], D[3], D[0])
End for
```

This mixes the plaintext with the key and itself rotates the first plaintext element D[0] to the right by twenty four (Note: Word length = 32, therefore 24/32 = ¾). The four plaintext elements are all moved one element to the left at the end of each round, and there are eight rounds, which means that each element will be modified twice by all operations in the round.

This mixes the plaintext with the key and itself rotates the first plaintext element D[0] to the left by twenty four (Note: Word length = 32, therefore 24/32 = ¾). The four plaintext elements are all moved one element to the left at the end of each round, and there are eight rounds, which means that each element will be modified twice by all operations in the round.

Key Transformation

The keyed transformation is composed of a sixteen round Feistel network of operations. The function that transforms the values is known as the E-Function , it takes one plaintext element and returns three elements which are mixed in with the rest of the plaintext elements.

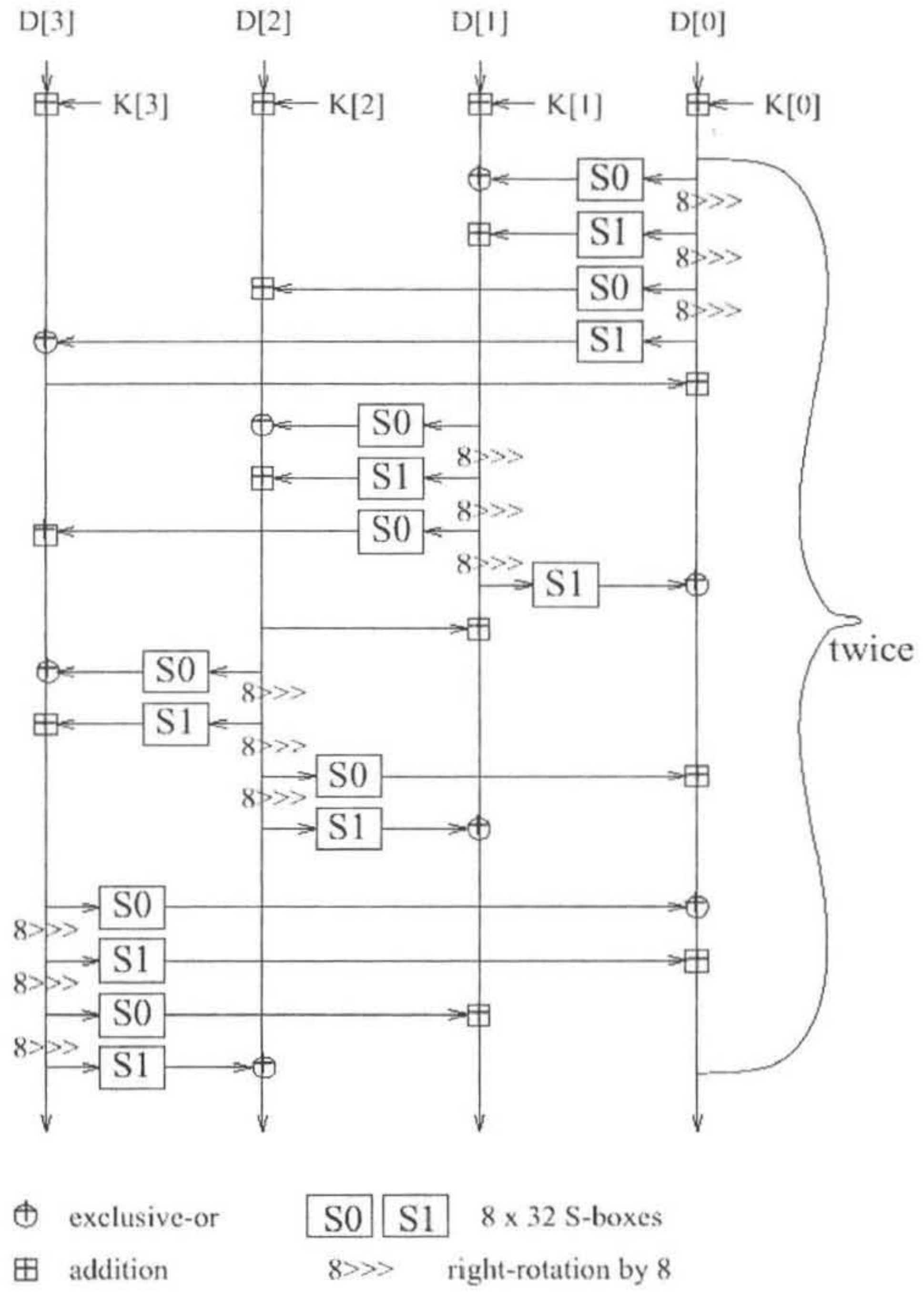


Fig 8.2.1 The Forward Mixing phase [C.Burwick,*et a*,1998]

```
For (i=0 to 15)
        Out[] = E-Function(D[0], K[2*i+4], K[2*i+5])
        D[0] = D[0] <<13
        D[2] = D[2] + Out[2]
        If i=8 then
                        D[1] = D[1]+Out[1]
                        D[3] = D[3]⊕ Out[3]
                Else
                        D[3] = D[3]+Out[1]
                        D[1] = D[1] ⊕ Out[3]
        End if
        (D[0], D[1], D[2], D[3]) = (D[1], D[2], D[3], D[0])
End for
```

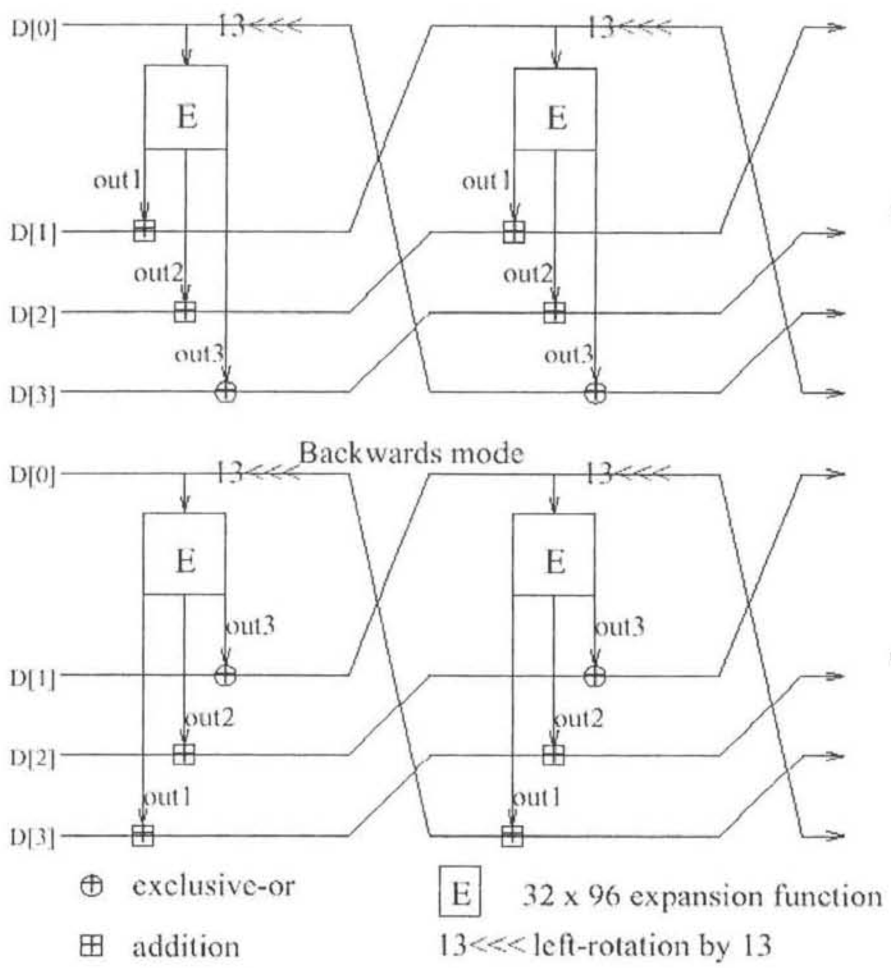


Fig 8.2.2 The Keyed Transformation [C.Burwick,*et a*,1998]

The E-Function accepts D[0] and two keys, it then generates three outputs which are placed in Out[1..3]. These outputs are used in the addition and XORing of the D[1]..D[3] plaintext elements. Once these operations have been performed all the values in D[] are moved one position to the left. This is done for every round and hence each plaintext element will have four occasions in which it will occupy a certain plaintext field.

E-Function

The E function accepts three values, D[0], K[2*i+4] and K[2*i+5], which will be denoted by d,k_1 and k_2 respectively. Three temporary variables are used to manipulate this information A,B and C, which are returned as the output of the E-Function

$A = d + k_1$
$B = (d<<13) * k_2$
i = lowest nine bits of A
C = S[i]
B = B<<5
b = lowest five bits of B
A = A << b
B = B << 5
$C = C \oplus B$
b = lowest five bits of B

C = C << b

A,B and C are a mix of the two keys, the plaintext element and one S-Box element. The initial value of A is equal to the sum of the first key and the plaintext element. B is the value of the plaintext element rotated left thirteen places multiplied by the second key. The nine lowest bits of A are used as the index to the S-Box, which C takes the value of. B is then rotated left by five. A is rotated by the lowest five bits of b. The value of C is the XORed with B, and then shifted to the left by the lowest five bits of B. The important thing to realise here is that the information is being spread over as much of the ciphertext as possible, hiding all the possible statistical data available.

Another important aspect is to realise that these operations have been implemented in such a fashion that they are reversible, and hence decryption possible. When examining the decryption one will see that values are XORed, shifted right instead of left and divided instead of multiplied.

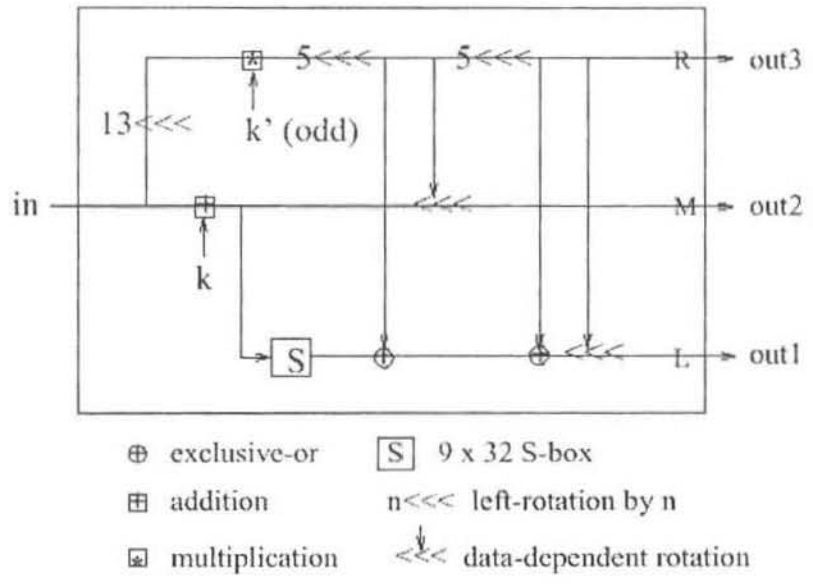


Fig 8.2.3 The E-Function [C.Burwick,*et a*,1998]

Backward Mixing

The backward mixing also performs whitening and its types of operations are very similar to the forward mixing phase.

```
For (i = 0 to 7)
        If i=2 or 6 then D[0] -= D[3]
        If i=3 or 7 then D[0] -= D[1]
        D[1] = D[1] $\oplus$ $S_0$[lowest byte of D[0]]
        D[2] = D[2] - $S_0$[Highest byte of D[0]]
        D[3] = D[3] - $S_1$[Third byte of D[0]]
        D[3] = D[3] $\oplus$ $S_0$[Second byte of D[0]]
        D[0] = D[0] <<24
        (D[0], D[1], D[2], D[3]) = (D[1], D[2], D[3], D[0])
End for
For (i=0 to 3)
        D[i]=D[i]-K[36+I]
End For
```

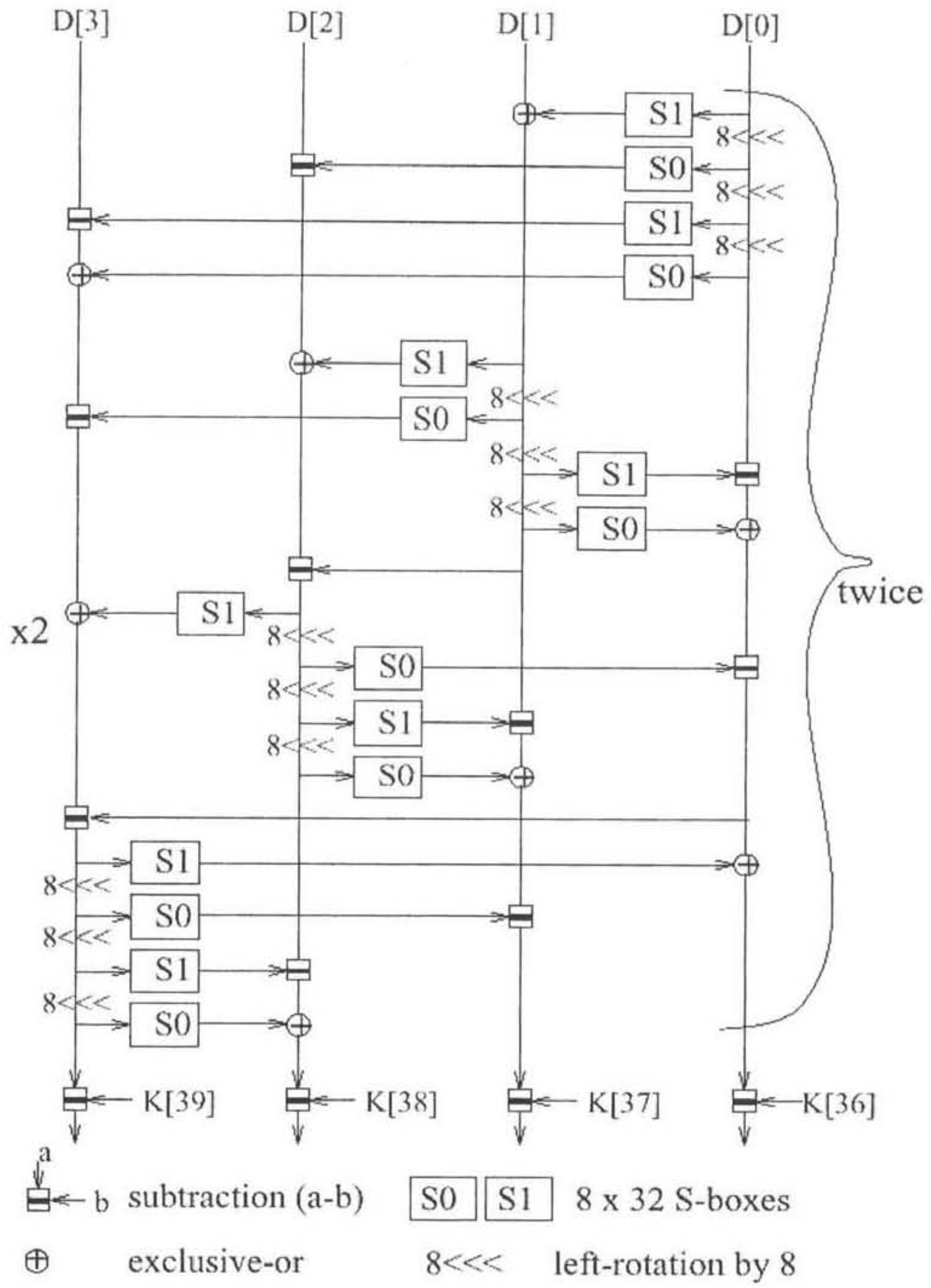


Fig 8.2.4 The Backward Mixing phase [C.Burwick,*et a*,1998]

Each round one of the plaintext elements (D[0]) is used as the source words to modify the three other plaintext elements. The following round the value of D[1] is moved into D[0], and hence it becomes the source word or the current round, each element is the source word on two occasions. On the second and sixth round the source word subtracts the value of D[3]. In the third and seventh rounds the source word subtracts the value of D[1].

Finally a bit of whitening takes place when elements of the key are subtracted from all the plaintext elements. The values remaining in the D[0]..D[4] are now the ciphertext elements.

The Decryption Algorithm

The decryption of MARS is simple to understand if the encryption was understood. As the encryption is made up of rotations, XOR's, additions, subtractions and multiplication's. All of which are reversible operations, it stands to reason that the decryption will be the opposite of the encryption.

There are three different steps, the forward mixing, keyed transformation and backward mixing.

Forward Mixing

The forward mixing takes places first, and it uses the same variables that the encryption algorithm used. D[] is an array of four 32-bit words that contain the ciphertext. The S[] array which contains the S-Box array of five hundred and twelve elements of 32-bit words. The key K[] array has forty elements.

The forward mixing of the decryption reverses the process of the backward mixing in the encryption algorithm and one will notice how similar it is. Additions are replaced by subtractions. XORed values are XORed, and the ciphertext elements are shifted one element to the right in the swapping process. Rotations occur in the opposite direction with the same measurement of rotation. All operations are performed in reverse

```
For (i=0 to 3)
        D[i]=D[i]-K[36+I]
End For

For (i = 7 downto 0)
        (D[0], D[1], D[2], D[3]) =  (D[3], D[0], D[1], D[2])
        D[0] = D[0] >>24
        D[3] = D[3] ⊕ $S_0$[Second byte of D[0]]
        D[3] = D[3] +$S_1$[Third byte of D[0]]
        D[2] = D[2] + $S_0$[Highest byte of D[0]]
        D[1] = D[1] ⊕ $S_0$[lowest byte of D[0]]
        If i=2 or 6 then D[0] += D[3]
        If i=3 or 7 then D[0] += D[1]
End for
```

The Keyed Transformation

The same process takes place in the keyed transformation. It reverses exactly the operation performed by its encryption counterpart. The rounds start at fifteen and finish at zero. The values are first swapped to the left. The value of D[0] is rotated in the opposite direction by the same amount thirteen. Three values are generated again from the E-Function. As the value of D[0] does not change, besides its rotation, its value will be consistent with the encryption algorithm, the exact same E-Function may be used with D[0], and the same key values. This will result in the same values existing in Out[1]..Out[3].

```
For (i=15 to 0)
        (D[0], D[1], D[2], D[3]) =  (D[3], D[0], D[1], D[2])
        D[0] = D[0] >>13
        Out[] = E-Function(D[0], K[2*i+4], K[2*i+5])
        D[2] = D[2] - Out[2]
        If i=8 then
                        D[1] = D[1]-Out[1]
                        D[3] = D[3]$\oplus$ Out[3]
                Else
                        D[3] = D[3]-Out[1]
                        D[1] = D[1] $\oplus$ Out[3]
        End if
End for
```

As one can see, instead on addition, subtraction is used thereby reversing the encryption operation by operation.

Backward Mixing

The backward mixing reverses the process performed by the forward mixing of the encryption algorithm. Once again all operations are performed in the reverse order, reversing the original operations performed on them.

```
For (i = 7 to 0)
        (D[0], D[1], D[2], D[3]) =  (D[3], D[0], D[1], D[2])
        If I=0 or 4 then D[0] -= D[3]
        If I=1 or 5 then D[0] -= D[1]
        D[0] = D[0] <<24
        D[3] = D[3] $\oplus$ $S_1$[Highest byte of D[0]]
        D[2] = D[2] - $S_0$[Third byte of D[0]]
        D[1] = D[1] - $S_1$[Second byte of D[0]]
        D[1] = D[1] $\oplus$ $S_0$[lowest byte of D[0]]
End for

For (i= 0 to 3)
        D[i]= D[i] - K[i]
```

The values in D[0]..D[3] will contain the deciphered plaintext elements.

An Example

The key scheduling begins with a user key
k = { 0, 1, 2, .. ,31}

The initial mixing takes place resulting in the temporary array T[], with the following elements,

T[-7..38] = {164676729, 684261344, -2069205959, -1649577337, 2113903587, -735673503, -915562028, -323995194, 170718370, 1272489713, -1291699165, -1574176379, 775155458, 1093196693, 184593213, 1529051314, 29780847, 1241525645, 274897490, 848037223, -1656480989, -679515193, 740622749, -1228274602, 909075862, 1045363007, 706022039, 189071841, -62578810, 538581743, 1265025929, -1297483056, -1371025234, -823293138, 655918954, -2041138099, 788724001, 1922672416, -230292849, -437142186, -497399764, 474208460, 795471396, -1994711459, -124494573, -1041039443, -198107373, -198107373}

Once the temporary array T has been stirred and placed in the key array K[], one is left with, K[0..39] = {-1344297214, 875956763, -801971149, -1571757423, 179380674, 195762001, -515924257, -2072789416, 1651641163, 1601951000, -1259914813, 1345696749, -1404700122, 949929765, -351704798, 2088320838, -1281311071, -1037466454, -117857471, 1627198274, -677770879, 1632128159, -1702510644, -459911906, 1999764266, 1917876914, 162413934, 212157580, 1571707305, -1260684647, 1227506311, -823164753, 276248922, -187899174, 1917877738, -614471195, -1205950396, -1093647705, -954722349, 949280097}

Now one must fix the weak keys, checking elements 5,7,...,35 in the key array K.

K[5,7,..35] = {195762001, -2072789416, 1601951000, 1345696749, 949929765, 2088320838, -1037466454, 1627198274, 1632128159, -459911906, 1917876914, 212157580, -1260684647, -823164753, -187899174, -614471195
}

One can see that there were no elements that are considered weak in the current key, and hence none were changed.

The final key then is as stated above in K[0..39].

Encryption

Now to encrypt the message "Hello World!", first one converts it to integer values, this is done using the CryptLib method ConvertAnyString(String), that converts the string to bytes and then to integers.

Hello World!
1819043144 1867980911 560229490 0

When the values are passed into the encryption algorithm they are placed in the D[] array where,

D[0]=1819043144, D[1]=1867980911, D[2]=560229490, D[3]=0

The first thing that is done is the addition of the keys D[i] = D[i]+K[i], i=0..3

D[0..3] = {474745930, -1551029622, -241741659, -1571757423}

The forward mixing takes place for eight rounds,

Round 0
D[0..3] = {1646355905, -1656557383, 742058680, 2017932500}
Round 1
D[0..3] = {201595491, 1721471700, 1332004190, 761912261}
"
"
Round 6
D[0..3] = {55044653, 88209549, 535438619, -267296678}
Round 7
D[0..3]= {1572015899, 792603926, -1466545613, 1206529283}

Now that the forward mixing is completed the Keyed transformation takes place over sixteen rounds.

Round 0
Out[1..3] ={167695470, 210115501, -776316752}
D[0..3] = {960299396, -1256430112, -1773111885, 1642294198}
Round 1
Out [1..3] ={605507730, 833079814, 161325982}
D[0..3]={-650922382, -940032071, 1753139240, -1607432409}
"
"
Round 14
Out [1..3] ={754136283, -1045599012, -73306089}
D[0..3]={-1637864550, -296466158, 1415363190, -965716442}
Round 15
Out[1..3] ={-1348234262, 711001132, -1970933296}
D[0..3]={1691465922, 2126364322, 1981016592, 91444172}

The backward mixing takes place for eight rounds resulting in the following values in D[].

Round 0
D[0..3]={89016844, 1373058604, 901405797, -1033580104}
Round 1
D[0..3]={573480788, 1637079300, 1259736458, 201674314}
"
"
Round 6
D[0..3]={266242868, 711977859, -937129531, -316107467}
Round 7
D[0..3]={-1340437116, -2063161365, 1469176171, -1310364001}

Once the final whitening has taken place, with D[i]=D[i] - K[36+i], i=0..3 the final ciphertext is

D[0..3] = {-134486720, -969513660, -1871068776, 2035323198}

Decryption

The first thing that takes place is the forward mixing, this reverses the process of the backwards mixing in the encryption algorithm. The values of the D[] array is as follows

D[0..3] = {-134486720, -969513660, -1871068776, 2035323198}

The whitening then is reversed, D[i]=D[i] + K[36+i], i=0..3

D[0..3] = {-1340437116, -2063161365, 1469176171, -1310364001}

The forward mixing then proceeds for eight rounds

Round 7
D[0..3] = {266242868, 711977859, -937129531, -316107467}
Round 6
D[0..3] = {417439973, -1065722789, -1419324492, -263427336}

"

"

Round 1
D[0..3] = {89016844, 1373058604, 901405797, -1033580104}
Round 0
D[0..3] = {1691465922, 2126364322, 1981016592, 91444172}

The Keyed transformation then proceeds
Round 15
Out[1..3] = {-1348234262, 711001132, -1970933296}
D[0..3] = {-1637864550, -296466158, 1415363190, -965716442}
Round 14
Out [1..3] ={754136283, -1045599012, -73306089}
D[0..3]={-1637864550, -296466158, 1415363190, -965716442}
"
"
Round 1
Out [1..3] ={605507730, 833079814, 161325982}
D[0..3]={-650922382, -940032071, 1753139240, -1607432409}
Round 0
Out[1..3] ={167695470, 210115501, -776316752}
D[0..3] = {960299396, -1256430112, -1773111885, 1642294198}

Once the transformation is complete the backward mixing takes place, which is exactly the reverse of the forward mixing of the encryption algorithm. There are eight rounds in the mixing,

Round 7
D[0..3] ={55044653, 88209549, 535438619, -267296678}
Round 6
D[0..3]={1525682528, 592963985, 2013255885, -505864879}
"
"
Round 1
D[0..3] = {1646355905, -1656557383, 742058680, 2017932500}
Round 0
D[0..3] = {474745930, -1551029622, -241741659, -1571757423}
The last operation that takes place is D[i] = D[i]-K[i], i=0..3, resulting in the correct plaintext elements in D[], which is

D[0]=1819043144, D[1]=1867980911, D[2]=560229490, D[3]=0

"Hello World!"

8.3) MARS Implementation

Due to the non-disclosure agreement that was signed with NIST, I cannot display any of the actual code for the algorithm. However I can discuss pseudo code for the MARS algorithm.

The Class MARSCrypt.Java has a number of predefined variables, these are the S-Key array and the user key array. All of these are stored as arrays of integers. The

S-Box is a predefined set of data values and has been recommended, there are five hundred and twelve entries in the S[] array.

The MARSCrypt initialise method performs the following:

```
//RC6Crypt.Java RC6Crypt method
CL = New CryptLib
User_key[32] = 0
S = generate_Keys(User_key,32)
```

A new object of the CryptLib class is created, as a few of the methods are used by the encryption and decryption functions. A default user key is created which is set to zero. This enables a default S-key array to be created, when generate_Keys is called. There are two encryption and decryption functions, and the method for generating the keys.

Generate_Keys(userKey, size)

```
n = Size/4
integer B[4],T[47],K[40]
K[] = CryptLib.getInteger(UserKey)

For (loop=0;loop<8;loop++)
  T[loop]= S[loop]
End For
```

The user key is converted from bytes to words and placed in K[]. The temporary array is then initiated with the first seven values of the S-Box. To makes things easier to control, the array index of T[] goes from 0..46, instead of -7..38, as it is easier to work with.

```
For (loop=0;loop<40;loop++)
        w = (T[loop] XOR T[loop+5]);
        T[loop+7] = ((w>>>29) OR (w<<3)) XOR K[loop mod n] XOR loop;
End For
T[46] = n;
```

The initial mixing of the key and the temp array take place. One will notice that two rotations are executed instead of one, this is so one will have a data independent rotation and no timing information can be obtained.

```
For (rounds = 0 ;rounds <7;rounds++) {
     For (loop=1;loop<40;loop++) {
              T[7+loop] = T[7+loop] + S[Low bits of T[7+loop-1]];
              T[7+loop] = (T[7+loop] << 9)| (T[7+loop] >>23);
    End For
         T[7] = T[7]+S[low Bits of T[46]];
         T[7] = (T[7] << 9) OR (T[7] >>23);
End For
```

The seven rounds of mixing or stirring then occur as the information of the S-Box is then used. Once again the same methodology is used in the rotation operations. One will notice that all the index's have the value of seven added to them. This is because the array index starts at zero instead of negative seven and the same algorithm is used for mixing so one needs to offset these indexes. (As opposed to working with negative indexed array elements)

```
For (loop=0;loop<40;loop++) {
        K[(7*loop) Mod 40] = T[loop+7];
End For
```

Once the Key values have been set they are then checked for weak subkeys.

```
For (loop = 5;loop<36;loop+=2) {
                K[loop] = fix_subkey(K[loop],K[loop+3]);
        }
```

The function fix_subkey returns a corrected sub key, all it does is check for consecutive bits. If there are none, no changes take place. If there are refer back to the algorithm discussion to see what is performed. The value of K is returned as the key of the algorithm with forty entries.

MARSEncrypt(plaintext String, userKey[])

There are two versions of the encryption, one accepts a string and the other four integer values in an array. This occurrence deals with splitting up the strings into encryptable values.

```
Tempint[] = CryptLib.ConvertAnyString(Plaintext)
For (loop=0;loop<TempInt.Length;loop+4)
        BlockPlaintext[4] = Tempint[loop,loop+1,loop+2,loop+3]
        BlockPlaintext[4] = MARSEncrypt(BlockPlaintext[],use key)
        Ciphertext += BlockPlaintext
End for
```

The current method accepts the plaintext, and converts it to an array of integers, which is of uniform length that is divisible by four. The ConvertAnyInteger pads the end with zeros if it is not long enough. Then the code copies four elements of the plaintext integers into an array of four elements. These are the four elements that are then encrypted using the other MARSEncrypt method, passing the array of four plaintext integers and the user key. The result of this is then added to a string called ciphertext. The loop continues looping until all elements in the plaintext integer array Tempint have been encrypted. The ciphertext string is returned.

MARSEncrypt(plaintext [],Key[])

The algorithm takes the four plaintext words and places them in the array D[], and adds similar positioned keys elements to them. This is the whitening stage.

```
For (loop=0;loop<4;loop++)
        D[loop]= Plaintext[loop]+Key[loop]
End For
```

The forward mixing then takes place,

```
For (loop=0;loop<8;loop++)
        D[1] = D[1] ⊕ S0[low byte of D[0]]
        D[1] = D[1] + S1[Second byte of D[0]]
        D[2] = D[2] + S0[Third byte of D[0]]
        D[3] = D[3] ⊕ S1[Highest byte of D[0]]
        D[0] = (D[0]>>24) OR (D[0]<<8)
        If loop = 0 or 4 then D[0]+=D[3]
        If loop = 1 or 5 then D[0]+=D[1]
D[] = SwapValuesLeft(D)
```

The values are XORed, and added to one another, and once again the same rotation methodology is used to prevent timing attacks. The values in D are then shifted one position to the left in the array D when the SwapValuesLeft method is executed. The keyed transformation step now proceeds.

```
For (loop=0;loop<15;loop++)
        Out = Efunction(D[0],K[2*loop+4,2*loop+5])
        D[0]= (D[0]<<13) or (D[0]>>19)
        D[2]+=Out[1]
        If loop = 8     Then    D[1]+=Out[0] , D[3] XOR=Out[2]
                        Else    D[3]+=Out[0] , D[1] XOR=Out[2]
        D[] = SwapValuesLeft(D)
End For
```

The final step which is backward mixing then takes place.

```
For (loop=0;loop<8;loop++)
        If loop=2 or 6 then D[0]-=D[3]
        If loop=3 or 7 then D[0]-=D[1]
        D[1] = D[1] XOR S1[low byte of D[0]]
        D[2] = D[2] - S0[Highest byte of D[0]]
        D[3] = D[3] – S1[Third byte of D[0]]
        D[3] = D[3] XOR S0[Second byte of D[0]]
        D[0] = (D[0]>>24) or (D[0]<<8)
        D[] = SwapValuesLeft(D)
For (loop=0;loop<4;loop++)
        D[loop]-= Key[36+loop]
End For
```

The values in D[] are returned as the ciphertext as an array of integers.

EFunction(d,K1,K2)

The E-Function transforms the values in using a Feistel network.

```
M = d+K1
R = ((d<<13) or (d>>19)) * k2
L = S[i = lowest 9 bits of M]
R = (R<<5) or (R>>27)
M = (M<<(lowest 5 bits of R)) or ((M>>-1*(lowest 5 bits of R))
L = L XOR R
R = (R<<5) or (R>>27)
L = L XOR R
L =  (L<<(lowest 5 bits of R)) or ((L>>-1*(lowest 5 bits of R))
Return L,M,R
```

Decrypt(cipherText String, key[])

The string is converted to a uniform length integer array (Divisible by four). A loop is then executed taking four consecutive integers at a time and calling the MARSDecrypt(Ciphertext[] integer, Key) method. The results are then appended to a string called plaintext.

```
Tempint[] = CryptLib.ConvertAnyString(CipherText)
For (loop=0;loop<TempInt.Length;loop+4)
        BlockCipherText[4] = Tempint[loop,loop+1,loop+2,loop+3]
        BlockCipherText [4] = MARSDecrypt(BlockCipherText[],use key)
        Plaintext+= BlockCipherText
End for
```

The results are then returned in the plaintext string variable.

Decrypt(cipherText[],key[])

This algorithm takes an array of four words and then places them in the array D[], such that

```
For (loop=0;loop<4;loop++)
        D[loop]+= Key[36+loop]
End For
```

The values of D[] are now ready for the forward mixing stage of the decryption algorithm.

```
For (loop=7;loop>-1;loop--)
        D[] = SwapValuesRight(D)
        D[0] = (D[0<<24) or (D[0>>8)
        D[3] = D[3] XOR S0[Second byte of D[0]]
        D[3] = D[3] + S1[Third byte of D[0]]
        D[2] = D[2] + S0[Highest byte of D[0]]
        D[1] = D[1] XOR S1[low byte of D[0]]
        If loop=2 or 6 then D[0]+-=D[3]
        If loop=3 or 7 then D[0]+-=D[1]
End For
```

The forward mixing operations of the decryption algorithm reverses the process performed by the backward mixing operations in the encryption algorithm. The values of D[] are all moved one index position to the right, with wrap around. This is done the function SwapValuesRight().

The keyed transformation operations now proceeds,

```
For (loop=15;loop>-1;loop--)
        D[] = SwapValuesRight(D)
        D[0]= (D[0>>13) or (D[0]<<19)
        Out = Efunction(D[0],K[2*loop+4,2*loop+5])
        D[2]-=Out[1]
        If loop = 8     Then    D[1]-=Out[0] , D[3] XOR=Out[2]
                        Else    D[3]-=Out[0] , D[1] XOR=Out[2]
End For
```

The keyed transformation reverses the operations performed by its counterpart in the encryption algorithm. The backward mixing then takes place,

```
For (loop=7;loop>-1;loop--)
        D[] = SwapValuesRight(D)
        If loop = 0 or 4 then D[0]-=D[3]
        If loop = 1 or 5 then D[0]-=D[1]
        D[0] = (D[0]<<24) or (D[0]>>8)
        D[3] = D[3] XOR S1[Highest byte of D[0]]
        D[2] = D[2] - S0[Third byte of D[0]]
        D[1] = D[1] - S1[Second byte of D[0]]
        D[1] = D[1] XOR S0[low byte of D[0]]
End For
```

Once this has been completed the final removal of whitening takes place

```
For (loop=0;loop<4;loop++)
        D[loop]-=Key[loop]
End For
```

The plaintext elements are then returned in the integer array D[].

8.4) MARS Usage

The client and server applications have a copy of the MARSCrypt class. These are initialised on the initialisation of the CryptLib class. As has been mentioned before, the client generates the user-defined key. The client sends this key, first encrypting it with RSA to the server. The server then decrypts the RSA message and obtains the key. The key is specific to each client, and therefore the copy of the key is kept in each clienthandler class. Therefore each client has their own session key which will be different from any other client key. Once the key is received it is then given to the generate_Keys method which execute the key scheduler and which will create the correct key for encryption and decryption algorithm.

As the Cryptlib class executes the correct key scheduler method. It generates the key and places it in the S array. The client and the server perform this key schedule method and hence they will have the same key for the current client's session. Once the key has been generated, and there are no problems, the encryption then changes from RSA to the specified encryption algorithm (in this case its MARS).

```
//ClientHandler.Java - send method
...
        If (encrypt_mode>0)
                        CurrentMesg= CL.Encrypt(CurrentMesg,Key)
        EndIf
...
```

Thereafter whenever the SendMsg method is called by the client, and Send method by the server, the messages are first passed to the Cl.Encrypt() method. This method passes the message to the MARSEncrypt(String,UserKey) method, which encrypts the message and returns a set of numbers in a string format. This is the ciphertext, and it is the sent to the recipient of the message.

```
//ClientHandler.Java - run method
...
        CurrentMesg = ReadStream
        If (encrypt_mode>0)
                        CurrentMesg= CL.Decrypt(CurrentMesg,Key)
        EndIf
...
```

When the client or server receives a message, it is first passed to the CL.Decrypt method which passes it to the correct decryption method (MARSDecrypt(String, UserKey)), this decrypts the message, and then the message is dealt with. If any errors occur while decrypting messages they are ignored, and the error is reported to the administrator. Both the MARS and RC6 algorithms have been implemented in such a way that they have the same methods and functions to maintain standard specifications of the implementation.

8.5) Timing

Speed tables re used to compare the performance of the algorithm to other implementations. The performance shows MARS to be relatively quick, and it has been shown to be even more so in a hardware implementation of the algorithm.

Language	Scheme	Blocks/Sec	Mbits/Sec
Auction MARS			
JDK 1.2	Encryption	20000	2.56
JDK 1.2	Decryption	20000	2.56

Fig 8.5.1 Encryption and Decryption figures of MARS

Language	Scheme	Blocks/Sec	Mbits/Sec
Optimised Implementation			
JDK 1.1.6	Encryption	?	14.5
JDK 1.1.6	Decryption	?	17.3
Borland C++	Encryption	?	28
Borland C++	Decryption	?	28

Fig 8.5.2 Encryption and Decryption figures of MARS [C.Burwick, *et al*]

The auction implementation of MARS was completed on an Intel PII 450Mhz chip with 196MB Ram, running MS Windows 98. A few optimizations were implemented in the algorithm, but for readability's sake it is as close to the pseudo code as possible. As it can be seen the MARS speeds, it is slightly faster than RC6. This is due mainly to the compilers and CPU's used, as discussed in [B.Schneier, *et al*], and MARS is actually considered a slower algorithm as it requires more operations. It can be seen that the auction applications implementation is about five times slower than the optimized version of the code.

The figures shown here for the optimized implementations were obtained using an Intel Pentium Pro 200MHz chip with 64MB Ram, running in MS Windows 98. As it has been mentioned before, the Pentium Pro chipset is better suited for the operations that are performed by the algorithms. This and the fact that the algorithms are optimized, result in the higher speeds. The main result of these figures is to show that the current implementation of the MARS algorithm is able to support a large number of simultaneous clients.

As the algorithm is encrypting or decrypting over two and a half megabits a second, it equates to a large capacity of messages that may be handled. This is more than adequate for the auction application as it could deal with over a thousand clients a second.

8.6) Attacks

There are a number of attacks that the MARS algorithm may be susceptible to, but it is still a rather new algorithm and has shown good resistance to the classical attacks. There are a few different roots of attacking the algorithm, and this includes the key schedule, the S-Box, Timing attacks and linear and differential attacks.

The S-Box attack

As the S-Box is a known value and is not random, it means that it may be extensively examined. The developers of MARS, [C.Burwick, et al] relate this attack, based on the assumption there is a weakness in the S-Box.

Assume that one of the S-Box halves S_0 or S_1, has two elements where

$\Delta = S_1 [i] \text{ XOR } S_1 [j] = 0$ in the highest bits

This weakness will mean that the key and the encryption is at risk, as the high order bits in multiplication, addition and subtraction will have greater significance in these operations. This means that when one performs encryption the change in the elements will not be as great as expected, and hence a part of the plaintext may be revealed during the encryption process. One may be able to determine this information with weaknesses in the S-Box.

There are a few other attacks that could be mounted on an S-Box, but the specified S-Box has been chosen such that the information is relatively random and all elements differ by a number of bits. This is to prevent all the well-known attacks mounted on DES's S-Box. The inventors of the algorithm showed that there are characteristics that hold in the S-Box which occur with a probability of more than 2^{-32}. This means it's an unlikely source of attack, as the S-Box is about as random as needs be, and that an attack on the S-Box will probably be the easiest.

The Key Schedule Attack

According to the creators of MARS, there are no weak or equivalent keys. The keys used for multiplication's are checked for any form of weaknesses and are fixed if any problems do arise. It is also suggested that the key schedule or expansion is random enough to produce very expanded keys even though the user keys have similar characteristics. The expanded key has about 21248 effective bits. It has been shown that the pairs of original n bit keys mapped to the same expanded key are about $2^{2n - 1249}$, a rather large value for the normal 128 bit key.

The reason that an attack on the key expansion is unlikely is because some of the operations are not easily reversible; the most important one being the checking of the multiplication keys. This however is a problem as this method may actually cause a collision, in which the user key and expanded keys are the same. All this would require at least a searching of 2^{32} different keys [M-J.O.Saarinen] for one to find the original key. This is assuming one is able to reverse the key fixing or checking step, and reverse all of the other steps. If one were to attempt this it would not take too long for a standard computer to search the 2^{32} keys. The reason why it will not fail in the case of the auction application is that the expanded or user key is never seen, due to the fact that it is encrypted by RSA. If RSA failed the entire protocol would fail as one of its primitives was flawed.

Linear and Differential Attacks

"In linear analysis one tries to find a subset of the bit positions in the plaintext, ciphertext and expanded keys, so that for a uniformly chosen plaintext and expanded key, the probability that the sum of the bits in these positions is equal to zero modulo two, will be bounded away from one half" [C.Burwick, *et al*]. The difference between the probability and one half is known as a bias. The bigger the bias the better as it requires less guessing of values, and hence less of a workload.

To mount a linear attack the algorithm is broken down into its constituent operations, and each of these operations is given an approximation. All these approximations are combined together to find the global approximation of the complex operation or algorithm. Once this has been completed one can then attempt to approximate bits and bit values within the algorithm for each round. As the end result is known (the ciphertext), one works backwards giving the approximations of bits for each round, until one can guess or approximate the plaintext. The problem however is that these approximations have shown that the linear approximation of the algorithm has a bias of 2^{-69}, which is very small. This means that the complexity of the problem is of the order 2^{128}, which is incredibly difficult to solve.

Many aspects of the algorithm, such as the S-Box, and E-Function which maintain the security of the algorithm have been designed to minimize linear attacks. For instance the E-Function which transforms the data has no operations or values totally dependant on one another, making it difficult to perform approximations.

[C. Burwick, *et al*] have shown a number of features of differential cryptanalysis which would make breaking the algorithm very difficult. One is an examination of the Key transformation characteristics of the algorithm. To construct such characteristics using the expanded key and plaintext has a probability of 2^{-240}. This is assuming one can obtain enough ciphertext, plaintext and keys to perform this analysis (2^{128} pairs, which is almost impossible to obtain). This is just for the keyed transformation and does not deal with any of the other operations.

Once again it, as has been mentioned before it is not feasible in any terms to mount these attacks on the practical implementation of the auction application. The rewards are far to slim to make it anywhere near a profitable exercise.

Timing Attack

As the algorithm uses data dependent rotations, which have been shown to execute at different speeds depending on the rotational value, it is considered a weakness.

One could possibly time the rotations and then work out the important values in each round of the encryption, thereby giving away information of the plaintext.

To resolve this problem the same methodology that was used for RC6 has been used for MARS. This is the inclusion of a second rotation in the opposite direction. The rotational value is the negative rotational value modulus the word length. This two rotations are then ORed together to get the final result. This double rotation results in a constant time for rotation and hence no timing information may be concluded.

Brute Force Attack

The brute force attack will attempt every set of keys until the correct plaintext is determined. As the length of the user key is unknown (128 – 1248 bits). The security of the algorithm grows quite extensively as the size of the key gets to 256 bits. Once the size of the key goes over 300 bits one will not essentially benefit from much more security.

It has been suggested that the security of the algorithm is 2^n, where n is the number of bits used in the key (with $n<256$). It is not even easier to guess the expanded key, as the S-Box would have played a role in the generation and it will conform to a whole lot of rules. Using this as a general guideline once again brings us to the conclusion that this attack is just not feasible in fiscal and time constraints.

8.7) Conclusion

The last three chapters have examined the encryption algorithms used in the auction applications protocol and they are the primitives of this protocol. The protocol as a whole could not work without one other, both the symmetric and asymmetric systems need one another as has been explained. The algorithms known to be relatively secure, barring any new advances in technology or cryptanalysis.

The reason that the overall protocol is considered relatively secure is really because of the way in which it is used. It has been stressed repeatedly that the even if the protocol is breakable the resources currently available (money, time, expertise and information) would not be sufficient to meet the requirements of an attack. Even if they were, the rewards for breaking the system would not be sufficient. The correct implementation of the algorithms has ensured the security of the overall project

9) Conclusion

The auctioning application has the met criteria laid out at the beginning of the thesis in number of ways.

The use of Java as the development language has had dual benefits. Java has enabled the creation of a system independent application, which does not require any specific operating system or hardware. Java's intrinsic security support has allowed the development of a relatively secure application that poses no threat for the client or server.

The auctioning application also had a dual purpose. The application was a vessel for the implementation and usage of secure communication. The auctioning application turned out to be viable, efficient, practical and even meets a certain need in the industry.

The Three algorithms RSA, RC6 and MARS handled secure communication. RSA is a proven public key encryption algorithm that meets the requirements for secure key exchange between the client and the server. MARS and RC6 are relatively new algorithms that are based on sound theory which have shown to be more than adequate in meeting the requirements of secure communication. The algorithms have been used in such a way that their weaknesses are not exposed. The protocol as a whole, proves to be relatively secure and currently is not susceptible to any feasible attacks. However this does not guarantee that the system will remain secure as new attacks and weaknesses are devised very rapidly.

The application pulls together as a whole, using as little bandwidth as possible, thereby benefiting the clients, and minimizing server resource usage. The result is then: "A Real Time, System Independent, Secure, Internet Based Auctioning System".

10) Installation and Usage

The installation of the application is rather straightforward. The are a number of requirements though:

Requirements

- TCP/IP connection
- Web Server with ASP (VBScript) for Win32 or CGI (PERL 5+) for Unix
- Java 1.2 or above
- 1MB Free Space

The installation packages have been divided into two types Win32 and Unix. The files are named win32aucserv.zip and unixaucserv.tar, respectively. The only reason that there are two different files is that the CGI scripting used to pass parameters from the HTML documents to the applet and vice versa is different for the two operating systems. On Win32 system, VBScript through ASP is used, and on the Unix system, PERL through CGI is used. The class files are exactly the same.

Files Included

The files included in both zip files are:

Aucserv.jar
HtmlHelp Directory
Logs Directory
Images Directory
Html Directory
Docs Directory

The aucserv.jar file contains all the class files for the auction server. The HtmlHelp directory contains the Html Help pages for the Auction Server application. The Logs directory contains empty log files. The Images directory contains the image icons used by the Auction Server application. The Html directory contains a default Html site and a few required files. For a more comprehensive discussion see the "Auction Implementation" chapter. The Win32aucserv.zip file has a run.bat file used for executing the application.

9.1) Installation & Set-up

Extract the zip file by using a Zip utility. On Win32 systems use pkunzip or Winzip, and in Unix use Tar. When unzipping be sure to include all directory structures and subdirectories.

Win32
pkunzip –d win32aucserv.zip

Allow the html directory to be executable.

Unix
tar –zvf unixaucserv.tar

Set up a symbolic link from the web root to the html directory.
Ln –s <Installed Directory> <Link name>
The web site will then be available at http://hostname/<link name>

Ie: ln –s /home/auction/public_html /httpd/html/auction
http://hostname/auction

One needs to configure the web server. In the /etc/httpd/conf/access.conf file add the following:

```
<Directory /home/httpd/html/auction
AllowOveride No
Options ExecCGI Include
</Directory>
```

In the /etc/httpd/conf/srm.conf file add the following:

```
AddHandler cgi_script .cgi
AddHandler Server_parsed .htm
```

Allow the files in the html directory to be executable
chmod +x *.*

9.2) Executing the Server

In the aucserver directory execute the following

Win32

run.bat

Or

java –jar aucserv.jar

Unix

java –jar aucserv.jar

9.3) Application Maintenance

The source is included in AucServersrc.zip and AucClientsrc.zip. Within each file is the source code for the specified application. All source files are included, except for the RC6Crypt and MARSCrypt source files as I cannot reveal the source code for the applications due to the AES non-disclosure agreement. However the class files were included so that the application may still execute.

9.4) Server Usage

Once the application has started it does not automatically start the server.

Note: Unavailable menu items (Greyed out) indicate that other options or features need to be addressed before one may proceed.

Starting the Server

To start the server, select the Server Menu | Start Server

Selecting the Port number

If the server fails because the port the server has attempted to start on (Default 3002), one must change the port number. Once the port number has been changed one must then restart the server (see above).

To change the port number, select Server Menu | Port Number. Type a port value between 1000 and 65000, and then click on OK.

Exiting the Application

To exit the application there are two options

Select the close option on the Window.

Or

Select the Serve Menu | Exit.

Once the server has been started the auctions may then be configured.

Entering Item details

To enter the item details:

Select Auctions Menu | Item Details

Select the amount of items to be auction and click OK. One is then asked to enter the item details one by one in the Auction Item Details frame. The frame tells one what the number in the auction list the item is, and it requests the following details: the auction item ID number, the minimum starting price and a brief description. Once all the details for the current item have been entered click OK. Repeat until all details for all items have been entered. If any errors are made the application will notify one about the errors.

Note: If one wants a picture of the item to be sent to the client, during the auction, one must place the image in the Images directory off the html directory. The image must have the name (in lower case) of the item ID number, and it must be a GIF image.

The details entered here are written to the file auchtm.inc file, which is included in an HTML file, so the clients may see what being auctioned. If one wants to enter more details about the items auctioned this will have to be done manually on the web site. Once the details have been entered one can then set the stop time for the current auction. Auctions start with the first item entered and finishes with the last.

Entering Stop Time

When choosing the stop time it is for the current item auctioned only. To extend the current auction change the finish at any stage throughout the auction. All users will be notified about this change. When the auction finishes for the current item one may then set the following items stop time.

Select the Auctions Menu | Stop Time. Then choose the correct year, date and time to finish.

If no finish time is entered and the auction is started it will finish immediately and proceed to the next auctioned item.

Starting an Auction

Once all the above sections have been dealt with the auction may be started. If any of the previous steps have been missed this menu option will not be available.

Select Auctions Menu | Start Auction

Or

Select the Play button on the Menu Bar.

The auction will start and all details pertaining to the auction will be displayed in the status bar and on the information panel. To see information about the users select the Users panel (See below).

Stopping an Auction

When one stops an auction, the clients will be notified and the next item will be prepared for auctioning. This will forcefully stop the auction at the present time. If the auction is left to its own devices it will only stop at the pre-determined time.

Select the Auction Menu | Stop Auction

Or

Click on the Stop button on the Menu Bar

Demo Auction

If one does not want to go through the entire rigmarole of setting up an auction one can run the Demo auction. This auctions four items with random stop times and minimum values.

Select Auction Menu | Demo Auction

Or

Click on the D in the Menu Bar

Selecting Key Length

One may change the RSA key length from its default (128 bit) to a number of different values up to and including 4096 bit keys. However one should be aware that large keys take longer to generate.

Select Security Menu | Key Length

This changes the key length but does not generate new keys, this need to be done manually.

Regenerating Keys

One may manually force the application to generate new keys.

Select the Security Menu | Generate Keys

Selecting Symmetric Cryptosystem

One has a choice of two symmetric key systems. This may be changed before and auction starts, and thereafter it will remain fixed until the auction has ended. Only once the auction has ended may one select another cryptosystem.

Select Security Menu | Cryptosystem of choice (RC6 / MARS)

Help

To get online help.

Select the Help Menu | Help

This brings up another frame, which is a web browser. There are two buttons: the contents page, which will return one to the default help page, and the close help button which will close the help page. The application will continue running even though one may be perusing the help pages.

Information

There are various bits of information that is displayed in the auction application and it is displayed in two panels. The Information panel and the User panel.

Information Panel

The information panel displays information pertaining to the server, current auction and any problems. One may get information about the current auction by clicking on the details menu bar button. If the screen becomes too cluttered one may click on the Clear button, which will remove all information from the information panel.

User Panel

The user panel displays a list of the current users connected to the application. By selecting a client it will display their details in the client information box, The current top bid is displayed in the bottom left hand corner.

9.5) Client Usage

The client selects the item to bid for from the web page. It will take one to the page containing the applet. The applet execute and asks the client to type thirty random characters. These are used as the key for the encryption process.

Once this has been done the client needs to full in all relevant data (Name, Email, etc). Once this has been completed, click on the connect button to connect to the auction server. If there are no problems the applet displays the bidding page.

The bidding page only has one action available. This is simply to bid. The client enters a bid value which must be greater that the minimum bid or current top bid and clicks on the submit button to send it. This page relates all the details to the client about the current auction.

11) References

Aura, "Strategies against Replay Attacks", IEEE Computer Society, 1997

Bain, K. & Engelhardt,M. 1991 "Introduction to probability and Mathematical statistics", ISBN 0-534-92930-3

Bleichenbacher, D. & Joyle ,M. & Quisquater, J.J. "A New and Optimal Chosen-message Attack on RSA-type Cryptosystems"

Blaze,M. & Diffie,W. & Rivest,R. & Schneier,B. & Shimomura,T. & Thompson,E. & Weiner,M. 1996 "Minimal Key Lengths for Symmetric Ciphers to Provide Adequate Commercial Security

Bellare,M., 1998." Practice -Orientated Provable Security", Proceedings of First International Workshop on Information Security (ISW 97), Lecture Notes in Computer Science Vol. 1396, E. Okamoto, G. Davida and M. Mambo eds., Springer Verlag

Bellare,M. and Rogaway,P.,1995,"The complexity of approximating a nonlinear program; Journal of Mathematical Programming B, Vol. 69, No. 3, pp. 429-441

Bellovin, S.M., 1996," Problem Areas for the IP security Protocols, Proceedings of the Sixth Usenix Unix Security Symposium, pp. 1-16, San Jose,CA.

Bellovin, S.M.,1998,"Cryptography and the Internet", Proceedings of CRYPTO '98, , pp. 46-55.

Boneh, D., 1999, "Twenty years of Attacks on the RSA cryptosystem", j-NAMS 46 n. 2, pp. 203-213.

Boneh, D. & Durfee,G. ,1999, "New Results on the Cryptanalysis of Low Exponent RSA", submitted to Eurocrypt '99.

Boneh, D. & Venkatesan, R.,1998, "Breaking RSA may be easier than factoring", Proceedings Eurocrypt '98, Lecture Notes in Computer Science, Vol. 1233, Springer-Verlag, pp. 59-71.

Brickell, E.F & Odlyzko, A. M., 1988,"Contemporary Cryptology, Cryptanalysis: A Survey of Recent Results", chapter 10, IEEE Press, pp. 501-540. Preliminary version Proc. IEEE 76, pp. 578-593.

Burwick, C. & Copersmith, D. & D'Avignon, E. & Gennaro, R. & Halevi, S. & Jutla, C. & S.M. Matyas Jr., S.M. & O'Connor, L. & Peyravian, M. & Safford, D. & Zunic, N. ,1998, "MARS - A candidate for AES", IBM, First AES conference.

Koffman,Elliot B., Quote - Pg186,Turbo Pascal / Elliot Bo Koffman with Bruce R. Maxim. - 4th Edition, ISBN 0-201-558811-4 , Addison-Wesley , 1993

Franklin, M. & Wright, R.N, 1998,"Secure Communication in Minimal Connectivity Models", Eurocrypt 98

Gilbert, H. & Gupta, D. & Odlyzlo, A.M & Quisquater, J.J, 1998, " Attacks on Shamir's 'RSA for paranoinds'". Information Processing Letters (1998)

Gennaro, R. & Krawczyk, H. & Rabin, T., "RSA-Based Undeniable Signatures", Proceedings of CRYPTO'97.

Goldreich, O. & Micali, S., 1984, "Increasing the Expansion of Pseudorandom Generators".

Halevi, S. & Krawczyk, H., 1998, "Public-key cryptography and password protocols", , Proceedings of the Fifth ACM Conference on Computer and Communications Security, to appear

Handschuh, H. & H. Heys, H., 1998, "A Timing Attack on RC5", Workshop on Selected Areas in Cryptography, Springer-Verlag.

Hughes, M. & Hughes, C. & Shoffner, M. & Winslow,M., 1996," Java Networking programming", ISBN 1-884777-34-1

Joyle, M. & Quisquater, J.J, 1997, "Faulty RSA encryption", Tech. Report CG-1997/8, UCL Crypto Group, Louvain-la-Neuve.

Kelsey, J. & Schneier, B. & Wagner, D. & Hall, C., 1998,"Cryptanalytic Attacks on Pseudorandom Number Generators", Fast Software Encryption, Fifth International Workshop Proceedings (March 1998), Springer-Verlag, pp. 168-188.

Lenstra, A.K & Lenstra, H.W & Pollard, J.,1990, "The Number field sieve", proceedings of the 22nd ACM symposium on the Theory of computing.

Peterson,H. & Michels, M., 1998,"Cryptanalysis and improvement of signcryption schemes" , IEEE Proc.- Comput. Digit. Tech, Vol 145 , No2.

RSA Frequently Answered Question, RSA Laboratories

Rivest, R., 1995, "The RC5 Encryption algorithm".

Rivest, R., 1992, "The MD5 Message Digest algorithm", Internet RFC 1321.

Rivest, R. & Robshaw, M. & Sidney, R. & Yin,Y., 1998, "The RC6™ Block Cipher"

Saarinen, MJ. O. ,"A note regarding the Hash function use of MARS and RC6"

Schneier, B., 1996, "Advanced Cryptography Second Edition", ISBN 0-471-12845-7, Wiley publishers.

Schneier, B. & Kelsey, J. & Whiuting, D. & Wagner, D. & Hall, C. & Ferguson, N., 1999, "Performance Comparison of the AES Submissions", Second AES Candidate Conference.

Stinston, D.,1996, "Cryptography: Theory and Practise", CRC Press.

Zakon, R., 2000 "Hobbes' Internet Timeline" Copyright ©1993-2000
http://info.isoc.org/guest/zakon/Internet/History/HIT.html

ITL Bulletin, ADVISING USERS ON INFORMATION TECHNOLOGY, August 1999, http://www.nist.gov/itl/lab/bulletns/aug99.htm

Note: Almost all of these papers can be found at: http://www.counterpane.com/biblio/

Bibliography

Lewis, J. & Loftus, W., 1998, "Java Software Solutions", ISBN 0-201-57164-1, Addison Wesley.

Oaks, S. & Wong, H., 1999, "Java Thread, Second Edition", ISBN 1-56592-418-5

Hopson, K.C. & Ingram, E., 1996, "Developing Professional Java Applets", ISBN1-57521-03-5, Sams.net Publishing,

S.Singh, 1999, "The Code Book", ISBN 0-385-49531-5.

Schneier, B., 1996, "Advanced Cryptography Second Edition", ISBN 0-471-12845-7, Wiley publishers.

Bain, L. & Engelhardt, M., 1992, "Introduction to Probability and Mathematical Statistics, Second Addition", ISBN 0-534-98563-7, PWS-Kent Publishing.

Herman, E., 1996, "Teach Yourself CGI Programming with Perl 5 in a week", ISBN 1-57521-009-6, Sams.net publishing.

Appendix A

MSc Progress

February to June 1998

Attended Honours Java course for the trimester.
Design Concept of application.

July 1998 to April 1999

Implemented Auction application without security.
Researched security features.

May 1999 to October 1999

Implemented security aspects of program.

November 1999 to May 2000

Write up and presentation of MSc

May 2000

Hand in of MSc.

Appendix B

MD5

MD5 accepts a message M, which is made up of b number of bits. The following implementation is from [R.Rivest, 1992].

I.e $M_0 .. M_{b-1}$

Step 1

The message is then padded so it has a length of 448 modulo 512. It length is then considered congruent to 448 modulo 512. The first bit added is one, thereafter all other bits are zero. As modulo 512 is applied to the value, one needs to add at most 512 bits or at least one bit.

Step 2

A sixty four-bit representation of the original value of b is then added to the result of step 1, which means it has an overall length of 512 bits. Where M_0 to M_{n-1} is the 32 bit words in the new value. The value of n is divisible by sixteen.

Step 3

Four word buffers (A, B, C, D) are used as registers, which are initialized with the values

A: 01 23 45 67
B: 89 ab cd ef
C: fe dc ba 98
D: 76 54 32 10

Step 4

Four functions are used that accept three thirty two bit words and output one thirty two bit word.

$F(X,Y,Z) = XY \text{ OR } \text{not}(X) Z$
$G(X,Y,Z) = XZ \text{ OR } Y \text{ not}(Z)$
$H(X,Y,Z) = X \oplus Y \oplus Z$
$I(X,Y,Z) = Y \oplus (X \text{ OR } \text{not}(Z))$

```
For (i= 0;I<N/(16-1),I++)
        For (j=0;j<15,j++)
                Set X[j] to M[i*16+j].
        End For
End For
```

AA = A, BB = B, CC = C, DD = D

This following rounds uses a 64-element table T[1 ... 64]. This is made from the sine function. Let T[i] = 4294967296 * abs(sin(i)), (i is in radians).

Round One

Let abcd k s i denote

a = b + ((a + F(b,c,d) + X[k] + T[i]) <<< s).

Now apply it to the following operations, where ABCD is the four register 32-bit registers A, B, C, D in that order.

```
[ABCD  0  7  1]  [DABC  1 12  2]  [CDAB  2 17  3]  [BCDA  3 22  4]
[ABCD  4  7  5]  [DABC  5 12  6]  [CDAB  6 17  7]  [BCDA  7 22  8]
[ABCD  8  7  9]  [DABC  9 12 10]  [CDAB 10 17 11]  [BCDA 11 22 12]
[ABCD 12  7 13]  [DABC 13 12 14]  [CDAB 14 17 15]  [BCDA 15 22 16]
```

Round Two

Let [abcd k s i] denote

a = b + ((a + G(b,c,d) + X[k] + T[i]) <<< s).

Now apply it to the following operations:

```
[ABCD  1  5 17]  [DABC  6  9 18]  [CDAB 11 14 19]  [BCDA  0 20 20]
[ABCD  5  5 21]  [DABC 10  9 22]  [CDAB 15 14 23]  [BCDA  4 20 24]
[ABCD  9  5 25]  [DABC 14  9 26]  [CDAB  3 14 27]  [BCDA  8 20 28]
[ABCD 13  5 29]  [DABC  2  9 30]  [CDAB  7 14 31]  [BCDA 12 20 32]
```

Round Three

Let [abcd k s t] denote

a = b + ((a + H(b,c,d) + X[k] + T[i]) <<< s). */

Now apply it to the following operations:

```
[ABCD  5  4 33]  [DABC  8 11 34]  [CDAB 11 16 35]  [BCDA 14 23 36]
[ABCD  1  4 37]  [DABC  4 11 38]  [CDAB  7 16 39]  [BCDA 10 23 40]
[ABCD 13  4 41]  [DABC  0 11 42]  [CDAB  3 16 43]  [BCDA  6 23 44]
[ABCD  9  4 45]  [DABC 12 11 46]  [CDAB 15 16 47]  [BCDA  2 23 48]
```

Round Four

Let [abcd k s t] denote

a = b + ((a + I(b,c,d) + X[k] + T[i]) <<< s). */

Now apply it to the following operations:

```
[ABCD  0  6 49]  [DABC  7 10 50]  [CDAB 14 15 51]  [BCDA  5 21 52]
[ABCD 12  6 53]  [DABC  3 10 54]  [CDAB 10 15 55]  [BCDA  1 21 56]
[ABCD  8  6 57]  [DABC 15 10 58]  [CDAB  6 15 59]  [BCDA 13 21 60]
[ABCD  4  6 61]  [DABC 11 10 62]  [CDAB  2 15 63]  [BCDA  9 21 64]
```

Finally perform the following operations, and return the values in A, B, C, D as the result. A += AA, B+= BB, C += CC, D+= DD