"**Linux Gazette...*making Linux just a little more fun!***"

Using Java and Linux to crack the DES challenge

By Carlos Serrao

Abstract

DES has been used for a long time to guarantee the privacy of transactions in open communication environments, especially in inter-banking and financial ones. Nevertheless, the security level offered by this algorithm is not the same that was before. The DES working conditions have been shifting with the exponential growth of the Internet, and what was considered as being safe a decade ago is not safe anymore. What this article intends to show is how weak the actual DES strength is, and how and what a determined attacker can do to break it using the new Java programming language and the Linux operating system.

1. Introduction

Cryptography has always been used as an effective way for protecting sensitive information from non-authorised and curious eyes. Cryptography had its exponential growth with the Second World War. Both the Allies and the Nazis and the Japanese used this technology successfully. Basically, cryptography is the process that allows maintaining private communications. There are two main issues in cryptography: encryption and decryption. Both encryption and decryption are two of the issues in cryptography, but not the only. Issues like digital signatures and digital certificates are equally important. Nowadays, two different kinds of cryptography are used: secret key or symmetric cryptography and public key or asymmetric cryptography. The major difference between them is the key usage. In the first one, only one key is used both for encryption and decryption, and in the second one, two different keys are used, one for encryption and the other for decryption. Normally, the public key is freely distributed, while the private key is kept secret. All the public key system security lies on the secrecy of the private key. In this article, our attention will be mainly focused on the first case, the secret key cryptography. DES is just one of the algorithms that uses secret key cryptography. Others, like IDEA (International Data Encryption Algorithm), RC2 (Ron's Code 2) or RC5 (Ron's Code 5) also use it. DES (Data Encryption Standard) is a block cipher algorithm that was defined and adopted by the US government in 1977 as an official standard. It was originally developed by IBM in 1970 under the name of LUCIFER, and was rapidly adopted as the most used cryptographic system in the world [1]. Banks and financial institutions mainly use DES.

2. Attacks on cryptography

The main goal of cryptography is to maintain all the information secret from non-authorised people. Cryptanalysis is a subtopic of Cryptography that tries to break the secrecy (find the original text from a given encrypted text without knowing the key used for its encryption). A successful cryptanalysis can find both the original text and the used key. There are several methods for performing attacks to these algorithms. The success or failure of each of the attacking methods is strongly related to the amount of information the attacker can obtain from both the ciphertext and plaintext [2]. Some of the most used attacking methods are:

- Cyphertext only attack: the cryptanalist only has one or several encrypted text samples using the same key, and tries to find out the plaintext and key used.
- Known plaintext attack: in this case, both the ciphertext and the plaintext of one or several messages are available for the crytanalist.
- Choosen plaintext attack: it's quite similar to the one before. However in this case, the cryptanalist can choose the plaintext that he wishes to see encrypted.
- Adaptative chosen plaintext attack: similar to the method presented before. However, the cryptanalist is able to choose the subsequent texts according to a previously selected cyphertext / plaintext pair.
- Chosen ciphertext attack: for a symmetric cipher it's similar to the chosen plaintext attack. For asymmetric ciphers, the cryptanalist can choose the ciphertext to be decrypted.
- Chosen key or brute force attack: consists in the exhaustive search of of all the possible keys, decrypting the ciphertext with each one of the keys and trying to obtain a intelligible result.

3. Key robustness

The issue of key robustness has always been highly discussed. The bigger the cryptographic key is, the stronger is the ciphertext generated. However, if the cryptographic key is big, the cryptographic system is also more demanding in terms of processing power.

```
+------------+----------------------------------+
```

Year	Millions of Encryption per Second
1995	4
2000	32
2005	256

Key size	1995	2000	2005
40 bits	68 seconds	8,6 seconds	1,07 seconds
56 bits	7,4 weeks	6,5 days	19 hours
64 bits	36,7 years	4,6 years	6,9 months
128 bits	6,7e17 Myears	8,4e16 Myears	1,1e16 Myears

Table 1 - Key robustness computation based on key size and on hardware with 4000 dedicated chips. Source: Department of Computer Science, University of Bristol, 1996

Year	Millions of Encryption per Second
1995	50
2000	400
2005	3200

Key size	1995	2000	2005
40 bits	1,3 days	3,8 hours	28,6 minutes
56 bits	228 years	28,6 years	3,6 years
64 bits	58,5 Myears	7,3 Myears	914 years
128 bits	1,1e12 Myears	1,3e30 Myears	1,7e19 Myears

Table 2 - Key robustness computation based on key size and on 200 dedicated PCs. Source: Department of Computer Science, University of Bristol, 1996

The robustness of the keys highly depends on the available processing capacity. With a bigger processing power, less will be the necessary time to crack a cryptographic. On the other end, if the processing power increases also does the key size and consequently the robustness of cryptographic algorithms. But, what seems clear, is that keys that were considered has being safe two or three years ago, are not safe at all now.

4 DES - Data Encryption Standard

In 1972, the National Bureau of Standards (now, known as NIST) had launched a request to the scientific community to conceive a new cryptographic algorithm. This new algorithm should have the following characteristics:

1. High level of security
2. Completely specified and easy to understand
3. Available to everyone
4. Adaptable
5. Efficient enough to implement in a computer.

In 1974, IBM answered this request with an algorithm named LUCIFER (later it was called DEA - Data Encryption Algorithm or DES). Finally, in 1976 DES was adopted as a standard in US.

DES has kept itself, until now, for twenty years as an international standard. Although DES is finally showing some signs of its age, is still considered as one of the strongest and efficient algorithms in the world.

4.1. How DES works

DES is a block cipher. It encrypts 64 bits blocks each time. A set of 64 bits of plaintext enters the algorithm and a set of 64 bits of ciphertext comes out of the algorithm.

DES is a symmetric key algorithm and uses the same key for encryption and decryption processes. The keysize is 56 bits. DES uses a 56 bit key (the key is normally represented by 64 bits, and every eight bit is used just for parity checking).

At its simplest level, the algorithm is based on two simple principles: diffusion and confusion. DES applies substitutions followed by permutations to a plaintext, based on a given key. This process is called round, and DES applies it 16 times. The following scheme represents a more detailed look of DES.

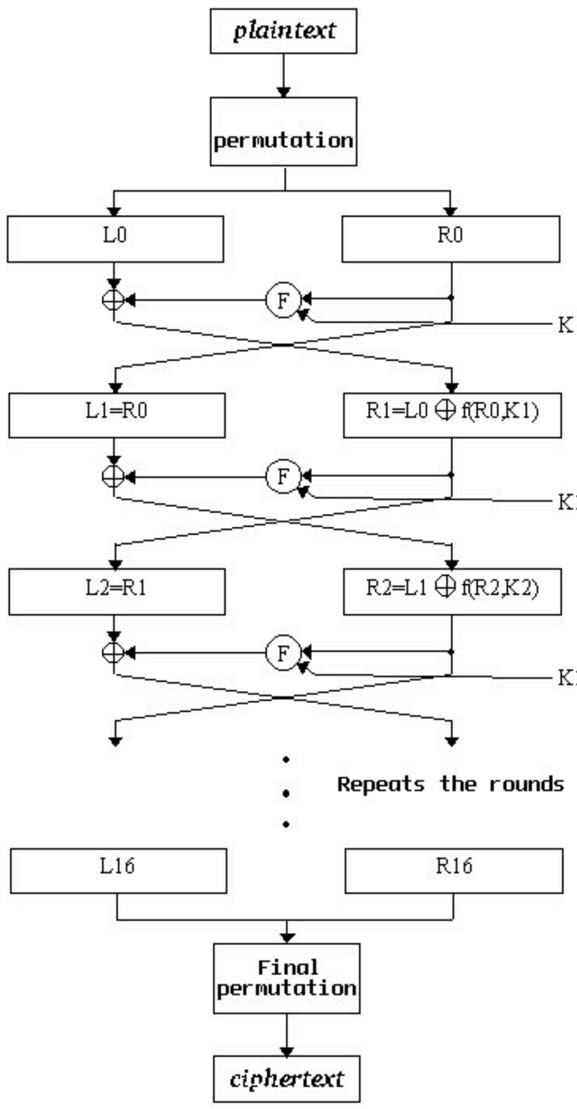

Figure 1 - A detailed scheme showing how DES works

A more detailed explanation is outside the scope of this article. More information can be found on any good book about cryptography.

5. RSA Labs Cryptographic challenges

The RSA Laboratories Data Security Division promotes and maintains a set of cryptographic challenges as research tools [3]. Some of the challenges that presently RSA holds, are:

1. RSA factoring challenge
2. Secret-key challenge
3. DES challenge

5.1. DES challenge

The original DES challenge was launched in January 1997, with the aim of demonstrating that 56-bit security, such as that offered by the US government's DES, offers only marginal protection against a committed adversary. This was confirmed when the secret key used for encryption was recovered on June 17, 1997. Since then it has been widely acknowledge that much faster exhaustive search efforts are possible and DES challenge II is intended to show how fast.

While the original showed DES was crackable using an exhaustive search attack, the goal of the new DES challenge is to see how quickly an exhaustive search attack can be accomplished to help judge the true vulnerability of DES.

Twice a year, on January 13 and July 13, at 9:00 AM Pacific Time, a new contest will be posted on the RSA homepage. The contest will consist of the ciphertext produced by DES-encrypting some unknown plaintext message that has a fixed and known message header. The first to recover the key wins, and the amount of the prize will depend on how fast the key was recovered.

5.2. DES challenge details

For each contest, the unknown message will be preceded by three known blocks of text containing the 24-character phrase: ``The unknown message is:``. While the mystery text that follows will clearly be known to a few employees of the RSA Data Security, the secret key actually used for the encryption will be generated at random and destroyed within the challenge-generating software. The key will never be revealed to anyone.

The goal of each contest is for participants to recover the secret randomly generated key that was used on the encryption in a faster time than that required for earlier challenges in the series.

6. Breaking DES

Many problems require a large amount of computational power to reach to a solution. Some problems, though, are amenable to an extremely high level of parallelization, and with today's Internet it is possible to broaden the reach of any large-scale effort to previously unanticipated levels.

Breaking DES is one of these problems. It is necessary to use a somewhat large computational power. Even if we consider a 56-bit key it is very difficult and hard task to perform.

The "best" strategy to break DES is to perform a brute force attack. This means having to test all the possible keys and analyse and compare the results. If we consider a 56-bit key, meaning that we will have approximately 256 possible combinations. Even with today common computational power this takes a while.

Breaking DES is clearly a NP-complete problem. It is impossible to find a solution to a NP-complete problem in polynomial time, but given a solution, it is possible to check whether it is valid or not. Typically, all cryptographic problems are NP-complete problems.

7. General architecture

The first aspect to consider when planning a general architecture for breaking DES is to choose between a hardware or software attack.

DES can be very easily implemented on a hardware chip, and lots of these chips can be used for breaking DES. One of the most obvious advantages of this approach is the resulting processing power. On the other hand, one of the problems arising from this architecture is its large cost.

Another possible approach for a general architecture is to use a software attack. One of the problems with this kind of architecture is the processing power. If only one computer is used the processing power obtained is quite disappointing. This type of attack is very easy to implement and is also less expensive than the previous one.

However, more interesting results can be achieved when using a distributed computing attack, based on the computational power increase obtained by joining the computing power of several computers.

8. Distributed computing

Distributed computing can be easily described as the effort to unify multiple networked machines in such a way that information or other resources can be shared by all of these connected computers. The hope is that sharing can take place over large areas, many machines, and many users, unifying them in a consistent and coherent framework.

Distributed computing became a field of study when computer hardware evolved from the mainframe, where everyone shared all the resources of a single machine, to the minicomputer. The minicomputer made it necessary for two people (or programs) to work together or share resources when they were on different machines. Co-ordinating that work, or providing access to those resources, is the goal of distributed computing.

Interest in distributed computing has increased with the advent of individual workstations and networked PCs, mainly with the spread of the Internet. Because these are single user machines, the need to share information, computing resources or data resources became immediate as soon as there was any joint work to be done.

9. Specific architecture

As it was stated before, one of the possible ways for building an architecture to break DES is through a distributed computing architecture.

However, some considerations should be take into account. For instance, what is the best configuration and which are its constraints.

No doubt that, in order to profit the availability of Internet's computing power the best solution is to implement typical client-server architecture. A server for distributing the keys and the clients to do the hard work: test all the keys and check the result.

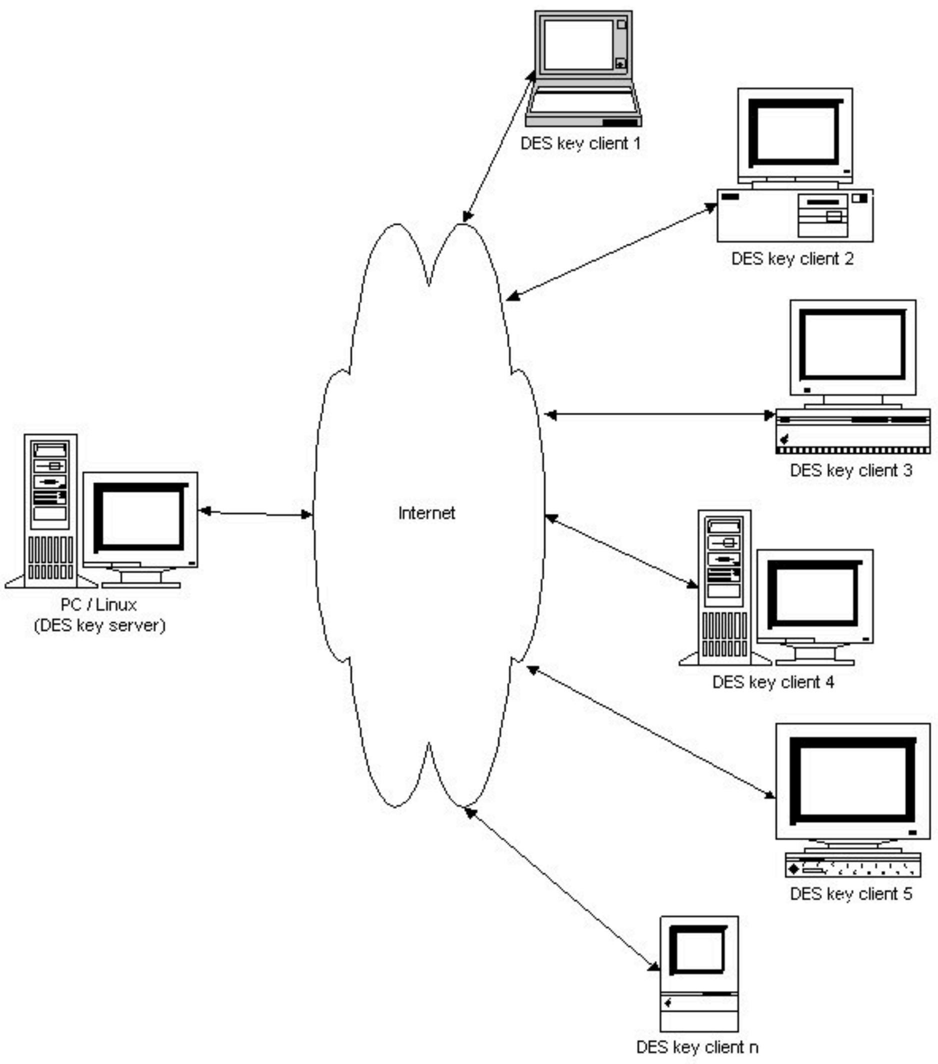

Figure 2 - Overview of the distributed architecture

9.1. Considerations

There are different approaches to an exhaustive search. The major consideration is whether the search is co-ordinated by some central server, or whether multiple processes start at random positions in the search space and run independently until a key is found.

The use of a central server poses some difficulties. As well as providing a single point of failure, there is also the potential of network congestion and failures.

A variety of precautions should be taken in account. Servers can be networked into a hierarchy, or replicated if resources allow, so that points of failure are less catastrophic. In addition, clients can test themselves to provide some level of assurance against malfunction and servers to provide assurance against malevolent clients can conduct more explicit testing on the clients. The server can have the client report on server-fabricated problems that can be checked at very low cost. Alternatively a client could calculate a checksum over all attempted solutions in the range examined and another client of the same architecture could check this.

9.2. Functionality

Although the client-server architecture presents some problems, it is still the easiest and cheapest one to implement and maintain.

The basic idea is to have a central server, that is in charge of performing simple tasks, like adding new clients to the contest, distribute new groups of keys and verify and update the results.

The server itself does only the easy work on the system. The hard work, trying all the possible keys and check the result, is done by the set of clients. The more clients the system has, the faster will be coverage of all the key-space.

The functionality of the server can be easily summarised in the following scheme.

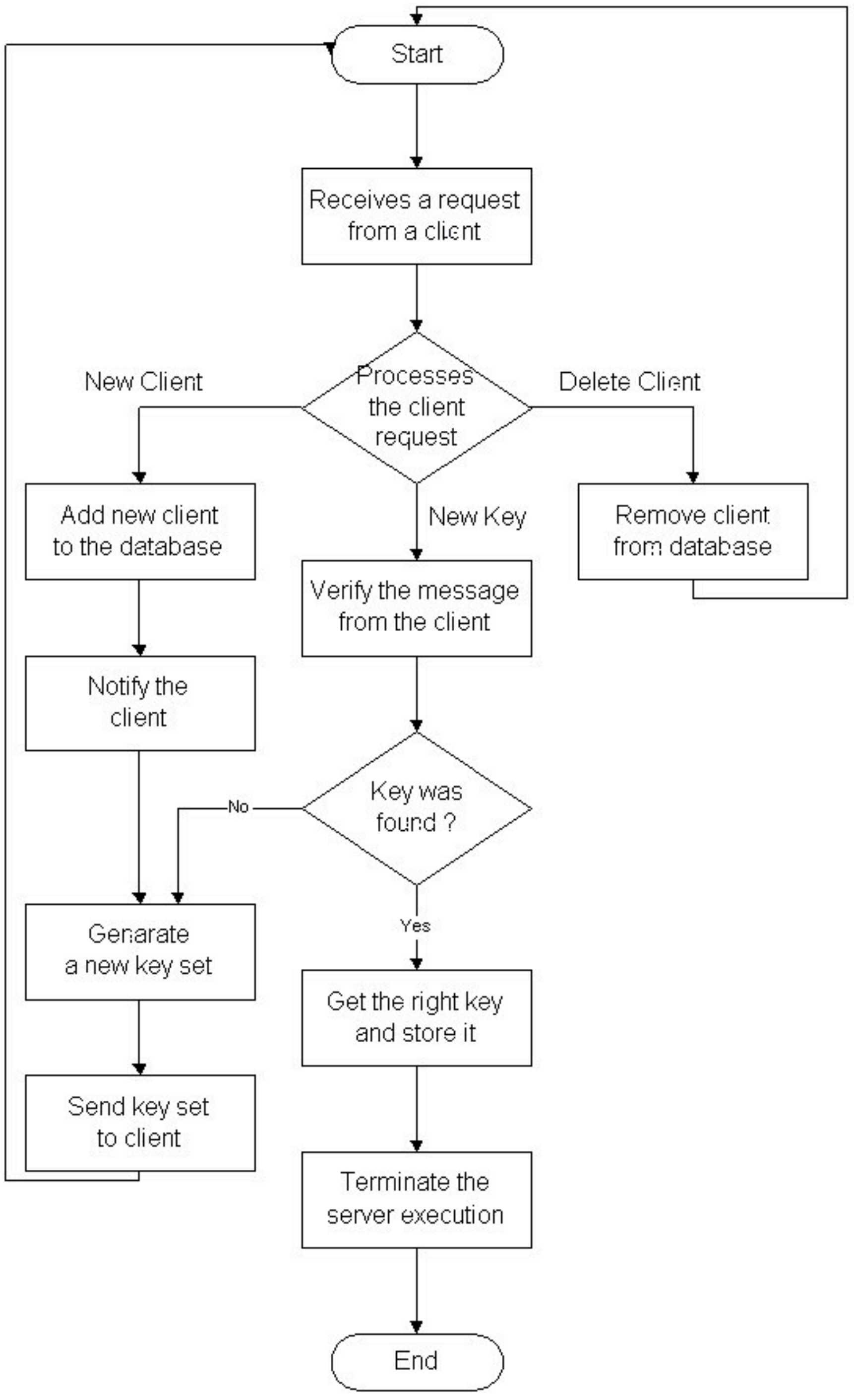
Start
Receives a request from a client
Processes the client request
New Client
Delete Client
New Key
Add new client to the database
Remove client from database
Verify the message from the client
Notify the client
Key was found ?
No
Yes
Genarate a new key set
Get the right key and store it
Send key set to client
Terminate the server execution
End

Figure 3 - DES key server functionality

The functionality of the client can also be summarized this way:

Basically, the main functionality of the client software is to test all the possible keys distributed by the key server and then analyse the result and test if the key is the right key or not.

If it is the right key the client key the client communicates with the server and sends a remark to the server stating that has found the key.

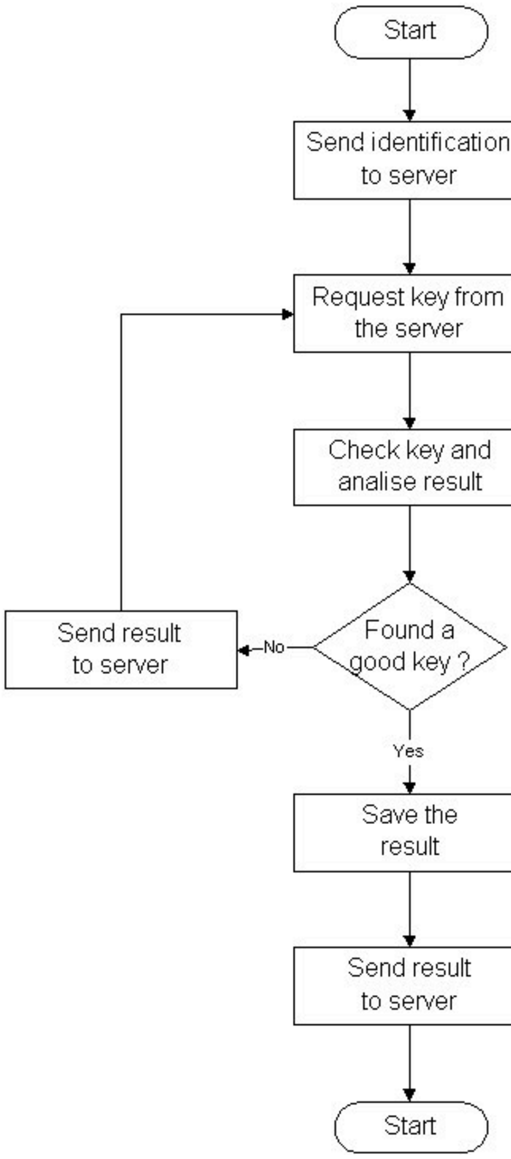

Figure 4 - DES key client functionality

10. Implementation

No doubt subsists that the best possible architecture for facing the DES challenge is to use a client-server distributed architecture over the Internet.

In order to implement such architecture, some important decisions have to be made, like choosing the best deployment platforms and the right tools.

10.1. Linux

Linux is a Posix Compliant operating system designed to run on the Intel architecture. The system also has extensions to accommodate System V and BSD requirements.

This OS is licensed under the GNU Public license and is as such, freely distributable provided the source accompanies the distribution or is at least made available to the recipient.

Linux runs on Intel processors 386 and later that are capable of utilising the 386 protected mode.

For a strictly minimum implementation you can expect to need about 10-15 Megabytes of disk space and 8 Meg of RAM. It is possible to run it in 4 Meg, but consider 8 a reasonable minimum for text based installation. In order to utilise X (the Unix windowing system) with any reasonable efficiency, expect to need about 16 Meg of RAM as a reasonable minimum (300M hardfile space).

The system is currently capable of compiling and running Posix compliant Unix programs, as well as Dos programs through the use of DosEmu. Windows applications have limited success running on Linux through the use of Wine (a windows emulator).

Although Linux is usually referred to in conjunction with 386/486/Pentium machines, it also runs on, or is currently being ported to, other architectures (eg DEC Alpha, Sun SPARC, MIPS, PowerPC, and PowerMAC).

10.2. Java

Java is an object oriented programming language developed by Sun Microsystems, and now further developed by its JavaSoft subsidiary.

At first glance it resembles C and C++, but it's different under the hood. Java is both a compiled and an interpreted language. Its source code is compiled into a universal form - bytecodes for a virtual machine - that can easily be ported across the Internet and interpreted and run on many platforms.

Compared to other programming languages Java is much simpler, robust, and cost-effective. It allows an application to be developed with a minimum of debugging and is instantly portable to many operating systems. Compared to other Internet solutions, Java offers unmatched performance and versatility while minimising the strain on the web servers by distributing the processing load to the client machines.

Java possesses also a series of additional APIs that allow the fast development of complex applications. Such APIs include for instance Networking, Distributed Method Communication, Database Connectivity and Cryptographic support.

10.2.1. Java RMI

Java RMI (Remote Method Invocation) technology is the basis for distributed computing in the Java environment. Because Java RMI was created after broad acceptance of the Internet and object oriented design, developers are treated to a dynamic and flexible environment for building distributed applications.

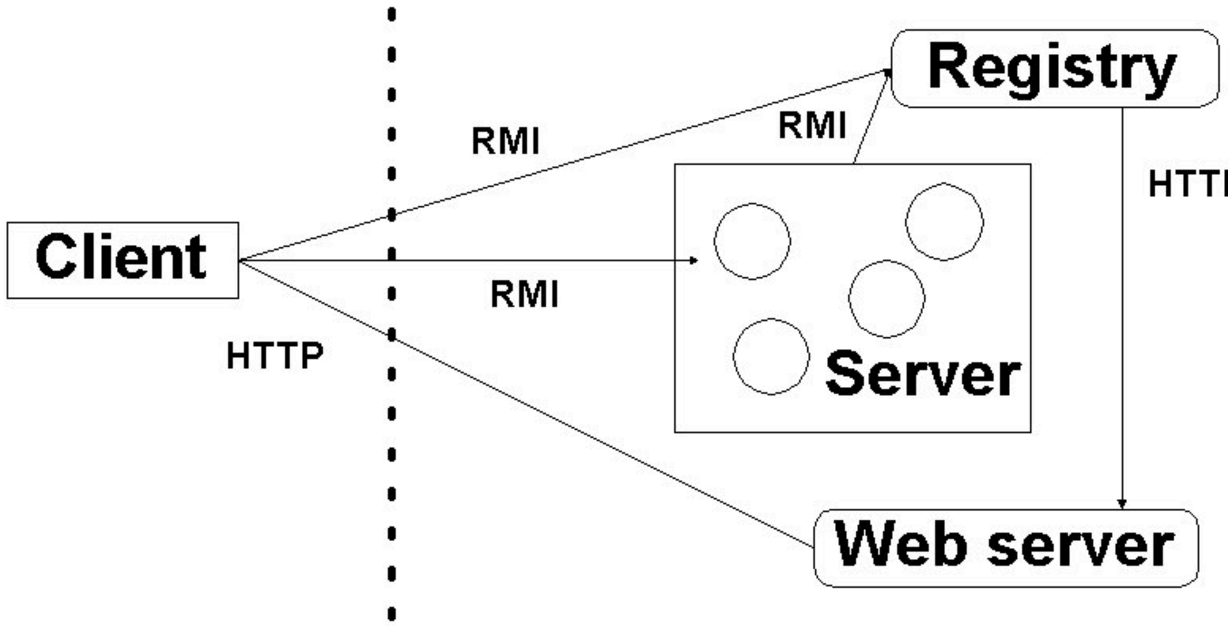

Figure 5 - The behaviour of Java RMI technology in a distributed application

Because of Java RMI technology, developers can now:

1. Easily create powerful distributed computer applications and network services for Java and non-Java environments
2. Use Java RMI, a single programming interface, for object communication in distributed applications.

10.2.2. JDBC

JDBC (Java Database Connectivity) is an application program interface (API) specification for connecting programs written in Java to the data in popular databases. The application program interface lets you encode access request statements in structured query language (SQL) that are then passed to the program that manages the database. It returns the results through a similar interface. JDBC is very similar to Microsoft's Open Database Connectivity (ODBC) and, with a small "bridge" program, you can use the JDBC interface to access databases through Microsoft's ODBC interface. For example, you could write a program designed to access many popular database products on a number of operating system platforms. When accessing a database on a PC running Microsoft's Windows 95 and, for example, a Microsoft Access database, your program with JDBC statements would be able to access the Microsoft Access database.

JDBC actually has two levels of interface. In addition to the main interface, there is also an API from a JDBC "manager" that in turn communicates with individual database product "drivers", the JDBC-ODBC bridge if necessary, and a JDBC network driver when the Java program is running in a network environment (that is, accessing a remote database).

When accessing a remote database, JDBC takes advantage of the Internet's file addressing scheme and a file name looks much like a Web page address (or URL). For example, a Java SQL statement might identify the database as:

```
jdbc:odbc://www.somecompany.com:400/databasefile
```

JDBC specifies a set of object-orient programming classes for the programmer to use in building SQL requests. An additional set of classes describes the JDBC driver API. The most common SQL data types, mapped to Java data types, and are supported. The API provides for implementation-specific support for transactional requests and the ability to commit or roll back to the beginning of a transaction.

10.2.3. JCE

The Java Cryptography Extension (JCE) extends the Java Cryptography Architecture (JCA) API about additional features for supporting encryption and key exchange.

The Java Cryptography Extension (JCE) is a set of APIs and implementations of cryptographic functionality, including symmetric, asymmetric, stream, and block encryption. It supplements the security functionality of the default Java JDK 1.1.x / JDK 1.2, which itself includes digital signatures (DSA) and message digests (MD5, SHA).

The architecture of the JCE follows the same design principles found elsewhere in the JCA.

10.2.4. PostgreSQL

Traditional relational database management systems (DBMSs) support a data model consisting of a collection of named relations, containing attributes of a specific type.

In current commercial systems, possible types include floating point numbers, integers, character strings, money, and dates. It is commonly recognised that this model is inadequate for future data processing applications. The relational model successfully replaced previous models in part because of its "Spartan simplicity". However, as mentioned, this simplicity often makes the implementation of certain applications very difficult. Postgres offers substantial additional power by incorporating the following four additional basic concepts in such a way that users can easily extend the system:

- classes
- inheritance
- types
- functions

Other features provide additional power and flexibility:

1. constraints
2. triggers
3. rules
4. transaction integrity

These features put Postgres into the category of databases referred to as object-relational. Note that this is distinct from those referred too as object-oriented, which in general are not as well suited to supporting the traditional relational database languages. So, although Postgres has some object-oriented features, it is firmly in the relational database world. In fact, some commercial databases have recently incorporated features pioneered by Postgres.

PostgreSQL is then a sophisticated Object-Relational DBMS, supporting almost all SQL constructs, including transactions, sub-selects and user-defined types and functions. It is the most advanced open-source database available anywhere.

10.3. Putting everything together

Now, after having in mind the tools and what architecture to use, it is necessary to put everything together.

The chosen development platform is Linux, because of its characteristics as a good development system. The language to be used is Java, because of its easiness.

On the server side, one of the most important things to be defined is the server database. This database is quite important since it will store important information about the contest, like detailed information about the contest clients, the set of keys that each client is currently processing, which was the last key to be issued among other things.

The database will be developed using a free object-relational database called Postgresql. The server software, totally developed on Java, will interface with the database through the JDBC API.

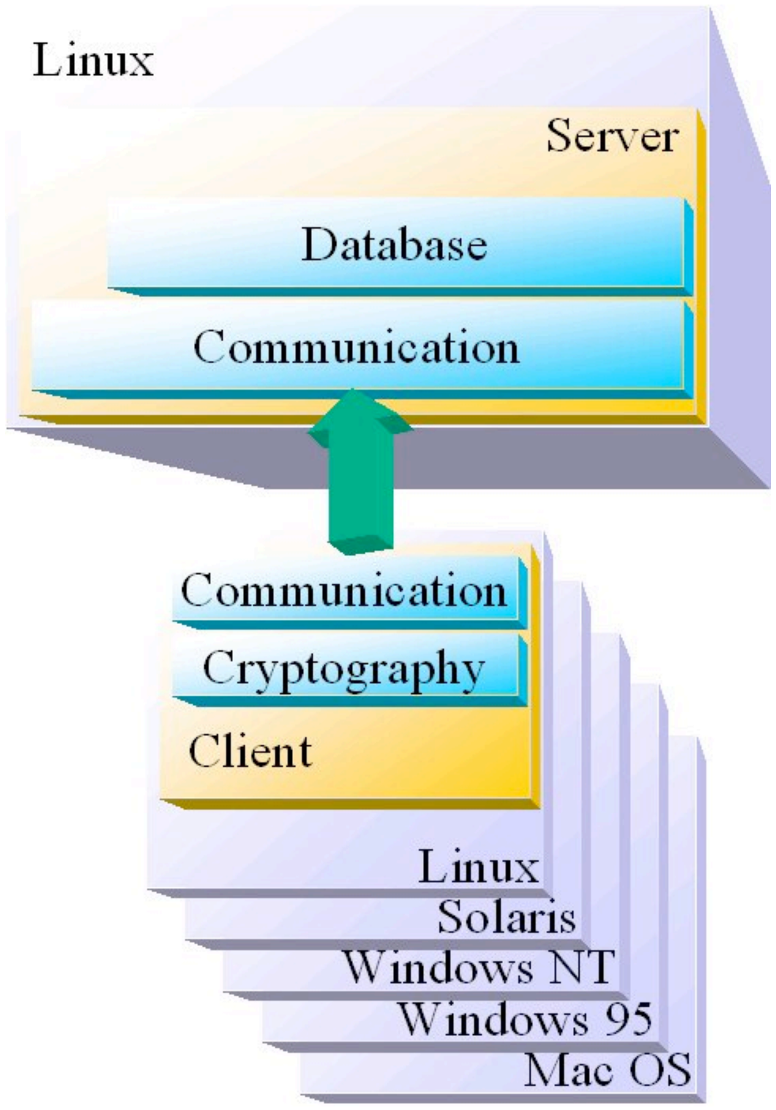

Figure 6 - Client-Server final architecture configuration

One of the main functions of a server is to wait and receive requests from a client. The DES key server has the same behaviour. The server starts, waits and processes requests. In order to communicate with the clients it is necessary to have some network functionality's. In this case, the server receives requests from the clients through RMI. RMI was chosen, because it adds a layer of abstraction between the Java program and the network complexities.

On the client side, one of the most important things to be implemented is the cryptographic functionality. The client software should be capable of use and process DES algorithm. Since the client software will know part of the plaintext and the full secret message, it has to be capable of trying to decrypt the secret message using a key supplied by the server and compare the result with the partial plaintext.

As it is in the server, this client software is totally implemented in Java. This allows that a larger number of different computers and platforms to join rapidly to the contest, enlarging considerably the computational power to find as quickly as possible a solution for the proposed problem - find the correct DES key.

11. Conclusion

DES 56-bit is not safe anymore. It is clear that with today's computing power, and with increasing network capabilities, like Internet, it is easy to set-up an architecture for cracking a DES key.

Linux and Java, are two of the tools that allow the easy creation of such architectures and make it available to almost everyone. Linux, because it is a simple, fast, powerful and free operating system that allows the building of power server capabilities. Java, because it is easy to learn and architecture independent allowing a fast development for a large number of different platforms.

References

[1] RSA Laboratories - Cryptographic Research and Consultation, "Answers to Frequently Asked Questions About Today's Cryptography - Version 3.0", RSA Data Security, 1996

[2] Schneier, Bruce, "Applied Cryptography - Protocols, Algorithms and Source code in C", John Wiley & sons, Inc., 1996

[3] "DES Challenge II", RSA Laboratories, RSA Data Security, http://www.rsa.com/rsalabs/des2, 1997

Table of Contents | Front Page | Frequently Asked Questions