Buletinul Ştiinţific al Universităţii "Politehnica" din Timişoara

Seria ELECTRONICĂ şi TELECOMUNICAŢII
TRANSACTIONS on ELECTRONICS and COMMUNICATIONS

Tom 51(65), Fascicola 2, 2006

Cryptographical System For Secure Client–Server Communication

Mircea-Radu Campean[1] and Monica Borda[2]

ABSTRACT

The aim of this paper was the research of a way to implement a cryptographical system for secure Client-Server communication, designed to satisfy the specific needs of the health care domain. A real IP based Client-Server application was created, that assures confidential message transfer using standardized cryptographic algorithms and components. A particular PGP (Pretty Good Privacy) like architecture was designed to ensure the communication security. Low costs, along with an easy to use implementation, represent decisive advantages when trying to implement the system in the medical area,which has limited budget for informatization.

Keywords: Cryptography, Encryption, Decryption, Authentication, Confidentiality, Client-Server

I. INTRODUCTION

Internet network development created the need for new types of services. The health care domain is one area, which is starting to benefit from the advantages Internet offers, providing different kind of services to help patients when they need medical care.
In our research we wanted to create a system, which allows patients to communicate with theirs doctors, using a PC connected to the Internet. For this purpose we developed a Client – Server application with a series of key features:

i. limited access to registered users
ii. ensures confidentiality and authentication
iii. intuitive interface for the Client
iv. limited hardware resources

Borland Delphi 7 was chosen as the programming language to create the Client – Server application after carefully considering its advantages over other solutions (i.e. object oriented programming language; high quality debugging support; easy to create user friendly interface; provides good error handling).

II. CLIENT-SERVER APPLICATION

Before developing an application we had to establish what were the requirements of the service we wanted to offer through our project. We intended to create a simple system, which could be used by patients to easily contact their doctor, using special software installed on a PC connected to Internet network. As shown in Figure1, we imagined that all communication, between doctor and patients would be filtered through the Server component, which runs on a computer situated at the Medical Care Centre.

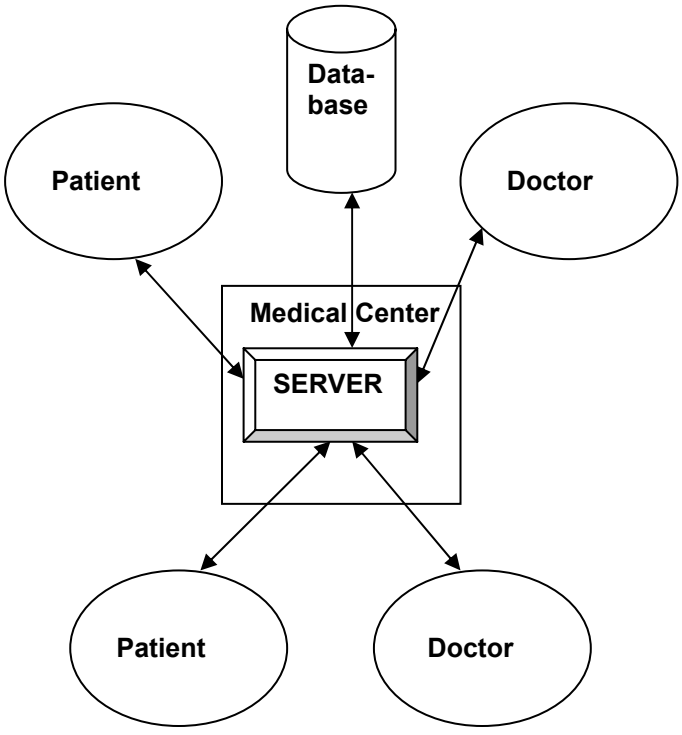

Fig 1: System structure for application

Due to the fact that both doctors and patients use the same software (the Client application) to gain access

[1] Facultatea de Electronică şi Telecomunicaţii, Departamentul Comunicaţii Str. Constantin Daicoviciu, No. 15, 400020, Cluj-Napoca, e-mail Radu.Campean@com.utcluj.ro
[2] Facultatea de Electronică şi Telecomunicaţii, Departamentul Comunicaţii Str. Constantin Daicoviciu, No. 15, 400020, Cluj-Napoca, e-mail Monica.Borda@com.utcluj.ro

into the system, the Server has to activate different features of the Client component according to the specific user rights, either doctor or patient. Also a simple database is used in order to store messages sent to/from users. The messages are stored in their original encrypted form, and this is done only if the recipient is not logged into the system when the messages are sent.
For the Client – Server part of the project, which solves the communication problems, we have used Indy (Internet Direct). Indy is an open source Internet component suite composed of Internet protocols written in Delphi and it is included in Delphi 6 and the later versions. There are more than 60 components especially designed to support network programming with Delphi, all grouped in four different categories:

- Indy Clients
- Indy Servers
- Indy I/O Handlers
- Indy Misc

II.1. THE SERVER APPLICATION

The Server is designed in such a way to implement a few basic functions like: connecting clients, authenticating users (verifying their *UserID* and *Password*), basic message processing, administrating the database, which contains user information and stored messages.
The Server part of the project is built around *TIdTCPServer* component, which implements a multi-threaded TCP (Transmission Control Protocol) server. This component uses one or more threads to listen for clients connections, and in conjunction with *TIdThreadMgr*, allocates a separate thread to handle each client connection to the server.
To successfully start a TCP server we need to specify the IP address and Port where the Server listens for clients. This is done using the Bindings property of the *TIdTCPServer* component. Also an OnExecute event handler had to be written, enabling the Server to reply to commands sent from the client.
As mentioned before the Server has to manage information found in a database. For this purpose we had used components found on the ADO (ActiveX Data Object) page. ADO is a set of COM components (DLLs) that allows you to access databases as well as well as e-mail and system files. Three data aware components were used in this project from the ADO page:

- *DBGrid:* - it is used to browse through the records retrieved from a table or by query
- *DataSource*: -used to provide a link between a dataset and DBGrid component on a form that enables display, navigation and editing of the data underlying the dataset
- *ADTtable:* -represents a table retrieved from an ADO data store

II.2. THE CLIENT APPLICATION

The Client component is based upon *TIdTCPClient*, which encapsulates a complete TCP (Transmission Control Protocol) client. As a first step when connecting to a TCP server, the Host and Port properties (the IP address and port where the Server awaits client connections) of the TIdTCPClient, have to be set.
In general a server cannot, by default, send a command or data to a client without having the client specifically asking for something. In our case we wanted the server to be able to initiate a scenario, in case it has a message to deliver. Because a Client (TIdTCPClient) does not implement a standard listen event, a TTimer component was added for handling the eventual command from the Server in an OnTimer event handler.
As mentioned before, doctors and patients will use the same Client application in order to connect to the Server, and gain access into the system. Each user will have a unique UserID and Password. A user-friendly interface was designed for the Client component, which has the following features:

- "*Login*" button - used when a client is trying to get access into the system
- "*Logout*" button – used for exiting the system
- "*Send*" button – used for sending messages to other users
- "*Clear*" button – cleans the message box
- "*ChangePassword*" – allows users to change their passwords
- "*AddPatient*" – allows doctors to add a new patient to their own list of *CONTACTS*

Note: Every user, patient or doctor, has one "Contact List" that contains other users, and with whom they can communicate.

III. SECURITY ISSUES

When designing a communication system one of the main concerns is the protection of the transmitted data, to ensure the confidentiality of system users. This is done using different cryptographic algorithms.
In order to ensure the communication security, a particular PGP like architecture was designed.(Note: we did not use the PGP system, we only used the principles of PGP security). This solution was chosen because PGP can be used to protect data in storage, in contrast to security protocols like SSL, which only protects data in transit over a network. PGP uses symmetric and asymmetric – key cryptography. The asymmetric cryptography part assumes that the recipient of a message has previously generated a key pair, a public key and a private key. The destination's public key is used by the sender to encrypt a secret key (session key) that is then used to encrypt the message (plaintext). The recipient of a PGP encrypted message decrypts the session key using its private key and then decrypts the message using the decrypted

key. Using two ciphers makes sense since there is a considerable difference in operating speed between public key and symmetric key cryptography, the latter being much faster. As an addition to basic PGP is the possibility to detect whether a message has been altered, and whether it was actually sent by the person who claims to be the sender. To solve this, the sender creates a digital signature of the message using RSA or a DSA signature algorithm, which is then compared with a computed message digest at the recipient.

The algorithms chosen in our system were: Triple DES using 128 bits keys; RSA with 512 bits keys; SHA-1 with a digital signature of 20 bytes encrypted with a 512 bits RSA key.

For this part of our project we used LockBox, a cross-platform library that can be used in Borland Delphi and C++ Builder applications under Windows. LockBox provides services that enable programmers to add cryptography to their own projects. There is also the possibility to digitally sign documents, using components that encapsulate the required functionality.

LockBox also contains a component hierarchy to offer an easy to use and but still powerful encryption possibility. Figure 2 shows this class hierarchy. AT its base is an encryption engine class, TlbCipher. This class has virtual methods to encrypt and decrypt an arbitrary buffer of data into another, a file into another, a stream or a string into another.

Two simple descendant classes act as roots for the two types of ciphers, symmetric and asymmetric. These classes provide extra functionality by dealing with keys, a single private key in the symmetric case and the pair of public and private keys in the asymmetric case. Finally there are the descendent components that perform the actual encryption and decryption using specific ciphers.

In our Client – Server application we have used three of the components presented in the component hierarchy (Figure2):

- *TLb3DES* – used for implementing the symmetric key algorithm
- *TLbRSASSA* –creating and verifying digital signatures
- *TLbRSA* –used for asymmetric key cryptography

In the encryption process, except for the Triple DES keys, no key variables are assigned, those being set up in the user authentication phase. The encryption is done on strings instead of files, the main reason being the implementation of the client server communication part. The encryption has three important steps. First the digital signature (encrypted hash) is generated using the plaintext and the sender's private key. As mentioned before SHA-1 with a digital signature of 20 bytes encrypted with a 512 bits RSA key has been chosen in this project. Then using Triple DES algorithm the plaintext and digital signature are encrypted using a session key. The last step involves the encryption of the session key using the recipient public key. As I previously said the application works with strings as plaintext and ciphertext. The decryption process acts as the reverse function of the encryption. Firstly, the recipient decrypts the session key using its own private key, and then using the session key decrypts the plaintext and digital signature.

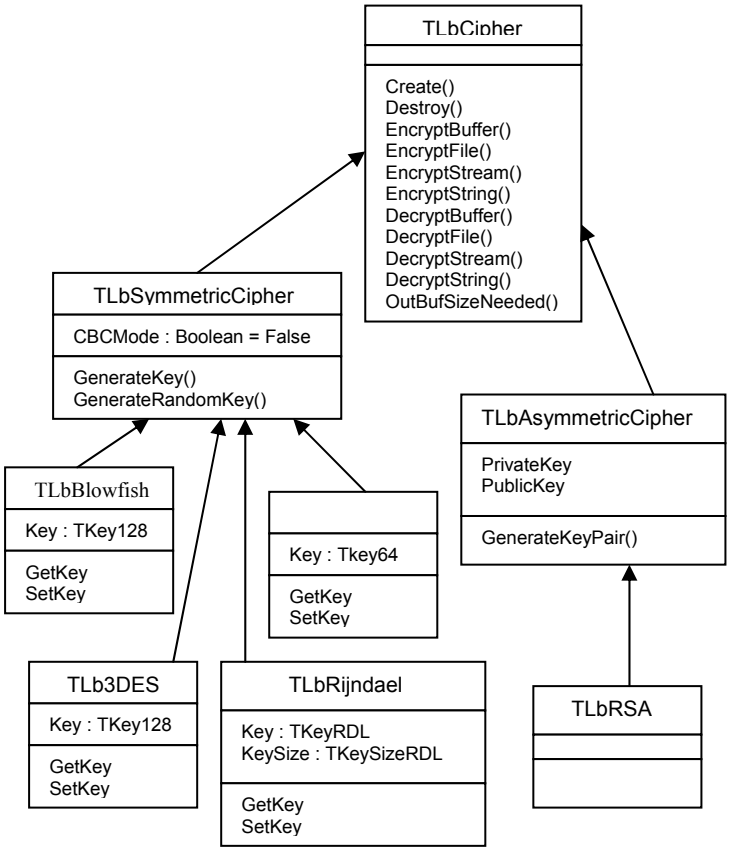

Fig*2*: Component hierarchy

IV. CONCLUSIONS AND FUTURE WORK

The system we have created has been tested in laboratory conditions on a limited number of computers thus on limited hardware configurations. We've had good results when we tested the application on computers with different versions of Windows OS (operating system) without having any kind of conflicts stating incompatibilities. The versions on which the application was tested were: Win 98, Win 98SE, Win 2000, Win XP (Home and Professional editions).

Another aspect we were interested in was the hardware resource needed on a PC (Personal Computer), which runs the Client or the Server application. The lowest hardware configuration that we tested had a 450 MHz processor and 64 MB of RAM. The Client application had no problem running on this system and both the processor and the memory usage were at low levels. The server on the other had would need better resources (at least 128 MB of RAM and a processor at around 1GHz) when there is a large number of clients connected in the system.

As said before the entire system was tested using a limited number of computers. The maximum Clients that we had connected to the Server at one moment were 20 and the system performed well. There should be no problem when a larger number of Clients connect to the Server since both implementations, Client and Server, are optimised.

In the future we want to develop the system so that it can be used in hospitals to create and maintain a database containing patients' medical charts and when needed, the patient's medical history could be sent to another hospital using the Client-Server application in a very short period of time. Access to the files containing the medical charts will be granted only to the patients' doctors ensuring in this way the patients confidentiality.

V. REFERENCES

[1]Bruce Schneier, Applied Cryptography Second Edition: Protocols, Algorothms, and Source Code in C, John Wiley & Sons, New York, 1996

[2] Alfred J. Menezes, Paul C. Van Oorschot, Scott A. Vanstone, Handbook of Applied Cryptography, CRC Press, Boca Raton, 1997

[3] William Stallings, Cryptography and Network Security – Principles and Practice. Second Edition, Prentice Hall, Upper Saddle River, New Jersey, 1999

[4]Titu Bajenescu, Monica Borda, Securitatea în informatică şi telecomunicaţii, Dacia, Cluj-Napoca, 2001

[5] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest , Clifford Stein , Introduction to Algorithms, Second Edition; MIT Press and McGraw-Hill, 2001

[6] Marco Cantu, Mastering Delphi 7, Publisher: Sybex Inc..,2003 (ISBN: 0-7821-2874-2)

[7] TurboPower LockBox2 – Manual [pdf] , TurboPower Software Company, 2000