Implementing Authentication Protocol for Exchanging Encrypted Messages via an Authentication Server based on Elliptic Curve Cryptography with the ElGamal's Algorithm

Konstantinos Chalkias, George Filiadis, and George Stephanides

***Abstract*—**In this paper the authors propose a protocol, which uses Elliptic Curve Cryptography (ECC) based on the ElGamal's algorithm, for sending small amounts of data via an authentication server. The innovation of this approach is that there is no need for a symmetric algorithm or a safe communication channel such as SSL. The reason that ECC has been chosen instead of RSA is that it provides a methodology for obtaining high-speed implementations of authentication protocols and encrypted mail techniques while using fewer bits for the keys. This means that ECC systems require smaller chip size and less power consumption. The proposed protocol has been implemented in Java to analyse its features and vulnerabilities in the real world.

***Keywords*—**Elliptic Curve Cryptography (ECC), ElGamal, and authentication protocol.

I. Elliptic Curve Cryptography (ECC)

ELLIPTIC curves have been extensively studied for over a hundred years, and there is a vast literature on the topic. Originally pursued mainly for aesthetic reasons, elliptic curves have recently become a tool in several important applied areas, including coding theory, pseudorandom bit generation, and number theory algorithms [1].

In 1985, Koblitz [2] and Miller [3] independently proposed using the group of points on an elliptic curve defined over a finite field in discrete log cryptosystems. Today elliptic curves are used widely in public key cryptosystems. They also provide a methodology for obtaining efficient implementations of network security protocols [4]. Since their introduction, a broad discussion on their security and efficiency has been carried on. It is this very efficiency that makes them so interesting for us today.

We investigate what an elliptic curve is, and what its algebraic properties are. Given any field $\boldsymbol{F}$ — thus a set with two operations, addition and multiplication that "know about" each other — we can consider solutions to various types of equations with coefficients in $\boldsymbol{F}$. An elliptic curve $\boldsymbol{E}$ over the field $\boldsymbol{F}$ is given by a cubic equation of the form

$$Y^2 + a_1XY + a_3Y = X^3 + a_2X^2 + a_4X + a_6 . \quad (1)$$

We let E(F) denote the set of points (x, y) ∈ F2 that satisfy this equation, along with a "point at infinity" denoted O. Fig.1 gives an example of an elliptic curve over the field of real numbers $\boldsymbol{R}$. In this particular case, the graph consists of two separated parts, but this is not always the case. It turns out that $\boldsymbol{E(F)}$ has some interesting properties. A group operation can be embedded in this set, and in this way, we obtain an abelian group. This is best explained geometrically. Take two points on the curve, P and Q and construct the line through these two points. In the general case, this line will always have a third point of intersection with the curve. Take this third point and construct a vertical line through it. The other point of intersection of this vertical line with the curve is defined as the sum of P and Q. If P and Q are identical, then the line to be constructed in the first step is a tangent to the curve, which again has exactly one other point of intersection with the curve. It can be shown that this operation is a properly defined group operation if an extra point at infinity O is added to the curve, which lies infinitely far on the vertical axis. This point O is the additive identity of the elliptic curve group. For example, in Fig. 1, this point should be added to P in order to obtain P as the sum.

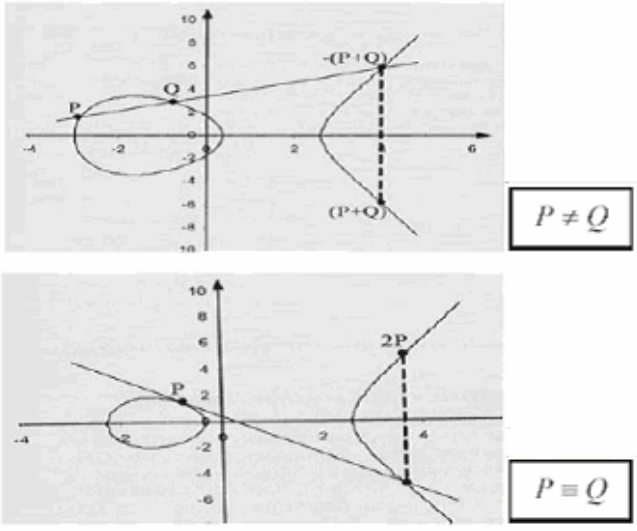

Fig. 1 Example of an elliptic curve over the field of real numbers $\boldsymbol{R}$ and demonstration of the group operation

The elliptic curves that are used in cryptography are defined over finite fields. Considering only finite fields we get an

Authors are with Department of Applied Informatics, University of Macedonia, Egnatia 156, GR-54006 Thessaloniki, Greece.

"easier" equation. If adding the multiplicative identity 1 to itself in F never gives 0 then we say the field has characteristic 0; otherwise, there is a prime number p such that $pa = 0$, for all $a \in F$ and p is called characteristic of the field — char(F). If now char(F) ≠ 2, 3, then there is an admissible change of variables which transforms E to the curve

$$y^2 = x^3 + ax + b. \quad (2)$$

where a, $b \in F$. The discriminant of the curve is $\Delta = -16(4a^3 + 27b^2)$. Algebraic formulas for the group law can be derived from the geometric description. These formulas are presented next for elliptic curves E of the simplified form (2).

This paper focuses on the elliptic curves defined over Fp when char(Fp) ≠ 2, 3 and $p > 3$. For a given prime, p, the finite field of order p, Fp is defined as the set Zp of integers {0, 1, ..., p – 1}, together with the arithmetic operations modulo p. Equation (2) is written as

$$y^2 \equiv x^3 + ax + b \ (mod\ p). \quad (3)$$

where $\Delta \not\equiv 0 \pmod p$ (to exclude singular curves). The point addition is based on the line property, which was described above. So, *if* $y2 \equiv x^3 + ax + b$ *(mod p)* is the equation of the elliptic curve and $y \equiv dx + c \pmod p$ is the equation of a line that intersects the curve at the points P(x1, y1), Q(x2, y2) then we can compute the third point –(P + Q) (its coordinates are $(x_3,\ dx_3 + c)$) by combining the equations. To do this we first need to find the equation of the line. We compute

$$d \equiv \frac{y_2 - y_1}{x_2 - x_1} \pmod p \text{ or } d \equiv \frac{3x_1^2 + a}{2y_1} \pmod p \text{ if } P \equiv Q.$$

Then, x3 ≡ d^2 – x1 – x2 (mod p) and y3 ≡ d(x1 – x2) – y1 (mod p). In case of Q = – P = (x1, – y1) there is no d, which lead us to the point at infinity O. [5]

The point multiplication is another well known process. If P is a point and n is a positive integer, then nP denotes P+P+…+P (with n summands). To compute nP for a large integer n, it is inefficient to add P to itself as an n-fold sum. Instead, we use the same binary expansion scheme we use to compute a^n. In other words, it is much faster to use successive doubling and even if n is very large, the size of the coordinates of the points does not increase very rapidly; when we are working over a finite field Fp we can continually reduce modulo p and thus keep the numbers involved relatively small. The method of successive doubling can be stated as follows [5], [6], [7]

Input: n (L-bits), P

$Q = 0$, $i = L - 1$

$c_0c_1\ldots c_{L-1}$ (n-binary representation), n = $\sum_{i=0}^{L-1} c_i \cdot 2^i$

while (i ≥ 0) ⇒ $Q = Q + Q$

if (c_i = 1) ⇒ $Q = Q + P$

$i = i - 1$

Output: Q

On the other hand if the points P and nP are given, it is very difficult to determine the value of n; this is the discrete log problem for elliptic curves and is the basis for the cryptographic application that will be discussed in the next section.

Several approaches to encryption/decryption using elliptic curves have been analysed in the literature, and in such a system the first task is to encode the plaintext message to be sent as a point of the elliptic curve. In order to assign the message to a point of the elliptic curve we can follow the next steps:

- We get the byte representation of the message (ASCII),
- $x = k$*message + 1, found = false
- While x <= (k*message + k) and found = false
  - We try to solve the equation. If there is a solution then found = true and the algorithm stops. So, the message has been successfully assigned to the point (x, y). Else, $x = x + 1$.

Shanks-Tonelli's algorithm is used to compute the square root of the equation. According to Koblitz [2] the value of k must be between 30 and 50 to be sure that the above algorithm will find a solution.

The number of points of an elliptic curve is definite. According to Hasse's theorem this number |E| is between a lower limit of $p+1-2\sqrt{p}$ and an upper limit of $p+1+2\sqrt{p}$ $\left(\left(p+1-2\sqrt{p}\right) \le |E| \le \left(p+1+2\sqrt{p}\right)\right)$. This is why p must be a very big prime number. [5]

II. ELGAMAL ALGORITHM

In the ElGamal algorithm each user has a private and a public key and of course an elliptic curve equation $y^2 \equiv x^3 + ax + b \pmod p$. The private key '$nb$' is a random Big Integer where $nb < p$. Only the user knows this number. The public key consists of the coefficients 'a' and 'b', the modulus 'p', a random point 'G' and the point $Pb = nbG$ of the elliptic curve. The public key is published to the other users.

The procedure of the encryption is described below. The message that will be sent to the recipient corresponds to a point 'Pm' of the elliptic curve of the recipient.

I. The sender chooses a secret random integer 'k', where $k < p$, and sends the points 'kG' and '$Pm + kPb$' to the recipient.

II. The recipient computes the point $Pm = (Pm + kPb) - (kG)nb$.

III. The decoding is successful because $Pb = nbG$,

$(Pm + kPb) - (kG)nb = Pm + knbG - knbG = Pm$

IV. Only the person who owns the nb can make the decryption. [5]

III. AUTHENTICATION PROTOCOL

A. Introduction and Assumptions

There are a lot of authentication protocols and too many algorithms have been suggested to secure key exchange. Even

though some of them are very efficient, most of them are a combination of two or three algorithms. For example, it is common to use public key system to establish a symmetric key that is then used in a symmetric system. Even though this combination has improved efficiency when massive amounts of data are being transmitted, the use of two different algorithms makes the cryptographic system more vulnerable to attacks. In cases where the sent data is limited the symmetric encryption could be avoided. For this purpose a complete public key encryption scheme must be implemented. The authors' effort was to implement a protocol based on one simple algorithm for exchanging text messages between the users of a "Message Server". The problems that had to be solved were both the sender's authentication and the secrecy of the messages [8].

In order to implement this protocol we have to work based on some assumptions. The first assumption is that **every user has to be connected to the server** to send a message and not send it directly to the recipient (and vice-versa). The second assumption is that **the public keys are published in a public directory** and in a web page for security reasons. Note that the server sends the public key of the recipient to the sender in a safe way (this is explained below). The third one is that **the server has a key pair** too. The fourth one and the most important is that **the server's public key is the only key that does not need verification**. Every user stores servers' public key when he creates an account. At last, we assume that **there is no need for a safe communication channel** e.g. SSL.

The reason of those assumptions is that the first idea was to create an application for students to send and receive encrypted messages by using a loyal and trustworthy server such as a university server. The role of the server is very important because no digital signature algorithm is implemented. The server takes all responsibility of the members' verification by providing to both sender and receiver a random session number known as nonce.

B. Description

Equation Notations:

- Clear message: M
- Mail Identity: Id (Every message has a unique number)
- Server: S
- Agents: A(Alice), B(Bob)
- Nonces: N_1, N_2, N_3
- Private Keys: Pr_X
- Public Keys: Pub_X
- Compound messages: $\{M_1,M_2,\ldots,M_n\}$
- Encrypted messages: Pub_X $(\{M_1,M_2,\ldots,M_n\})$

Sending Process: (From=Alice, To=Bob)

(1.) $\mathbf{A \rightarrow S : Pub_S(\{ From, To, N_1\})}$
(2.) $\mathbf{S \rightarrow A : Pub_A(\{ Pub_B, N_1, N_2, Id \})}$
(3.) $\mathbf{A \rightarrow S : \{ Pub_B(\{ M, N_2 \}), Id \}}$

Receiving Process: (Bob receives the message)

(4.) $\mathbf{B \rightarrow S : Pub_S(\{ To, N_3 \})}$
(5.) $\mathbf{S \rightarrow B : \{Pub_B(\{M, N_2\}), Pub_B(\{N_2,N_3\})\}}$

C. Explanation

The sending process includes three steps. In the first step the user sends a request to the server. The request consists of three elements: "From", "To" and a nonce "N_1". Of course "From" is the nickname of the sender and "To" is the nickname of the recipient. "N_1" is a random message needed for the verification of the server when the server replies. All the three elements are concatenated (using tags) and create a stream. This stream is encrypted with the server's public key and the encrypted stream is sent to the server.

After receiving the request, the server decrypts the stream and gets "From", "To" and "N_1". Then the server must reply by providing a mail Id, the "N_1", a new nonce "N_2" and the receiver's public key to the sender. Meanwhile, the server inserts a new row in his internal mail database. Every row consists of the following fields: {From, To, Id, $Pub_B(\{M,N_2\})$, $Pub_S(N_2)$}.

In the third step the sender has got the reply and verifies the "N_1". If the verification is correct then he can use the public key he received. The sender concatenates the clear message with the "N_2" he received and encrypts them with the recipient's public key. After that, he sends this encrypted message to the server followed by the "Id".

The process of sending a message has ended. Now, in case the receiver wants to download his messages he makes a request. The receiver sends to the server an encrypted stream consisted of "his nickname" and a new random message "N_3".

Then, the server decrypts the stream and gets the nickname and the "N_3". The server selects the rows, where "To"=nickname, and sends to the receiver two streams. The first one is the encrypted message that was stored in the database and the second one consists of "N_2"(after being decrypted) + "N_3", which is encrypted with recipient's public key.

The receiver gets the streams and verifies the "N_3" to check if the server was the one who replied to him, and then he compares the "N_2" he received from the two streams. In case of successful results, the receiver can read the clear message.

D. Vulnerability on Attacks

As it is referred above, the protocol does not need a safe channel for the message transactions. Of course the presence of a safe channel could make the transactions as safe as the proposed protocol but the point was to use only one algorithm on authentication and on encryption/decryption process. Moreover, the use of a digital signature algorithm could have better results in computations, but on the other hand with this protocol the receiver does not need to know the sender's public key [8].

Let us assume, that T(Trudy) is a third person (attacker) who wants to decrypt an encrypted message. During the first step of the Sending Process, i.e. user's request (1), Trudy cannot read any part of the request (From, To, N_1) because they are encrypted with the server's public key. What Trudy can do is to send a different request with only the same "To"

or "From" and a different "N_1" as she doesn't know the real "N_1". An attack like this will not create problems because the server will not be able to send the appropriate "N_1" to the sender for verification. The sending process will be cancelled. In the same way an attack at the second communication process (2) is not dangerous, because only the server can reply with the correct "N_1".

At the third stage of communication (3) the encrypted message is finally sent. In this case, Trudy can change the "Id" of the message, since it is not encrypted, or the whole encrypted message. In the first case, if the attacker changes the "Id" of the message, the verification of the message to the server will fail and the message will be discarded. If Trudy replaces the whole encrypted message (M, N_2) with another one (M', N_2'), then Bob when he will receive the message, he will compare the correct N_2 that the server sent to him with the N_2'. Because it is unlikely impossible that nonces N_2 and N_2' have the same value, Bob will understand that the message has been altered. This example is a characteristic attack in the integrity of the message. Any kind of an attack at this stage is not harmful.

At the receiving process (4) an attacker can change the request message the receiver sends. What Trudy can do is to send the same "To" with a different nonce "N_3' ". This does not cause any problem, as the attacker cannot read anything from the server's reply. Furthermore, when Bob will try to verify the server by comparing the values of "N_3" (that he created) and "N_3' " (nonce that Trudy send to server) he will not complete the verification process and the message will be deleted.

Finally at the mail receiving process (5) the attacker cannot create any kind of problem, as he doesn't know anything about the nonce "N_3".

During the communication process we have not found any kind of attack that could harm the secrecy of the data sent. We still have not check thoroughly the case of an attack to the server. However we have found that even though someone gets the contents of the server's database he couldn't read any of the mails, as the data stored is already encrypted (except from "From" and "To" fields).

IV. Application

A. Introduction to the Application

It is an application for sending encrypted messages to a number of users who have already created an account to our server. Data is stored in an SQL database at the server with their public keys. Their encrypted messages are also stored in the same database. For the encryption we use Elliptic Curve Cryptography with ElGamal algorithm and for the authentication of the users we use a protocol, both of which have been described above. For the implementation of the application we used JDK 1.4.2_06, MySQL 4.1.7 and Apache Tomcat 5.5.7.

B. Signing up a New Account

The first step is the creation of a new account at the server. The new users have to "Sign up a new account" by typing their nickname, password (this is required for the login), keypassword (this is required for storing and retrieving their private keys), first name, last name, address, phone number and their birthday date. They, also, have to agree with the applications' terms of use and of course retype password and keypassword. The application checks if there is another account with the same nickname. In case of nickname duplication, the new user is informed to type a different one. The password and keypassword fields must have at least 6 characters. If the user has forgotten to complete a number of fields the appropriate message is being displayed. A screenshot of the program is shown in Fig. 2.

Then, the user has to choose a directory for his Inbox. In addition, the user has to create the RSA (1024 bits) and the ECC (200 bits) keys. The RSA keys are used for encrypting and decrypting his private ECC key *nb*. The ECC keys are used for sending and receiving encrypted mails from the server.

The RSA keys are stored in a keystore file (key database) using the keystore class of the keytool of JDK. The user has to choose the directory and the name of the file. This keystore file is used for security reasons. A database with the RSA keys is created with the password entered by the user. The keypassword is used for storing and retrieving the RSA private key. At the time of creation of the account, the RSA public key is used to encrypt the ECC private key. Now the ECC private key cannot be read, and if we need it, we decrypt it with the RSA private key from the database by giving the password and the keypassword.

The reason we have chosen the RSA algorithm and the keystore class is because we wanted to protect the ECC private keys. Multiple users can have an account in the same computer. Optionally, by using RSA for encrypting ECC keys, a user can have access to his ECC keys remotely, using RSA public key cryptography. So, to prevent every ECC private key from being stolen by the other users of the computer or by Internet attacks we need to password protect these keys. We have a double security. Imagine a hacker who wants to get the private key from a computer. He has to know both the password and the keypassword to get the RSA private key in order to decrypt the ECC key.

The ECC public key and the encrypted ECC private key are stored in a property file. The nickname of the user is used for naming this file. The following data is stored in this file:

- Inbox Directory,
- the path of the keystore file,
- the private ECC key encrypted with the RSA public key,
- *a*, *b*, *p*, *G*, *Pb* (public ECC key).

When the RSA and the ECC keys are created, the following data is sent to the server:

- Nick Name,
- First Name,
- Last Name,
- Address,
- Phone Number,
- Birthday Date,
- *a*, *b*, *p*, *G*, *Pb* (public ECC keys).

The above data is being added to the users' table of the server's database. The primary key of this table is the nickname.

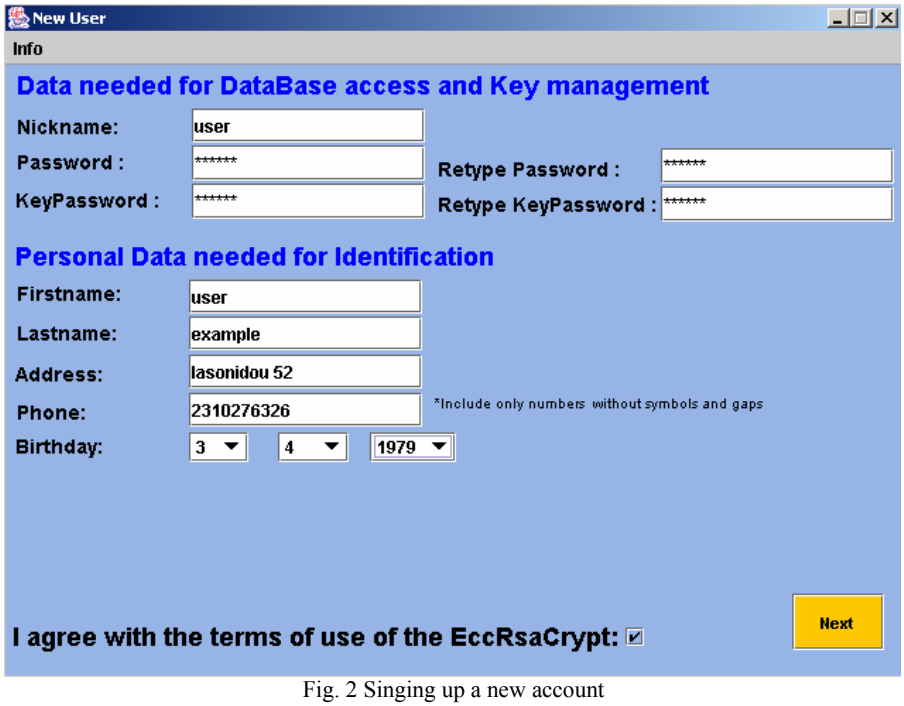

Fig. 2 Singing up a new account

C. *Login*

The users who already have an account, or have just created a new one, can send or receive encrypted mail messages from the server. But before that, they have to login first. They type their nickname and their password. If the nickname/password is incorrect, the keystore file isn't accessible and an error message is being displayed.

D. *Sending – Receiving –Reading Messages*

After a successful login the user has three choices:

1. Send a message to another user,
2. Receive new messages from the server,
3. Read messages from his Inbox.

If the user wants to send a message, he has to write the message and define the name of the recipient. He can either write the nickname of the recipient or, if he doesn't remember it, he has to connect with the server in order to retrieve the nicknames of the current users. Then, he has to retrieve the public key of the recipient and finally send the mail. In order to make the encryption procedure faster the message is being separated in smaller strings. Each string has 19 characters apart from the last one. For example, if the message has 42 characters, it will be divided in three strings with 19, 19, 4 characters correspondingly. Then, each string is being decrypted with the public key of the recipient. The two points that are produced from the encryption of the first string (see section 2) are being concatenated with the two points, which are produced from the encryption of the second string and etc.

For the concatenation specific tags are used. The procedure that follows has been described in section 3.

If the user wants to receive new messages from the server, he has to connect to the server using the authentication protocol, which has been described above. The new messages are stored in the inbox directory of the user.

If the user chooses to read a message, he has to type his keypassword, to get his ECC private key and decrypt the message. For the decryption of the message the inverse procedure is being used. First of all, we separate each pair of points and compute the decrypted messages. Finally, we concatenate the decrypted messages to produce the initial message. Fig. 3 illustrates an example of reading a message.

E. *Communication* between Client, Server and Database

The communication between clients and server is being done using a servlet. The servlet can 'read' the requests from the clients and corresponds accordingly by sending data to clients, communicating with the database, etc. The communication with the SQL database is carried out by the same servlet. For this purpose mysql connector for java is being used. Only localhost queries can be executed in the server's database. This is for security reasons. So, the database is accessible only from the server.

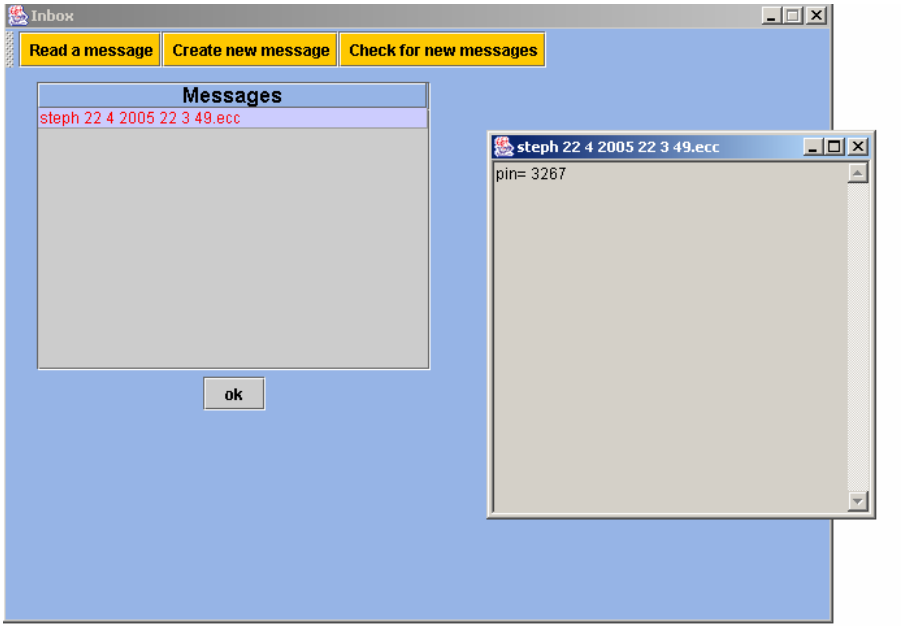

Fig. 3 Reading a message from Inbox

V. Conclusions

Elliptic Curve Cryptosystems are one of the best ways of sending – receiving encrypted messages or keeping encrypted data. The encrypted messages – data can only be read from one person only, the owner of the private key. If an attacker gets the encrypted data, he will need years in order to decrypt the message and the information he will get will be useless. So we have ultimate protection.

It is proved that the authentication protocol, which has been developed by the authors, is invulnerable. ElGamal's algorithm success is based on the difficulty of the discrete logarithm problem, which is still remaining as one of the most difficult problems to be solved. Moreover there is no use of symmetric algorithms (eg DES) and one way hash-functions (eg MD5) which have been proved to be vulnerable in brute-force attacks when small messages are being transmitted. Furthermore, the application that is based on this protocol works quite well. The sending and receiving processes take only a few seconds to be completed, with a 56 Kbps modem and a 500 MHz Pentium III server. Furthermore, we found out that the bit-length of the ECC public keys we use in the application is the appropriate. The keys are big enough to offer a high-level security but not as big as the keys of other algorithms (e.g. RSA) for faster communication.

VI. Future Work

We are thinking of making a statistical analysis on how often someone sends or receives an encrypted message, how often someone tries to attack our server and the success ratio of message transactions. Moreover we will make an analysis on what kind of information the encrypted messages contain (such as credit cards number, information about stocks, PIN codes etc.).

We could also develop a number of other protocols for obtaining the authentication of the users during the procedure of sending and receiving encrypted messages. Then, we could easily compare the computation speed of the implemented protocols and of course their vulnerability.

Furthermore, we believe that the following characteristic should be added to our application. The user could specify which area of the message would be sent as plain text and which area would be encrypted with the public key of the recipient.

In addition, each user will have the advantage of joining in a group of users. Each group will have its own public and private keys. So, if someone wants to send a message to the users of the group he will have to send it only to the 'group'.

Finally, we are going to expand our application for sending encrypted files attached to the messages. The following algorithm can be implemented for this purpose. The sender creates a random integer k. Then k can be used as a password for this file using a hash function. Finally the encrypted file and the integer k (encrypted with the public key of the recipient) are sent to the recipient.

References

[1] N. Koblitz, A. Menezes and S. Vanstone, The state of elliptic curve cryptography. *Designs, Codes and Cryptography*, 19 (2000), 173-193.

[2] N. Koblitz, Elliptic curve cryptosystems. *Mathematics of Computation*, 48 (1987), 203-209.

[3] V.Miller, Uses of elliptic curves in Cryptography. H.C.Williams, (ed.) Advances in Cryptology-CRYPTO 85, Proceedings, *Lecture Notes in Computer Science*, No 218 (1985), 417-426, Springer-Verlag.

[4] M. Aydos, E. Savas, and C. K. Koc , Implementing Network Security Protocols based on Elliptic Curve Cryptography, Proceedings of the Fourth Symposium on Computer Networks, pages 130-139, Istanbul, Turkey, May 20-21, 1999.

[5] L. C. Washington, *Elliptic curves Number Theory and Cryptography*, Chapman&Hall / CRC, 2003.

[6] M. Rosing, *Implementing Elliptic Curve Cryptography*, Manning, 1999.

[7] D. Hankerson , A.Menezes, and S.Vastone, *Guide to Elliptic curve cryptography*, Spinger-Verlag, 2003.

[8] C. Boyd and A. Mathuria, *Protocols for authentication and Key Establishment*, Springer-Verlag, 2003.