A Certification Authority for Elliptic Curve X.509v3 Certificates

Maria-Dolores Cano, Ruben Toledo-Valera, Fernando Cerdan
Dept of Information Technologies & Communications
Technical University of Cartagena (UPCT)
30202 Cartagena, Spain
mdolores.cano@upct.es

***Abstract*— Wireless networks are more and more common in current communications networks. Nevertheless, wireless communications entail a big concern: security. The use of X.509v3 certificates to carry out authentication tasks is an approach to improve security. These certificates are usually employed with the RSA algorithm. Elliptic Curve Cryptography (ECC) is a cryptographic technique eminently suited for small devices, like those used in wireless communications, and is gaining momentum. The main advantage of ECC versus RSA is that for the same level of security it requires a much sorter key length. The purpose of this work is to design and implement a free open-source Certification Authority able to issue X.509v3 certificates using ECC. This research is an implementation study on free open-source tools to issue digital certificates using ECC. Moreover, it contributes to the development of free open-source tools for network security based on ECC. The result of this research may assist organizations to increase their security level in wireless devices and networks, in a costless way, by including authentication techniques based on ECC digital certificates.**

Keywords- digital certificates; elliptic curve cryptography; security; wireless communications.

I. INTRODUCTION

Wireless networks have suffered a dramatic increase in recent years. Wireless technology is more and more present in our society and millions of wireless equipment are sold every year. However, one of the major concerns about wireless communications is security. RSA is the most common method employed in public key cryptography, for instance in X.509 digital certificates. These certificates are oriented to verify the identity of a person or an entity. However, new concerns are rising about the security of 1024-bit RSA [1].

Elliptic Curve Cryptography (ECC) is an innovative cryptographic technique. Its security resides in the same problem as RSA or Diffie-Hellman algorithms, but instead of using integers as symbols of the alphabet to be ciphered, it uses points in a mathematical object called elliptic curve. The real ECC potential is that, with a much smaller key length, it achieves the same security level as other proposals. Therefore, ECC presents some key attributes truly important in scenarios where the following resources are limited: processing power, storage space, bandwidth and power consumption [2] [3]. There are even some organizations working towards ECC standardization (IEEE, IETF, ISO, etc.), and leading enterprises developing new ECC products. Nevertheless, to favor the widespread use of ECC it is also essential promoting free open-source ECC tools.

In this paper we introduce a free open-source Certification Authority (CA) for ECC X.509v3 digital certificates. The CA we propose is able to generate its own root certificate and to issue clients' certificates. We also develop the software a client requires to create a certificate request. This certificate request is the one that the CA should sign after some verification steps. The tool we propose is mainly oriented to environments with limited resources. As it will be shown in next sections, its advantages are clearly noticeable.

The rest of this paper is organized as follows. In section 2, we give a brief overview about Elliptic Curve Cryptography, and the ECC mechanisms we use for the new X.509v3 ECC digital certificates. In section 3, we explain the design of the certification tool. In section 4, we describe the ECC CA working scenario. Section 5 shows and discusses the experimental implementation. The paper ends with the most important concluding remarks in section 6.

II. OVERVIEW OF ELLIPTIC CURVE CRYPTOGRAPHY

Public key (asymmetric) cryptography uses two keys (a private key and a public key), differing from private key (symmetric) cryptography, where there must be a shared secret key. Elliptic Curve Cryptography was discovered in 1985 by V. Miller [4] as an alternative method for public key cryptography. At that time, it was very difficult to perform the necessary calculations. With time, implementations were much more efficient, what allowed the performance of elliptic curve mathematics to take the same amount of time as implementations of integer factoring schemes for the same number of bits. This, in its turn, implies a reduction in cost, size, and processing time because elliptic curves require fewer bits for the same security level.

An elliptic curve is described by a cube equation, similar to those used to calculate an elliptic circumference. Usually, the cube equation of an elliptic curve is indicated by (1), where a, b, c, d, and e are usually real numbers that comply with some condition. Then, the elliptic curve is defined by the points (x,y) that satisfy this equation. The addition operation can be defined for an elliptic curve, together with the element point at infinity

Third International Conference on Networking and Services(ICNS'07)
0-7695-2858-9/07 $20.00 © 2007 IEEE

0. This addition operation fulfills the associative and commutative properties.

$$y^2 + axy + by = x^3 + cx^2 + dx + e\,. \quad (1)$$

Elliptic curves used in cryptography are defined over two types of finite fields: fields of odd characteristics (F_p, where *p* is a large prime number), and fields of characteristics two (F_{2^m}). For the sake of simplicity we focus on F_p. Observe that the field F_p only employs the numbers from 0 to (*p*-1), and all computations end by taking the remainder on division by *p*. In particular, cryptography is interested in elliptic curve groups over F_p. If we chose two positive integers, *a* and *b*, smaller than *p* such that (2) is true, then $E_p(a,b)$ denotes the elliptic curve group in F_p, whose elements (*x*,*y*) are pairs of positive integers smaller than *p* that satisfy the elliptic curve equation (3).

$$4a^3 + 27b^2 \pmod{p} \neq 0\,. \quad (2)$$

$$y^2 \pmod{p} = x^3 + ax + b \pmod{p}\,. \quad (3)$$

To create a crypto system using elliptic curves is necessary to find a difficult problem such factorizing the product of two prime numbers or calculating a discrete logarithm. Consider the equation $P = k \cdot G$, where *P* and *G* are points belonging to $E_p(a,b)$, and *k* is smaller than *p*. It is quite easy to assess *P* given *k* and G, but it is very complex to calculate *k* given *P* and *G*. This is called the Elliptic Curve Discrete Logarithm Problem (ECDLP). In fact, the *G* point is called the generator point. The criterion to select G is as follows: the smallest value of *n* such that $n \cdot G = 0$ must be a large prime number. Most of the elliptic curve cryptographic methods are related to the discrete logarithm schemes, which were originally formulated for usual modular arithmetic.

In order to use ECC, all parties must agree on all the elements defining the elliptic curve, that is, all parties should know the domain parameters. For the field F_p, the domain parameters are: the prime number *p*, constants *a* and *b*, the generator point *G*, and the integer *n*. The generation of these domain parameters is not straightforward. Several standards bodies publish domain parameters of elliptic curves [5] [6] [7].

Next, we briefly explain the two ECC algorithms that we use in this work to generate the ECC X.509v3 certificates: ECDSA to sign a digital certificate, and ECIES to generate the public key included in a digital certificate.

A. ECIES

The Elliptic Curve Integrated Encryption Scheme (ECIES), also known as Elliptic Curve Augmented Encryption Scheme or Elliptic Curve Encryption Scheme, is an ECC public-key encryption technique. ECIES is based on the Diffie-Hellman method. Let us explain briefly how it works [6] [7] [8].

First, one entity (e.g. *A*) should establish what key derivation function (KDF) to use (e.g. ANSI-X9.63-KDF with SHA-1 option [7], IKEv2-KDF [10] or TLS-KDF [11]). A KDF is used to derive keying data from a shared secret octet string. Entity *A* should also select: the MAC (Message Authentication Code) scheme (e.g., HMAC-SHA-1-160 with 160-bit keys, HMAC-SHA-1-80 with 160-bit keys, etc.), the symmetric encryption scheme (e.g. AES), and any option involved in them. *A* should decide on whether to use the standard elliptic curve Diffie-Hellman primitive or the elliptic curve cofactor Diffie-Hellman. In addition, *A* should establish the elliptic curve domain parameters (*a*, *b*, *G*, *n*, etc.) at the desired security level. Next, the other entity (e.g. *B*) should obtain in an authentic manner the selections made by *A*.

After that, *A* should set up an elliptic curve key pair associated with the elliptic curve domain parameters determined during the setup procedure. Let's call K_{PA} to the *A*'s public key and K_{pA} to the *A*'s private key. K_{pA} is an integer chosen randomly in the range [1, *n*-1]. K_{PA} is calculated as indicated by expression (4):

$$K_{PA} = K_{pA} \cdot G\,. \quad (4)$$

Then, entity *B* should obtain in an authentic way the elliptic curve public key selected by *A*, i.e. K_{PA}. From now on, *B* (*A*) should encrypt (decrypt) messages using the keys and parameters established previously. For instance, if *B* wants to send a ciphered message to *A*, then *B* does the following actions:

- *B* generates a random number $r \in [1, n\text{-}1]$ and assesses $R = r \cdot G$.
- *B* obtains a shared secret $K_S = r \cdot K_{PA}$. Note that *R* and K_S are points in the elliptic curve.
- *B* uses the KDF to derive a symmetric encryption and MAC keys, K_E and K_M respectively.
- *B* ciphers the message using K_E and the symmetric encryption scheme selected during the setup phase.
- *B* computes the tag of the ciphered message using K_M.
- The decryption process is straightforward knowing the Diffie-Hellman procedure.

In this work, we use ECIES to generate the public key of an entity, which can be used for ciphering in later services. The legitimacy of this public key is guaranteed by the digital certificate ECC X.509v3 that our proposed Certification Authority issues. The ECIES parameters that we have selected in our implementation are shown below in Table 1.

B. ECDSA

Elliptic Curve Digital Signature Algorithm (ECDSA) is a variant of the Digital Signature Algorithm (DSA) that operates with elliptic curves. Signature schemes are designed to be used

when an entity *A* wants to send a message *M* to an entity *B* in an authenticated way, and *B* wants to verify the authenticity of *M*. ECDSA acts as follows [7] [8] [12].

First, an entity *A* should select what hash function to use (e.g. SHA). Moreover, *A* should establish the curve domain parameters (*a*, *b*, *G*, *n*, etc.) at the desire security level. Entity *B* should get in an authentic manner the selections made by *A*.

Next, *A* and *B* should perform a key deployment procedure to be prepared to use ECDSA. *A* should set up an elliptic curve key pair associated with the elliptic curve domain parameters agreed on the setup procedure to use with ECDSA. Let's call K_{PA} to the *A*'s public key, as shown in (4), and K_{pA} to the *A*'s private key (randomly selected in [1, *n*-1]). *B* should obtain the elliptic curve public key selected by *A*, i.e. K_{PA}.

To sign a message *M*, *A* should proceed according to the following steps:

- *A* applies the hash function to the message, and derives an integer *e* from the obtained hash.
- *A* selects a random integer *k* in the range [1, *n*-1], and calculates $K = k \cdot G = (x_A, y_A)$.
- *A* assesses $r = x_A (\mathrm{mod}\, n)$.
- *A* calculates $s = k^{-1}(e + K_{pA} \cdot r)(\mathrm{mod}\, n)$.
- The signature is the pair (*r*, *s*).

To verify the signature, the entity *B* should proceed as follows:

- *B* checks if *r* and *s* are integers, otherwise the signature is not valid.
- *B* applies the hash function to the message, and derives an integer *e* from the obtained hash.
- *B* calculates $u_1 = e \cdot s^{-1} (\mathrm{mod}\, n)$ and $u_2 = r \cdot s^{-1} (\mathrm{mod}\, n)$.
- *B* computes $K = (x_A, y_A) = u_1 \cdot G + u_2 \cdot K_{PA}$.
- The signature is valid if $x_A = r (\mathrm{mod}\, n)$.

In this work, the Certification Authority uses ECDSA to sign an ECC X.509v3 digital certificate containing an ECIES public key, thus verifying the authenticity of the public key and its owner.

TABLE I. ELLIPTIC CURVE DOMAIN PARAMETERS.

Parameter	Value
a	0x7fffffffffffffffffffffff7fffffffff8000000000 07fffffffffc
b	0x6b016c3bdcf18941d0d654921475ca71a9db 2fb27d1d37796185c2942c0a
G	0x020ffa963cdca8816ccc33b8642bedf905c3d3 58573d3f27fbbd3b3cb9aaaf
n	883423532389192164791648750360308884807550341691627752275345424702807307
p	883423532389192164791648750360308885314476597252960362792450860609699839

III. ECC CERTIFICATION AUTHORITY DESIGN

We choose Java as programming language due to its platform independence. After a searching phase, we leaned on the open source library BouncyCastle [13] to write the code. Next, we give details about the ECC CA design.

Our work can be divided into three blocks: classes to create the CA, classes to create the certificate request by the client, and classes so that the CA can sign the certificate request. In addition, we define three classes (included in Fig. 1) that are shared by all blocks:

- *KeyGeneration* is in charge of generating an elliptic curve key pair. It uses the ECIES scheme specified in ANSIX9.63 and IEEE P1363.
- *X509Subject* takes the component of the client data, splits it into its minimum units, and composes it again to eliminate possible misspellings.
- *CertificateUtils* generates *.cer* certificates (certificates signed by the CA, i.e., the identity of the user has been verified), and *.der* certificates (certificates that have not been signed yet, i.e. a client certificate request).

In the first block, classes to make the CA, we define the class CAcertEC shown in Fig. 2. Its main task is to generate the X.509v3 root certificate and the PKCS#12 (Public Key Cryptography Standard, PKCS) with the corresponding CA's private key. A root certificate is a certificate that contains the public key of a CA. Clients can trust a CA only if a copy of the CA root certificate is in its trusted root certificate store. Moreover, the CA public key included in the CA root certificate is needed to verify the validity of any certificate that the CA issues. PKCS is a set of standard protocols to exchange secure information on the Internet using a public key infrastructure. PKCS#12 is a standard that specifies a portable format for storing a user's private key.

In the second block, classes to create the certificate request by the client, we characterize the classes illustrated in Fig. 3. They involve the following tasks:

- *GraphicClient* launches an applet that the client uses to fill in the data necessary for the certificate request.
- *Manager* captures the applet events.

KeyGeneration
Static KeyGeneration(int): KeyPair

X509Subject
Static decodeX509Subject(string) : string

CertificateUtils
Static X509Certificate genCert
Static createCertMaster(PublicKey, PrivateKey, string, int, string): X509Certificate Static createCertDER(PublicKey, PrivateKey, string, string): PKCS10CertificateRequest

Figure 1. Shared classes.

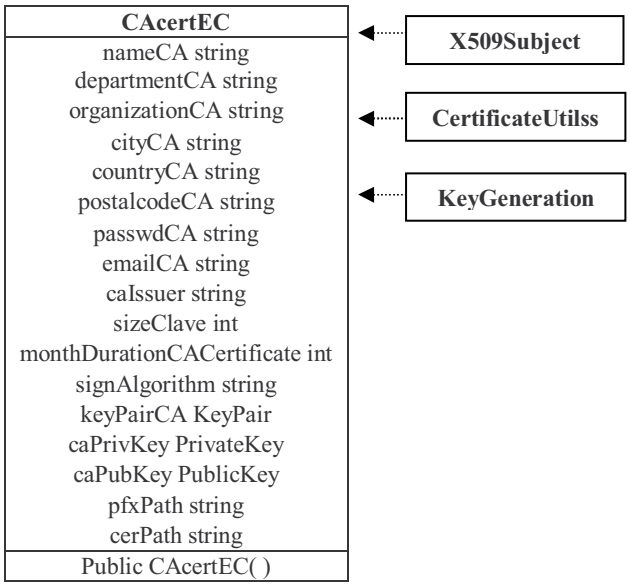

Figure 2. Classes to create the CA.

- *UserCertEC_DER* generates a *.der* certificate (a certificate request). This is a certificate without a signature, hence, not valid yet. The *.der* certificate should be sent to the CA for signing. This class also generates a PKCS#12 to store the private key.
- *MinimumClient* takes the client *.der* certificate and sends it to the CA.

In the last block, classes so that the CA signs the certificate request, we create the classes included in Fig. 4. The goals of these classes are:

- *MinimumServer*, the CA is listening, waiting for a client request. When a client connects to the CA, the CA checks if the client is authorized to demand the service.

  If the client is authorized then the CA signs the client certificate request using the *DER2CER* class. Afterwards, the CA returns the signed certificate (*.cer*), ready for use, back to the client. The CA keeps waiting for new client requests.
- *DER2CER*, this class firstly edits the client certificate request (PKCS#10), and adds new data such as key length, certificate serial number, period of validity (valid from-to), and signature algorithm. In our case, the CA uses the ECDSAwithSHA-1 (Elliptic Curve Digital Signature Algorithm with Secure Hash Algorithm 1) to sign certificates. *DER2CER* needs to know the key (usually known as *superkey*) to access the CA secret key, which is located in the file *server.pfx*. It is necessary to know the CA secret key otherwise the CA would not be able to sign the client certificate. Afterwards, the client certificate (*.cer*) is created with *CertificateUtils*.

IV. ECC Certification Authority protocol approach

In this section we explain the general procedure to obtain an ECC X.509v3 certificate (see Fig. 5). First of all, the client asks the CA to issue a certificate (*step 1*). In further services, the client could be authenticated with this certificate, or the public key included in the certificate could be used for ciphering. Then, the CA sends its root certificate (*serverCa.cer*) and the software needed by the client to generate the certificate request (*step 2*). The client installs the CA root certificate in its trusted store. Although not implemented, the software could also be signed so that the client can trust in it.

The client executes the software to generate a key pair and a certificate request (*client.der*) (*step 3*). The client sends its certificate request to the CA (*step 4*). The CA receives the certificate request and some information from the user to validate him/her. Different approaches can be taken to decide if a user is authorized or not to ask for an ECC X.509v3 certificate and verify his/her identity. For instance, if this system were used in a pre-paid hotspot (e.g. airport, hotel, etc.), the user could send a code number to be validated. This code could be obtained when a pre-paid card is bought. In other environments, like a small company, the user validation could be done in person.

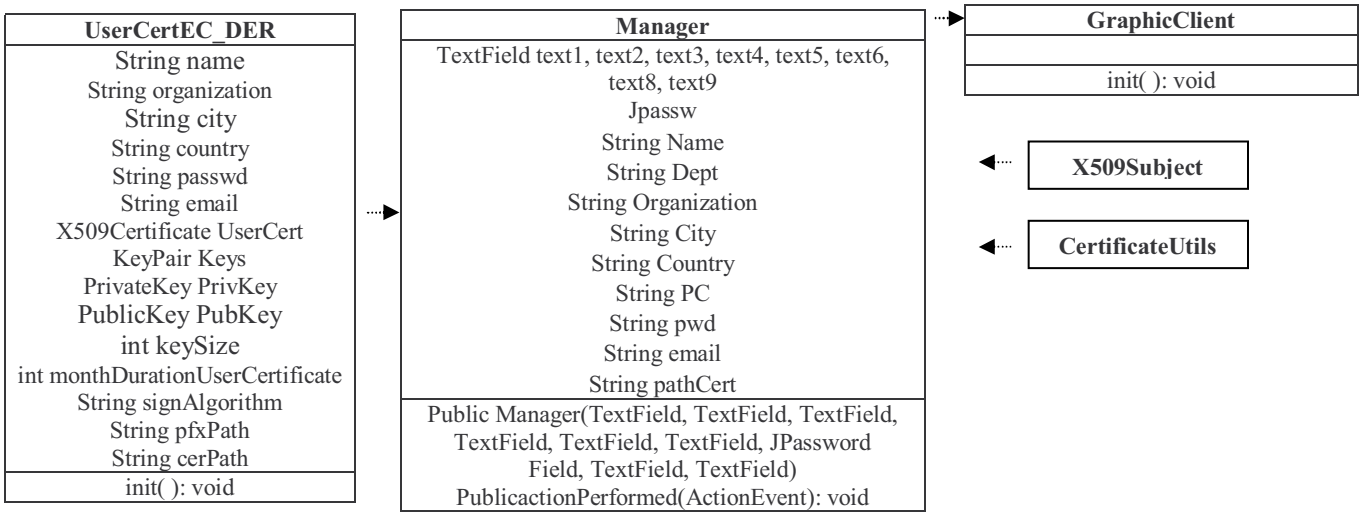

Figure 3. Classes to generate a certificate request.

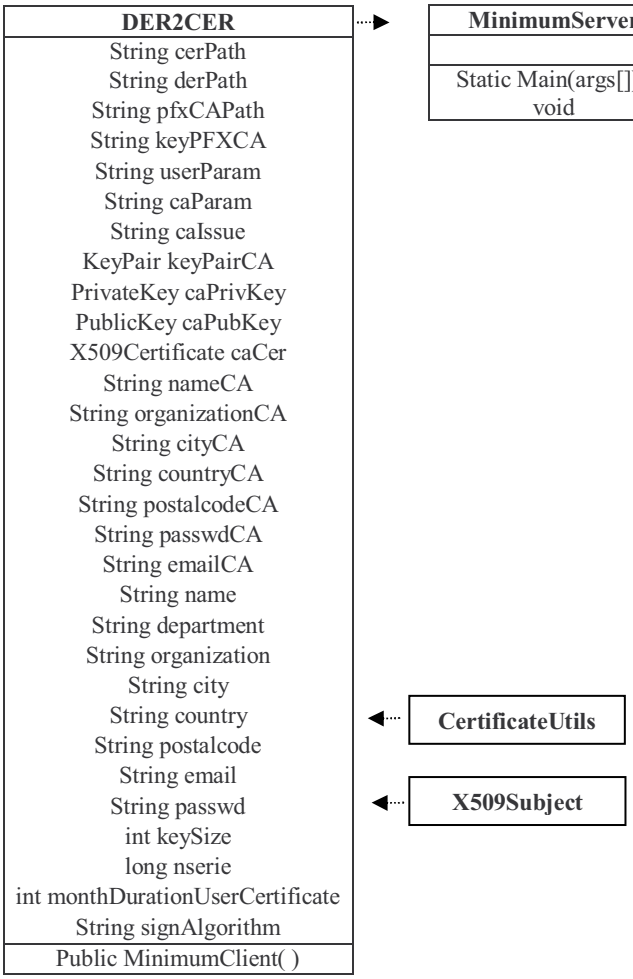

Figure 4. Classes to sign the client certificate request.

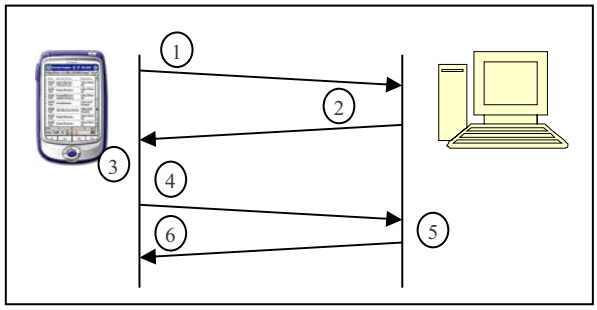

Figure 5. General procedure.

If the user is authorized to demand this service, and his/her identity has been confirmed, the CA processes the certificate request, signing it with its private key and sending the final X.509v3 certificate (*client.cer*) to the client. Using this certificate, the client can be authenticated for later network services. Note that the entire process is transparent from the user side.

V. ECC Certification Authority Implementation

In this section we present the implementation of our free open-source ECC Certification Authority. The implementation can be downloaded from [14]. For simplicity, we follow the same nomenclature (step 1, step 2, etc.) than we used in the previous section.

At first, the server is waiting for client requests. In our implementation, we assume that the client already has the software to generate the certification request (steps 1 and 2 in Fig. 5). To make the rest of the process easier, we have included a web page where the user can introduce some of the data needed for his/her certificate (e.g.: name, affiliation, etc.). Therefore, in the third step the client loads the web page (Fig. 6). When the form is filled in, the client clicks the "send" button. At that time, the client software creates a key store PKCS#12, where the private ECC key is stored and the certificate request *client.der* is generated. The client automatically sends the *client.der* certificate to the Certification Authority (step 4 in Fig.5).

The CA receives the certificate request *client.der*, the name of the client host, and its IP address. Received data belonging to *client.der* is shown in the screen. We assume that the client has proper access to the service, so the CA should only issue the final client certificate. Once the *client.cer* certificate is ready, the CA sends it back to the client.

At this moment, the client has three files: *client.cer*, *client.der,* and *client.pfx*. The file *client.cer* is the X.509v3 ECC certificate. The file *client.der* is the certificate request that can be deleted. The file *client.pfx* is the key store, where the client's private key is kept. These three files are located in the directory previously indicated in the form, in the box "path to store the certificate" (Fig. 6).

In Fig. 7 and Fig. 8, we observe the details of the certificate (*client.cer*). The certificate is issued to "Paco" by the Certification Authority "CA4ec" and is valid from 10/12/2005 (following the date format dd/mm/yy) to 09/05/2006. We see in Fig. 8 that the signature algorithm corresponds to the OID (Object Identifier) 1.2.840.10045.4.1. This OID matches the ECDSAwithSHA1 algorithm. From Fig. 8, we observe that the public key algorithm is ECIES, identified by the OID 1.2.840.10045.2.1. OIDs can be checked in [15].

Regarding the certification path, we note from Fig. 7 and Fig. 9 that the certificate appears as not valid (red cross in Fig. 9). This is due to the fact that Windows XP operating system does not include yet any library (or module) to use ECDSA. That is, it does not understand yet the algorithms ECIES or ECDSA. Consequently, it is not able to verify the integrity of the certificate.

VI. Conclusions

In this paper, we propose, design, and implement a free open-source Certification Authority that generates X.509v3 certificates by using elliptic curve cryptography. We explain the classes needed to create the Certification Authority, the classes needed to create a client certificate request, and the classes to sign and generate the final validated client certificate. We also show a real implementation. With the use of this type of application, we aim to help to spread the use of elliptic curve cryptography. Our implementation is notably useful for small wireless devices with processing power, storage space, or power consumption restrictions.

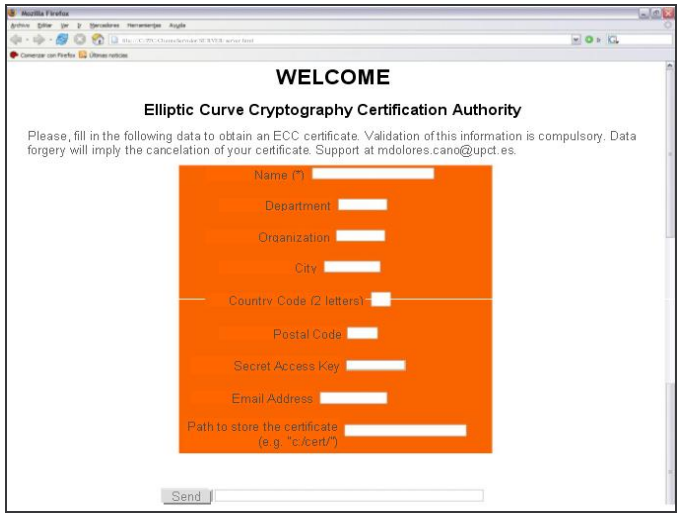

Figure 6. Client web page. The user can fill in: name, surname, department, organization, city, country code, postal code, secret key to access his/her private key, email address, and the path to store the certificate.

Figure 7. Certificate information.

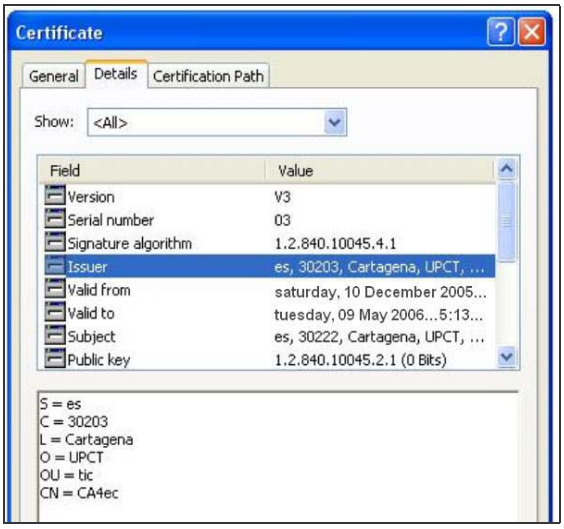

Figure 8. Details of the certificate.

Figure 9. Certificate path.

REFERENCES

[1] S. Vanstone, "Next generation security for wireless: elliptic curve cryptography", Computers & Security, Vol. 22, No. 5, pp. 412-415, 2003.

[2] N. R. Potlapally, S. Ravi, A. Raghunathan, N. K. Jha, "A study of the energy consumption characteristics of cryptographic algorithms and security protocols", IEEE Transactions on Mobile Computing, Vol. 5, No. 2, pp.128-143, 2005.

[3] W. Rao, Q. Gan, "The performance analysis of two digital signatures schemes based on secure charging protocol", Proc. International Conference on Wireless Communications, Networking, and Mobile Computing, Vol. 2, pp. 1180-1182, September 2005.

[4] V. S. Miller, "Use of Elliptic Curves in Cryptography", Proc. CRYPTO'85, Springer-Verlag, New York, pp. 417-426, 1986.

[5] National Institute of Standards. FIPS-PUB 186-2. "Recommended Elliptic Curves for Federal Government Use", 1999. Available online <http://csrc.nist.gov/CryptoToolkit/dss/ecdsa/NISTReCur.pdf>. Last accessed 3rd March 3rd, 2006.

[6] Certicom, "Standards for efficient cryptography. Sec2:Recommended Elliptic Curve Domain Parameters", Released Standard Version 1.0, 2000. Available online <http://www.secg.org> . Last accessed March 3rd, 2006.

[7] ANSI X9.63, "Public-Key Cryptography for the Financial Services Industry, Key Agreement and Key Transport Using Elliptic Curve Cryptography", 2001.

[8] D. R. Brown, "Standards for efficient cryptography. Sec1: Elliptic Curve Cryptography", Released Standard Version 1.0 and Working Draft v1.5, 2005. Available online <http://www.secg.org>. Last accessed March 3rd, 2006.

[9] IEEE1363 Working Group. IEEE Std P1363a-2004 (Amendment to IEEE Std P1363-2000). IEEE Standard Specifications for Public-Key Cryptography – Amendment 1: Additional Techniques, 2004.

[10] C. Kaufman, "Internet Key Exchange (IKEv2) Protocol", Internet Draft, 2005.

[11] V. Gupta, S. Blake-Wilson, B. Möller, C. Hawk, N. Bolyard, "ECC Cipher suites for TLS", Internet Draft, 2004.

[12] ANSI X9.62. (2005). Public-Key Cryptography for the Financial Services Industry, the Elliptic curve Digital Signature Algorithm (ECDSA).

[13] The Legion of the BouncyCastle. Available online <http://www.bouncycastle.org>. Last accesed March 3rd, 2006.

[14] M. D. Cano, R. Toledo Valera, F. Cerdan. Email corresponding author for code donwload.

[15] ASN.1 Information Site. OID Repository, 2006. Available online <http://asn1.elibel.tm.fr/oid/index.htm>. Last accesed March 3rd, 2006.