A Public-Key Cryptography Tool for Personal Use
A Real-world Implementation of ECC for Secure File Exchange

Luigi Maria Bottasso
R&T Department, AgustaWestland S.p.A., via Giovanni Agusta 520, Cascina Costa di Samarate (VA), Italy

Keywords: Elliptic Curve Cryptography, Direct Embedding Schemes, Public Key Infrastructure, Social Networking.

Abstract: A new library of modular arithmetic and cryptographic functions was coded, and then used for the development of a crypto tool. We present the architecture and functionality of a hybrid ECC-AES cryptosystem which can be quickly deployed even in absence of Public Key Infrastructures and associated Certification Authorities. The tool was conceived for use in combination with readily available resources, e.g. email and possibly social networks. It allows secure exchange of files with associated ECDSA digital signature, providing the user with substantial flexibility and control of the security settings. Established protocols were used in an original way, notably exploiting direct embedding of the AES session key into an elliptic curve. The code has been developed in C++ entirely from scratch, with no use of pre-existing libraries. The implementation is associated with a web site http://www.elcrypto.com, www.elcrypto.com aimed at promoting the benefits of Elliptic Curve Cryptography.

1 INTRODUCTION

Businesses around the world rely on well-established secure communication solutions, notably the concept of Public Key Infrastructures applied to Virtual Private Networks (VPNs). VPNs allow employees access to their company's intranet while being outside of the office, preserving the security features of a private intranet.

But what happens when a corporate computer with a VPN certificate is not available? There are indeed circumstances where subjects need to communicate securely but do not have a PKI available.

We perceive there is at least a niche need for a quickly deployable form of public-key cryptography, giving users more control and less dependence on third parties. A very interesting concept is the one suggested by Gruhn (V. Gruhn, 2007) where social networks are described as a possible new vehicle for the dissemination and authentication of public keys associated with users.

If public keys can be somehow authenticated by readily available web infrastructure (not complex and maintenance-intensive ones like PKIs, VPNs etc.) then an encryption tool can be used for personal exchange of information between any two subjects, just by exchanging an encrypted document as an email attachment or by loading it on the personal page of a social network. In this paper we describe the basic functionality and architecture of an Elliptic Curve Cryptography (ECC) tool which we developed for the above purpose.

In order to maximize the design flexibility a choice was made to start from a blank sheet of paper instead of making use of open source libraries. We thus developed and implemented all basic building blocks, starting from big-number arithmetic functions, modular arithmetic, primality tests, elliptic algebra etc.

The same applies for the AES block cipher and SHA-2 hashing libraries; the overall code diagram is graphically illustrated in Figure 1.

A distinctive feature of the instrument is the extensive control provided to the user for the selection of the security level and the associated curve parameters. This should be seen in contrast to current practices which privilege transparency to the user.

At least certain categories of users may value a closer involvement in the decision process. For example, the tool offers the ability to define a custom elliptic curve of prime order with (in principle) arbitrary many bits of security, or to change it periodically; the same applies to the private and public keys which can be re-generated at will, e.g. if a risk is perceived.

Alternatively, users may just opt for one of the standard NIST-recommended curves, associated with

DOI: 10.5220/0005511801940201
In *Proceedings of the 12th International Conference on Security and Cryptography* (SECRYPT-2015), pages 194-201
ISBN: 978-989-758-117-5

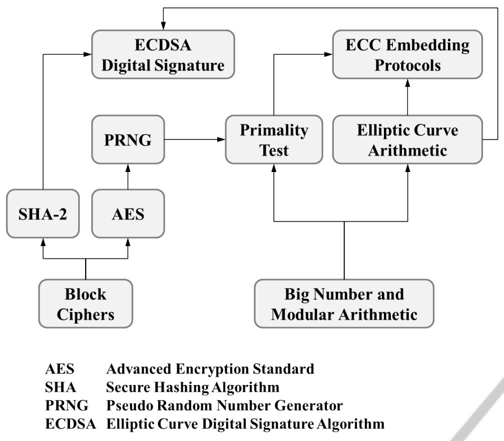
ECDSA Digital Signature
ECC Embedding Protocols
PRNG
Primality Test
Elliptic Curve Arithmetic
SHA-2
AES
Block Ciphers
Big Number and Modular Arithmetic
AES Advanced Encryption Standard
SHA Secure Hashing Algorithm
PRNG Pseudo Random Number Generator
ECDSA Elliptic Curve Digital Signature Algorithm

Figure 1: Code building blocks and their relations.

an adequate AES key length.

Elliptic Curve Cryptography was chosen mainly due to its historically higher resilience to attacks compared to RSA. This feature ensures that the recommended key lengths will suffice for the foreseeable future, without the need for periodical adjustments and updates.

The other major advantage of ECC, i.e. its high security per bit and the resulting short keys, is indeed a practical feature even in the present implementation but does not necessarily translate into significant speed benefits. This is because the bulk encryption is performed with AES and ECC is only used to encrypt relatively small amounts of keying material.

In the following, we take the opportunity to describe some relevant features and design choices which were implemented in the tool.

2 SYSTEM FUNCTIONALITY

The functionality is based on a hybrid public/private key approach, whereby bulk data encryption is provided through a block cipher (AES 128, 192 or 256 bit) while the private session key is encrypted through ECC. The encrypted data bytes are hashed through SHA-512 (NIST, 2002) and signed with ECDSA.

Figures 2 and 3 show the flow diagrams for both the encryption and the decryption processes.

Complementary data (keys, signature, padding) are appended to the encrypted file bytes, thus forming a single payload.

A visual "touch" is added by transforming the resulting object into a picture (bitmap); this is done through the addition of the BMP header bytes and some padding at the end. The latter may be required because a BMP file shall have a rectangular shape with rows of equal length.

Figure 4 illustrates the byte structure of the encrypted payload. We found convenient to store the size of the padding into byte positions 6 to 9 of the BMP header, which are not normally used, as shown in Figure 5.

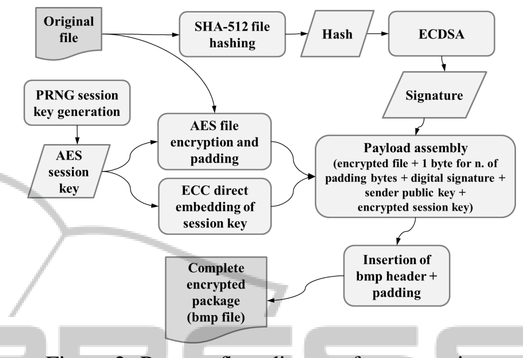
Original file
SHA-512 file hashing
Hash
ECDSA
PRNG session key generation
Signature
AES file encryption and padding
AES session key
Payload assembly (encrypted file + 1 byte for n. of padding bytes + digital signature + sender public key + encrypted session key)
ECC direct embedding of session key
Insertion of bmp header + padding
Complete encrypted package (bmp file)

Figure 2: Process flow diagram for encryption.

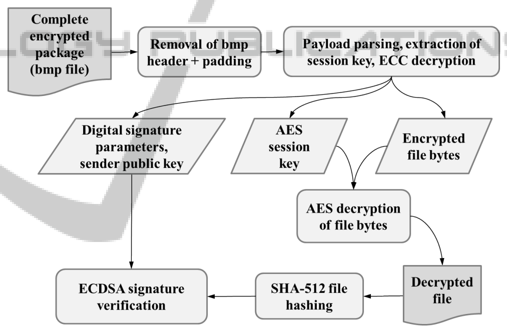
Complete encrypted package (bmp file)
Removal of bmp header + padding
Payload parsing, extraction of session key, ECC decryption
Digital signature parameters, sender public key
AES session key
Encrypted file bytes
AES decryption of file bytes
ECDSA signature verification
SHA-512 file hashing
Decrypted file

Figure 3: Process flow diagram for decryption.

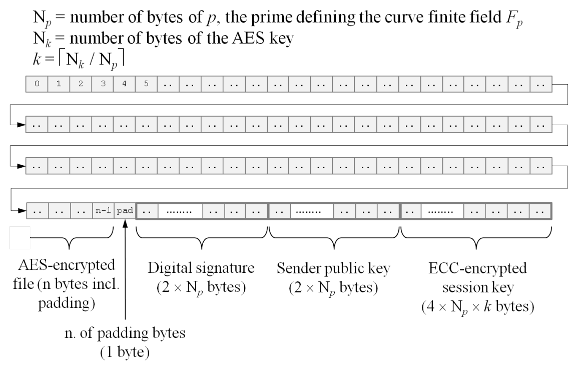
N_p = number of bytes of p, the prime defining the curve finite field F_p
N_k = number of bytes of the AES key
$k = \lceil N_k / N_p \rceil$
0 1 2 3 4 5
n-1 pad
AES-encrypted file (n bytes incl. padding)
Digital signature ($2 \times N_p$ bytes)
Sender public key ($2 \times N_p$ bytes)
ECC-encrypted session key ($4 \times N_p \times k$ bytes)
n. of padding bytes (1 byte)

Figure 4: Encrypted file: assembled structure.

Figures 6, 7, 8 show respectively: how a 500 KB PowerPoint file appears when viewed as a bitmap, the resulting image of the encrypted file and the key/signature data payload embedded into the picture.

Note that the key and signature data appear in the top right corner of the image in Figure 8 because the bitmap file bytes are shown in reverse order.

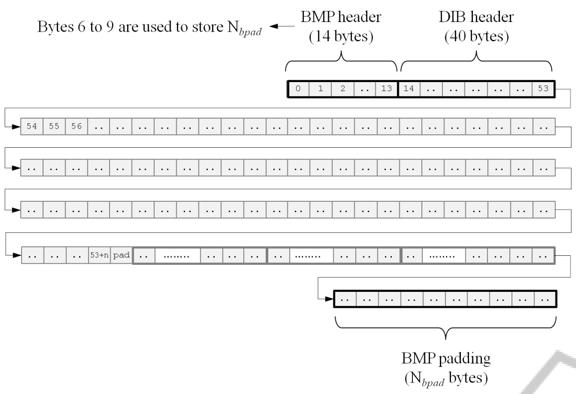
Bytes 6 to 9 are used to store N_{bpad}
BMP header (14 bytes)
DIB header (40 bytes)
BMP padding (N_{bpad} bytes)

Figure 5: BMP headers and padding added to the encrypted payload.

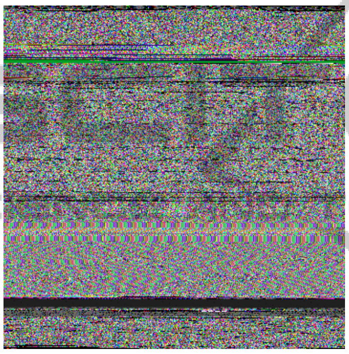

Figure 6: How a 500 KB pptx file looks like when viewed as a bitmap.

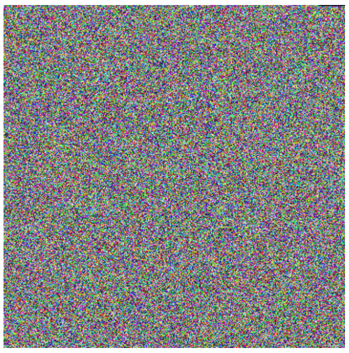

Figure 7: The same file of Figure 6, now in encrypted format and visualized as a bitmap.

3 ECC PRINCIPLES

ECC is based on the mathematical group defined by the points of elliptic curves over a suitable field F. In the current implementation, as is common practice, the choice is a prime field $F = F_p$ for prime p, associated with the following typical Weierstrass form in affine coordinates:

$$y^2 = x^3 + ax + b \tag{1}$$

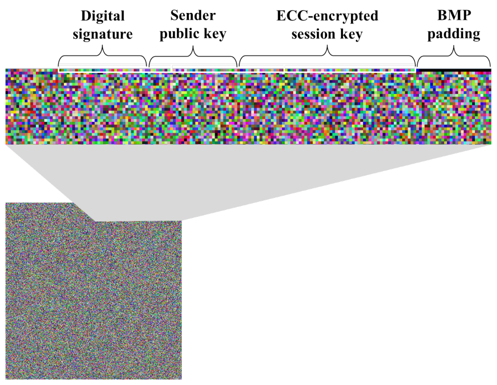

Figure 8: Embedding data into the encrypted file bitmap.

Elliptic curve cryptosystems exploit the Elliptic Discrete Logarithm (EDL) problem, which consists in finding the value of the integer k given the points P, $Q \in E$ and $Q = kP$, where E is an elliptic curve. The EDL problem has no known sub-exponential solution.

This characteristic makes the EDL problem harder than integer factorization, allowing for shorter keys which do not need to be upgraded over time as with RSA.

Optimization techniques were used to speed-up the elliptic curve computations. One of these consists in using Jacobian projective coordinates. We observed that this translates into roughly a three-fold speed-up compared to affine coordinates.

4 MAIN DESIGN FEATURES

4.1 Key Exchange Protocol

4.1.1 Diffie-Hellman Key Exchange for Elliptic Curves

We present a recap of the Elliptic Curve Diffie-Hellman (ECDH) key agreement protocol in order to explain why it was not considered suitable for this application.

ECDH allows two parties, each having an elliptic curve public-private key pair, to establish a shared secret over an insecure channel. The shared secret may be directly used as a key for a subsequent symmetric cipher, e.g. AES (the Advanced Encryption Standard).

ECDH works as follows: initially, the domain parameters (field prime p, curve coefficients and public point G) must be agreed upon. Each party must have a key pair consisting of a private key k (a randomly selected integer in the interval $[2, p-2]$) and a public key Q where

$$Q = kG \tag{2}$$

Let Alice's key pair be (k_A, Q_A) and Bob's key pair be (k_B, Q_B). Each party must know the other party's public key, thus an exchange must occur. Alice computes a curve point given by the product of her private key (a number) and Bob's public key (a point)

$$(X_k, Y_k) = k_A Q_B \tag{3}$$

The elliptic multiplication obfuscates the private key. Bob then computes the same point as

$$(X_k, Y_k) = k_B Q_A \tag{4}$$

The shared secret is X_k (the x coordinate of the point), which can be used as the key of a symmetric block cipher.

A desirable security feature of an encryption tool consists in the ability to generate a different symmetric cipher key (session key) for each message encryption.

However, the scheme above forces two users to interact twice for every exchange: first to share the public keys and then the actual encrypted message.

This would be impractical in a personal encryption system, and could potentially become a vulnerability. In-fact, in absence of strong identity authentication through certification authorities, each new public key sharing could in theory open the door to a man-in-the-middle attack.

4.1.2 Original Direct Embedding Scheme

In order to avoid the drawbacks of ECDH, a solution consists in the unilateral generation of the AES session key by the sending party, and its subsequent direct embedding into the elliptic curve.

The idea stems from the Elgamal scheme illustrated by Crandall and Pomerance (R. Crandall, 2005). There, the goal was to describe how to perform encryption and decryption of plaintext using just elliptic algebra.

Here we simply extend the concept by applying the method to the AES session key rather than to the whole message. Hereafter we summarize the original algorithm.

Alice and Bob have agreed upon a public curve $E(F_p)$, its twist curve E', and the respective public points P, P'. Bob has generated respective public keys

$$P_B = k_B P$$
$$P'_B = k_B P'$$

Alice wishes to send an encrypted parcel $X \in [0, \dots, p-1]$ to Bob. The process works as follows:

1. Alice embeds plaintext X: Alice determines for which of the two curves E or E' X is a valid x-coordinate and determines the associated y coordinate thus defining the message point (X, Y). Alice choses the relevant public point and Bob's public key:

   $$d = 0 \quad or \quad 1 \qquad \text{(bit for curve identification)}$$
   $$Q = P \quad or \quad P'$$
   $$Q_B = P_B \quad or \quad P'_B$$

2. Alice then chooses a random $r \in [2, p-2]$ and performs an elliptic add to obfuscate the message, and computes a clue which will be used for decryption

   $$U = rQ_B + (X, Y) \tag{5}$$

   $$C = rQ \tag{6}$$

3. Alice can now send a message parcel to Bob as the combination of (U, C, d).
4. Bob decrypts the encrypted message to recover the plaintext X: Bob knows which curve to use based on d, then performs an elliptic subtract and recovers the plaintext as the x-coordinate X.

   $$(X, Y) = U - k_B C \tag{7}$$

The above algorithm requires both a curve and its twist to be specified, because it is based on the plaintext embedding theorem (R. Crandall, 2005).

The theorem states that given prime $p > 3$, an elliptic curve $E : y^2 = x^3 + ax + b$ defined over F_p, and given $X \in [0, p-1]$, then X is either a valid x-coordinate of some point on E, or on its twist curve E' where $E' : gy^2 = x^3 + ax + b$ for some g with the Legendre symbol

$$\left(\frac{g}{p}\right) = -1 \tag{8}$$

Therefore the above embedding solution involves the added burden of defining two curves and sharing between the parties the extra piece of information related to the curve-identification bit.

4.1.3 Modified Direct-Embedding Scheme Applied to Key Exchange

In order to avoid the above problem, the tool adopts a probabilistic embedding of the AES key through the algorithm suggested by Koblitz (D. Hankerson, 2004; Koblitz, 1994), in lieu of the deterministic direct-embedding theorem.

Let E be an elliptic curve defined over the field F_p for prime p

$$E: \qquad y^2 = f(x) = x^3 + ax + b \tag{9}$$

We want plaintext message m to be embedded in point $P_m = (x_m, y_m)$ on E. In this case m is a random number generated by Alice and used as a session key to encrypt a message for Bob.

$$\text{set} \quad 30 \leq k \leq 50 \tag{10}$$

The probability that a random integer $\leq p$ is a valid x-coordinate of the elliptic curve E is $\sim 50\%$, thus the range of values for k is set to guarantee that a suitable x_m is found with very high probability (failure probability $\frac{1}{2^k}$).

The message parcel m shall satisfy the following conditions

$$0 \leq m \leq M, \qquad p > Mk \tag{11}$$

We define x_m as

$$x_m = mk + j, \qquad 0 < j \leq k \tag{12}$$

Alice cycles j until a square root of $f(x_m)$ is found, meaning that x_m is a legitimate coordinate and thus m can be embedded into point P

$$m \rightarrow P(x_m, y_m) \tag{13}$$

Plaintext can be recovered by Bob as:

$$m = \lfloor (x_m - 1)/k \rfloor \tag{14}$$

4.2 Encryption and Decryption Process

Based on the above algorithms, Alice can now generate a random AES session key, use it to encrypt a message for Bob using standard AES cipher (NIST, 2001a), embed the key as a point into an elliptic curve and send it to Bob together with the AES-encrypted message.

Bob decrypts the AES key with the probabilistic algorithm and recovers the message.

The following is a simplified description of the system set-up and encryption / decryption procedures between Alice and Bob:

1. Security Level Selection: Alice chooses a number n of bits for the key length, which will be the number of bits of the prime number p of the finite field F_p.
2. Alice computes a random prime p of n-bits of length, this is done through a combination of Fermat's Little Theorem and Miller-Rabin primality test.
3. Parameter choice: Alice uses Atkin-Morain CM (Complex Multiplication) method (R. Crandall, 2005) for generating curves and orders to identify a prime curve order q and the associated coefficients a, b of an elliptic curve defined over the field F_p. Alice then computes a random public point $P(x, y)$ on the curve.
4. Alternatively Alice can choose one of the standard NIST curves (NIST, 2009) and associated security levels (192, 224, 256, 384, 521 bit). These curves have prime moduli of special form allowing very fast modular reduction.
5. Alice shares with Bob the parameters p, q, a, b, P; this can be done by sending a parameter file.
6. Alice and Bob choose respective private keys as random integers $k_A, k_B \in [2, p-2]$ and calculate public keys:
$$Q_A = k_A P$$
$$Q_B = k_B P$$
Note: steps 1 to 6 above are repeated only when a key change is desired.
7. Alice generates a random session key k through an AES-based PRNG (Pseudo Random Number Generator) and encrypts with it a message for Bob; then embeds the key as x-coordinate(s) x_k of a point $P_k = (x_k, y_k)$ on the elliptic curve. This is done by means of the probabilistic algorithm described above.
8. Alice chooses random $r \in [2, p-2]$ (again, this is done through a random number generator). Alice then obfuscates the key with an elliptic add, essentially following a simpler version of the direct embedding algorithm previously described (R. Crandall, 2005)
$$U = rQ_B + (x_k, y_k) \tag{15}$$
9. Alice then prepares the clue for recovering the message
$$C = rP \tag{16}$$
10. Alice sends to Bob U and C together with the AES-encrypted message.
11. Bob then recovers the coordinate(s) x_k in the session key point(s) by means of a point subtraction
$$(x_k, y_k) = U - k_B C \tag{17}$$
12. The recovery of the encoded key k from the coordinate x_k is done as already mentioned in the probabilistic algorithm described before.
13. Bob can now use the session key k to decrypt the message.

4.3 Atkin-Morain Method for Curve Order

As we mentioned, the tool offers the possibility to use custom curves of a given security level, as an extra option besides the NIST-recommended curves.

A cryptographically secure curve suitable for our purposes shall be defined on a prime field F_p and have a prime order q. The algorithm used to search a curve with such features is based on a simplified version of the Atkin-Morain method (R. Crandall, 2005).

A first goal is to seek primes p with special form

$$4p = u^2 + |D|v^2 \tag{18}$$

for which it is straightforward to compute the associated possible orders, then the resulting curve coefficients can be easily calculated.

The idea is that by trying repeatedly to find values of p which satisfy the special form above, one can then check the resulting orders until a prime order is found.

In order to simplify the algorithm (possibly at the expense of some efficiency), we decided to run the check only for $D = -3$ and $D = -4$ whose associated orders are:

$D = -4 \quad \rightarrow \quad$ 4 orders:

$$p + 1 \pm u, \quad p + 1 \pm 2v \tag{19}$$

$D = -3 \quad \rightarrow \quad$ 6 orders:

$$p + 1 \pm u, \quad p + 1 \pm \frac{(u \pm 3v)}{2} \tag{20}$$

4.4 AES Implementation Features

The tool implements the CBC (Cipher Block Chaining) operation mode, as recommended in (NIST, 2001b). CBC combines each plaintext block with the previous ciphertext block by XORing; thus any given plaintext block always gets encrypted to a different ciphertext block, as shown in Figure 9. The cipher is therefore more secure than in an electronic codebook mode because no patterns can be discerned.

The initialization vector (IV) XORed with the first block was considered not necessary in our tool; this is because in the current implementation a new AES key is generated at each encryption.

In fact the purpose of the IV (as recommended by NIST in (NIST, 2001b)) is to ensure that the cipher is different for repeated encryptions of the same plaintext, but this is already ensured in the tool by having different session keys for each encryption. The resulting simplified CBC scheme is illustrated in Figure 10.

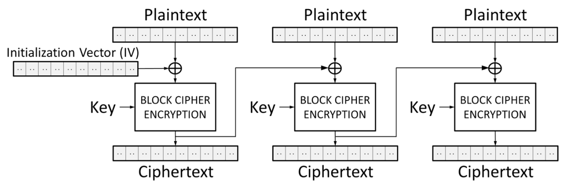

Figure 9: The AES Cipher Block Chaining mode of operation.

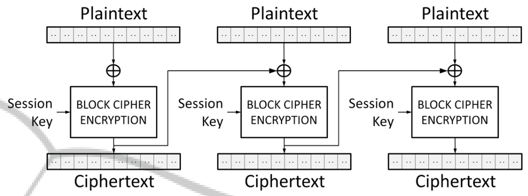

Figure 10: The modified Cipher Block Chaining mode without Initialization Vector but with an ever changing session key.

4.5 Primality Test

A combination of Fermat's Little Theorem (FLT) and Miller Rabin Primality Test (MRPT) has been implemented to optimize speed and probability of prime detection.

Random numbers are generated with an AES-based PRNG and then checked for primality with FLT.

If a candidate number passes the test it is then verified with MRPT which is slower but safer, being able to detect Carmichael numbers which pass FLT test though not being prime.

The pseudo algorithm goes as follows:

```
findprime:;
AES_PRNG(randnum);  // generates random number
While (CheckPrimeFLT(randnum) = nopass) {
   randnum = randnum + 2;
}
if (CheckPrimeMRPT(randnum) = nopass) {
   goto findprime;
}
```

4.6 Pseudo Random Number Generation

All random number generation in the tool is based on cryptographically secure algorithms. PRNGs are used in tasks such as session key generation, prime number search and testing, prime order curves search etc.

The protocol that we used in the tool is an implementation of the ANSI X9.31 algorithm recommended in (NIST, 2007; NIST, 2005), based on the AES used in a Counter Mode.

Let $AES_K(Y)$ represent the AES encryption of Y under the key K.

Let K be a key used only for the generation of pseudo-random numbers (it can be 128, 192 or 256 bit long).

Let V be a 128-bit seed value which is kept secret, and $\oplus$ be the exclusive-or operator.

Let DT be a date/time vector which is updated on each iteration, while I is an intermediate value.

A vector R is generated as follows (DT, I, and R are 128-bits each):

$$I = AES_K(DT) \quad , \quad R = AES_K(I \oplus V) \qquad (21)$$

A new V is generated by

$$V = AES_K(R \oplus I) \qquad (22)$$

5 SCREENSHOTS

Figures 11, 12, 13, 14 illustrate the graphical design of the tool's interface, showing how the various operations described above are actually conducted by the user in a seamless, rather intuitive way.

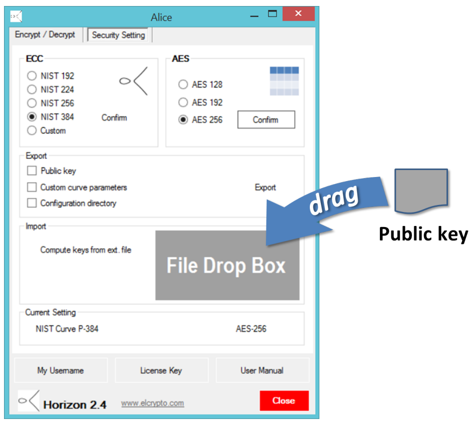

Figure 11: Security setting panel: loading a public key.

Table 1: Tool performance test results.

CPU Intel Core i7 @ 2.40 GHz, 12 MB RAM	
Security: NIST curve 384 bit, AES 256 bit	
File size (MB)	Time to encrypt (decrypt) (s)
10	≈ 1
100	8
700	60

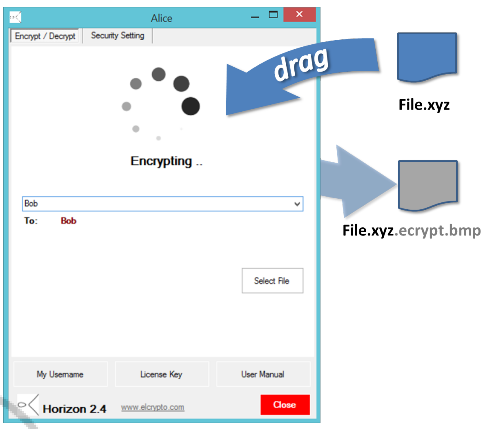

Figure 12: File encryption.

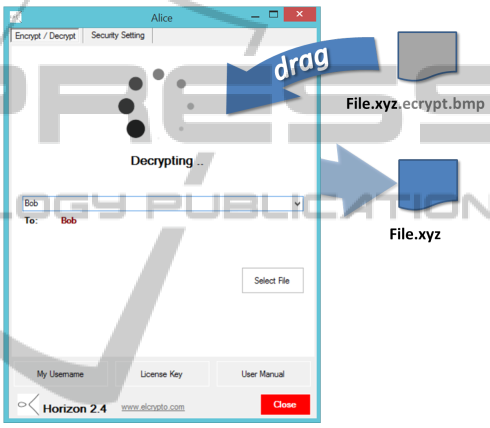

Figure 13: File decryption.

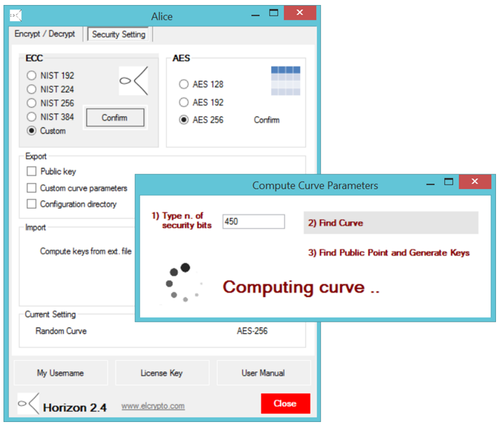

Figure 14: Panel for the calculation of prime order curve.

6 SYSTEM PERFORMANCE

Table 1 presents some performance data, showing that the tool is of practical use for most common file sizes. Multiple file encryption is possible by loading files in a folder, creating a zipped folder and encrypting it.

7 CONCLUSIONS

A new library of modular arithmetic and cryptographic functions was developed and then used for a specific implementation: a user-friendly file encryption tool aimed at quick deployment of secure communication capability even in absence of a PKI.

In its current configuration the system is mainly targeted to relatively small groups of users who have already established communication links able to provide some form of identity validation. This could take the form of a standard email system or possibly even social networks, which could play the role of certification authorities in the near future.

There are some features that distinguish the tool from other personal encryption systems, notably PGP. In our case the intention was to deploy a very light system, i.e. with virtually no learning curve for the user, but without sacrificing security.

Simplicity here means limiting the choice of cryptographic algorithms to only those which are relevant, thus avoiding the inclusion of a wide selection of older ciphers or public key protocols with limited added value. AES ad ECC represent the current state of the art, therefore the idea was to stick to those and simply offer different key lengths associated with proven, standard elliptic curves. Another feature is the ability to choose a random curve, thus creating a unique set of parameters for extra security guarantee.

As opposed to PGP, the tool here presented is a stand-alone system which is not designed to interface with email through the use of plug-ins. These features add sophistication but at the cost of complexity and compatibility issues.

We aimed at privileging the most critical use cases, typically associated with the occasional need to protect specific files rather than the systematic encryption of the whole email traffic. The adopted approach possibly sacrifices some elegance and integration in favor of enhanced simplicity.

In its current implementation the tool is not designed to encrypt emails, rather it can selectively encrypt files for a specific recipient. Then the files can be attached to an email addressed to the recipient who can download and decrypt.

Additionally, the tool can be used to encrypt files for secure storage on shared folders or in the Cloud.

Standard email systems are provided with some level of security, however the content of messages can in principle be altered because encryption takes place between individual SMTP (Simple Mail Transfer Protocol) relays and not between sender and recipient. Furthermore users normally have no control over this feature. Trust in external bodies is therefore implicitly required.

The proposed tool does not rely on centralized infrastructures for the set-up of the security level, key distribution and management of the encryption/decryption and signature functions. Full ownership of the functionality reduces chances of external interference.

In order to establish the system's initial set-up, one of the users shall first generate the curve parameters associated with a security level of choice and then communicate the chosen parameters to the other users. The security level can be changed by following the above procedure if required.

The tool represents a readily deployable solution, useful in absence of sophisticated PKIs.

Possible future developments may involve the addition of Supersingular Isogeny Diffie-Hellman Key Exchange (SIDH) which would provide post-quantum resilience, as suggested in (L. De Feo, 2011). The already available elliptic arithmetic library would make this extension not too complex.

REFERENCES

D. Hankerson, A. J. Menezes, S. V. (2004). *Guide to Elliptic Curve Cryptography*. Springer-Verlag.

Koblitz, N. (1994). *A Course in Number Theory and Cryptography*. Springer-Verlag.

L. De Feo, D. Jao, J. P. (2011). Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies. In *PQCrypto Proceedings*. Springer.

NIST (2001a). *FIPS PUB 197, Announcing the Advanced Encryption Standard (AES)*. Federal Information Processing Standard.

NIST (2001b). *Special Publication 800-38A, Recommendation for Block Cipher Modes of Operation, Methods and Techniques*. NIST Publication.

NIST (2002). *FIPS 180-2, Announcing the Secure Hash Standard*. Federal Information Processing Standard.

NIST (2005). *NIST-Recommended Random Number Generator Based on ANSI X9.31, Appendix A.2.4: Using the 3-Key Triple DES and AES Algorithms*. NIST Publication.

NIST (2007). *Special Publication 800-90, Recommendation for Random Number Generation Using Deterministic Random Bit Generators (Revised)*. NIST Publication.

NIST (2009). *FIPS PUB 186-3, Digital Signature Standard (DSS)*. Federal Information Processing Standard.

R. Crandall, C. P. (2005). *Prime Numbers, A Computational Perspective*. Springer.

V. Gruhn, M. Hulder, V. W.-M. (2007). Utilizing social networking platforms to support public key infrastructures. In *SECRYPT 2007 Proceedings*. SCITEPRESS.