ISSN:0975-9646

A Closer look at RSA and ECC

Aditya Kumar Kommera[1], Keerthana Kommera[2], Praneeth Kumar Gunda[3]

[1]*Dept. of Information Technology,*
Krishna Murthy Institute of Technology and Engineering,
Ghatkesar, Andhra Pradesh

[2]*Dept. of Information Technology,*
Guru Nanak Engineering College
Ibrahimpatnam, Andhra Pradesh, India

[3]*Software Engineer,*
eIQ networks R&D India Pvt. Ltd.,
Hyderabad, Andhra Pradesh, India

Abstract— **Information Security plays a key role in the field of modern computing. The public Key encipherment is the security mechanism which uses the idea of separating the key used to encrypt a message from the one used to decrypt it .The most widely used and industrially accepted algorithms of Public Key Cryptosystem are RSA and ECC(Elliptic Curve Cryptosystem). Even though decades of research has been done to answer the question "Which is efficient among RSA and ECC?" there is no conclusion from the researchers yet now. Through this project, we try to answer this question by providing some experimental results and an in-depth comparison between RSA and ECC on different tradeoffs involved in choosing them such as Speed of Encryption/ Signature etc., and Platform Considerations even. This will help the industry to choose one among them.**

Keywords— **Elliptic Curve Cryptography, RSA Algorithm, Performance Evaluation**

I. INTRODUCTION

In traditional cryptography, the sender and receiver of a message know and use the same secret key; the sender uses the secret key to encrypt the message, and the receiver uses the same secret key to decrypt the message. This method is known as secret key or symmetric cryptography. The main challenge is getting the sender and receiver to agree on the secret key without anyone else finding out. If they are in separate physical locations, they must trust a courier, a phone system, or some other transmission medium to prevent the disclosure of the secret key. Anyone who overhears or intercepts the key in transit can later read, modify, and forge all messages encrypted or authenticated using that key. The generation, transmission and storage of keys is called key management (see Section 4.1); all cryptosystems must deal with key management issues. Because all keys in a secret-key cryptosystem must remain secret, secret-key cryptography often has difficulty providing secure key management, especially in open systems with a large number of users.

In order to solve the key management problem, Whitfield Diffie and Martin Hellman [1] introduced the concept of public-key cryptography in 1976. Public-key cryptosystems have two primary uses, encryption and digital signatures. In their system, each person gets a pair of keys, one called the public key and the other called the private key. The public key is published, while the private key is kept secret. The need for the sender and receiver to share secret information is eliminated; all communications involve only public keys, and no private key is ever transmitted or shared. In this system, it is no longer necessary to trust the security of some means of communications. The only requirement is that public keys be associated with their users in a trusted (authenticated) manner (for instance, in a trusted directory). Anyone can send a confidential message by just using public information, but the message can only be decrypted with a private key, which is in the sole possession of the intended recipient. Furthermore, public-key cryptography can be used not only for privacy (encryption), but also for authentication (digital signatures) and other various techniques.

Encryption When Alice wishes to send a secret message to Bob, she looks up Bob's public key in a directory, uses it to encrypt the message and sends it off. Bob then uses his private key to decrypt the message and read it. No one listening in can decrypt the message. Anyone can send an encrypted message to Bob, but only Bob can read it (because only Bob knows Bob's private key).

Digital Signatures To sign a message, Alice does a computation involving both her private key and the message itself. The output is called a digital signature and is attached to the message. To verify the signature, Bob does a computation involving the message, the purported signature, and Alice's public key. If the result is correct according to a simple, prescribed mathematical relation, the signature is verified to be genuine; otherwise, the signature is fraudulent, or the message may have been altered.

Since the introduction of public key cryptosystem by Diffie Hellman [1] in 1976, numerous public key cryptosystems have been proposed and implemented. The first practical implementation followed in 1977 when Rivest, Shamir and Adleman proposed their now well-known RSA cryptosystem [2], in which security is based on the intractability of the integer factorization problem. The first use of elliptic curve in cryptography parlance was Lenstra's elliptic curve factorization algorithm. Inspired by this sudden unexpected application of elliptic curves in integer factorization, in the mid 1980s, Neal Koblitz and Victor Miller independently introduced the elliptic curve public key cryptography system, a method based on the discrete logarithmic problem over the points on an elliptic curve. Elliptic curve cryptographic schemes are public key mechanisms that provide the same functionality as RSA schemes. However, the security of ECC is based on the hardness of a different problem, namely the elliptic curve discrete logarithm problem (ECDLP). Currently the best algorithms known to solve the ECDLP have fully exponential running time, in contrast to the sub exponential-time algorithms known for the integer factorization problem. This means that a desired security level can be attained with significantly smaller keys in elliptic curve systems than is possible with their RSA counterparts. The advantages that can be gained from smaller key sizes include speed and efficient use of computing power, bandwidth and storage

II. RSA CRYPTOSYSTEM

RSA was proposed in 1977 shortly after the discovery of public-key cryptography. It has survived all attempts to break it for many years and is considered very strong. We will summarize how to use the method.

1. Pick two large primes, p and q.
2. Multiply them together to get n = p x q.
3. Compute z =(p-1) x (q-1). This will be the mod of the exponents.
4. Pick any number d, which is smaller than, and relatively prime to, z.
5. Solve the diophantine equation d x e=z x k+1. The value of k is not important, but the value of e is. We have to make sure a positive value of e, which is less than z

That is all. The public key is order pair (n, e) and private key is also order pair (n, d).

A. *Encryption/ Decryption*

Divide the message into blocks, so that each plaintext block, P, lies in the interval $0 \leq P < n$. We can group the message into plaintext blocks of k bits, where k is the largest integer for which $2^k < n$ is true. Anyone wishing to send a message to Bob make use of e and n. If Alice wants to send a message to Bob, he calculates the ciphertexts as

$$C = P^e (\text{mod } n)$$

Alice sends C, the cipehertext, to Bob.

When Bob receives the ciphertext, he uses his private key d to decrypt the message as

$$P = C^d (\text{mod } n)$$

B. *Signature Generation/ Verification*

Signer sends the Message m with his private key (n, d). i.e.,

$$S \equiv m^d (\text{mod } n).$$

Then the verification at the receiver side is carried out using the public key (n,e) of the signer if

$$S^e \equiv m (\text{mod } n).$$

III. ELLIPTIC CURVE CRYPTOSYSTEM

In this section, we include a brief overview of ECC.
ECC can be used to encrypt plaintext messages, M into cipher texts, C, and decrypt cipher texts into plaintext messages. The plaintext message M is to be encoded into a point Pm from the finite set of points in the elliptic group, Ep(a,b). One of the design issues in the use of the elliptic curve for cryptography is the mapping of arbitrary plaintext into points on the elliptic curve. One method used in this paper is given below. We assume a curve of the form

$$y^2 (\text{mod } p) = x^3 + ax + b (\text{mod } p)$$

A *Key Generation*

The entire document should be in Times New Roman or Times font. Type 3 fonts must not be used. Other font types may be used if needed for special purposes.

1.Alice and Bob agree on a generator point G=(xg,yg) and an elliptic group Ep(a,b).

2.Alice chooses an integer na and calculates Pa=naG=(xa,ya)

3. Alice’s public key is Pa=(xa,ya) and his private key is na.

4. Bob also chooses an integer nb and calculates Pb=nbG=(xb,yb)

5. Bob’s public key is Pb=(xb,yb) and his private key is nb.

B. *Encryption/ Decryption*

Alice wishes to send a message Pm= (xm,ym) to Bob. He carries out the following steps

1. Alice chooses a random number k.
2. He calculates c1=kG and c2=Pm+kPb.
3. Alice sends the Cm={c1,c2 } as cipher text to Bob.

Upon receiving the ciphertext pair (c1,c2) from Alice, Bob recovers the message as follows:

He multiplies c1 by his private key nb and subtract it from c2. That is, he calculates c2–nbc1= (Pm+kPb) -nb(kG) =(Pm+knbG) -nbkG =Pm =(xm,ym)

C. *Signature Generation/ verification*

Suppose Alice wants to send a signed message to Bob. Initially, the curve parameters (q,FR,a,b,[DomainParameterSeed,]G,n,h) must be agreed upon. q is the field size; FR is an indication of the basis used; a and b are two field elements that define the equation of the curve; DomainParameterSeed is an optional bit string that is present if the elliptic curve was randomly generated in a verifiable fashion;G is a base point of prime order on the curve (i.e., $G = (x_G, y_G)$); n is the order of the point G; and h is the cofactor (which is equal to the order of the curve divided by n).

Also, Alice must have a key pair suitable for elliptic curve cryptography, consisting of a private key d_A (a randomly selected integer in the interval $[1, n-1]$) and a public key QA (where $QA = d_A G$). Let Ln be the bit length of the group order n.For Alice to sign a message m, she follows these steps:

1. Calculate e = HASH(m), where HASH is a cryptographic hash function, such as SHA-1, and let z be the Ln leftmost bits of e.
2. Select a random integer k from $[1, n-1]$.
3. Calculate r = x1(mod n), where (x1,y1) = kG. If r = 0, go back to step 2.
4. Calculate $s = k^{-1}(z + rd_A)(\text{mod } n)$. If s = 0, go back to step 2.
5. The signature is the pair (r,s).

For Bob to authenticate Alice's signature, he must have a copy of her public key QA. If he does not trust the source of QA, he needs to validate the key (O here indicates the identity element):

1. Check that QA is not equal to O and its coordinates are otherwise valid
2. Check that QA lies on the curve
3. Check that nQA = O

After that, Bob follows these steps:

1. Verify that r and s are integers in [1,n − 1]. If not, the signature is invalid.
2. Calculate e = HASH(m), where HASH is the same function used in the signature generation. Let z be the Ln leftmost bits of e.
3. Calculate w = s − 1(mod n).
4. Calculate u1 = zw(mod n) and u2 = rw(mod n).
5. Calculate (x1,y1) = u1G + u2QA.
6. The signature is valid if r = x1(mod n), invalid otherwise.

IV. Performance Evaluation of RSA and ECC

A. In terms of Security

The saying "A chain is no stronger than its weakest link" is a very suitable for describing attacks on cryptosystems. The attackers' instinct is to go for the weakest point of defense, and to exploit it. Sometimes the weakness may have appeared insignificant to the designer of the system, or maybe the cryptanalyst will discover something that was not seen by anyone before. The important thing to remember (and this has been proven time and time again in the history of cryptography) is that no matter how secure you think your system is, there may be something you have not considered.

Attacks on RSA: Even though the security of RSA is said to be wholly dependent on Prime Factorization there are other possibilities of attacks which are summarized as:

Searching the message space: One of the seeming weaknesses of public key cryptography is that one has to give away everybody the algorithm that encrypts the data except the key. If the message space is small, then one could simply try to encrypt every possible message block, until a match is found with one of the ciphertext blocks. In practice this would be an insurmountable task because the block sizes are quite large.

Guessing d: Another possible attack is a known ciphertext attack. This time the attacker knows both the plaintext and ciphertext (they simply has to encrypt something). They then try to crack the key to discover the private exponent, d. This might involve trying every possible key in the system on the ciphertext until it returns to the original plaintext. Once d has been discovered it is easy to find the factors of n. Then the system has been broken completely and all further ciphertexts can be decrypted.

The problem with this attack is that it is slow. There are an enormous number of possible ds to try. This method is a factorizing algorithm as it allows us to factor n. Since factorizing is an intractable problem we know this is very difficult. This method is not the fastest way to factorize n. Therefore one is suggested to focus effort into using a more efficient algorithm specifically designed to factor n.

Cycle Attack: This attack is very similar to the last. The idea is that we encrypt the ciphertext repeatedly, counting the iterations, until the original text appears. This number of re-cycles will decrypt any ciphertext. Again this method is very slow and for a large key it is not a practical attack. A generalisation of the attack allows the modulus to be factored and it works faster the majority of the time. But even this will still have difficulty when a large key is used. Also the use of p-- strong primes aids the security.

Common Modulus: One of the early weaknesses found was in a system of RSA where the users within an organization would share the public modulus. That is to say, the administration would choose the public modulus securely and generate pairs of encryption and decryption exponents (public and private keys) and distribute them all the employees/users. The reason for doing this is to make it convenient to manage and to write software for.

Faulty Encryption: Joye and Quisquater showed how to capitalise on the common modulus weakness due to a transient error when transmitting the public key. Consider the situation where an attacker, Malory, has access to the communication channel used by Alice and Bob. In other words, Malory can listen to anything that is transmitted, and can also change what is transmitted. Alice wishes to talk privately to Bob, but does not know his public key. She requests by sending an email, to which Bob replies. But during transmission, Malory is able to see the public key and decides to flip a single bit in the public exponent of Bob, changing (e,n) to (e',n).

When Alice receives the faulty key, she encrypts the prepared message and sends it to Bob (Malory also gets it). But of course, Bob cannot decrypt it because the wrong key was used. So he lets Alice know and they agree to try again, starting with Bob re-sending his public key. This time Malory does not interfere. Alice sends the message again, this time encrypted with the correct public key.

Malory now has two ciphertexts, one encrypted with the faulty exponent and one with the correct one. She also knows both these exponents and the public modulus. Therefore she can now apply the common modulus attack to retrieve Alice's message, assuming that Alice was foolish enough to encrypt exactly the same message the second time.

And other common attacks are Side channel attacks like Timing attacks etc.,

Attacks on ECC: Mainly the security of ECC depends on ECDLP. It is already proven that ECC is vulnerable to Shor’s algorithm for solving DLP. Many other algorithms have been developed to solve the DLP, some of them are Pollard-Rho’s Attack, Pohling-Hellman’s Algorithm etc.,

Other kinds of attacks that have been tried on ECC are Side channel attacks, like Power Analysis Attacks, Cache Based Attacks etc.,

B. System Requirements for Setting up of Cryptosystem

In setting up any cryptosystem a certain amount of computation is required. In this section we will compare some of the basic set-up requirements for elliptic curve cryptosystems with those for users of RSA

There are several approaches for selecting an appropriate elliptic curve. They all tend to be mathematically very complicated and they have some limitations. It is perhaps worth pointing out at this stage that implementing elliptic curve cryptosystems can in fact be quite challenging without a good understanding of the mathematics of elliptic curves.

So we see that setting up the system parameters for an elliptic curve cryptosystem is quite involved. However, once it is done, the resulting elliptic curve parameters may be used for multiple users within a group (just as in the case of discrete logarithm cryptosystems) and each user has his or her public/private key pair. These key pairs are easy to generate and consist of a random, secret integer k that acts as the private key and that multiple of the generator point G on the curve that acts as the public key kG for the user. The security assumption is that it is hard to compute the private key k and the public key kG.

By way of comparison, the RSA cryptosystem requires no system parameters. The first stage of computing a public/private key pair consists of the user generating two primes of the appropriate size and computing the public modulus n as their product. This part of the computation can be rather computationally intensive (though not as intensive as setting up elliptic curve system parameters). The second stage for the user is then to compute the secret exponent d, or certain information that allows decryption to be optimized (so-called Chinese Remainder Theorem information), from what is usually a fixed public exponent e. The calculation of the secret exponent (or related information) is insignificant when compared to the time required to generate the primes. The various requirements for the different cryptosystems are given in Table 1 below.

TABLE I
SYSTEM REQUIREMENTS FOR RSA AND ECC CRYPTOSYSTEMS

	RSA	**ECES and ECDSA**
System Parameters	None	the field F, two field elements that represent the curve, the generator G on the curve and the order of G
Public Key	Modulus n and exponent e	point P= kG on the elliptic curve
Private Key	Exponent d or corresponding CRT information	an integer k where 0<k<q

C. Storage Requirements

Interest in elliptic curve cryptosystems is fueled by the appeal of basing a cryptosystem on a different hard problem and the fact that currently such a choice appears to lead to smaller system parameters and key sizes for the same level of security.

Throughout this section we will be comparing the requirements and performance of 1024-bit RSA (with public exponent $2^{16}+1$) with an elliptic curve cryptosystem implemented over the field GF(q) where q is 160 bits in length and the field is either of characteristic 2 or of odd characteristic. For the purposes of this note, we will assume that these different fields have essentially the same implementation requirements. In Table 3 we give a rough comparison of the storage requirements in bits for the schemes of interest to us in this note.

TABLE III
STORAGE REQUIREMENTS

	RSA 1024 bit n and e=2^{16}+1	**ECES and ECDSA on GF(q)**
System Parameters	0	(4*160)+1= 641
Public Key	1024+17=1041	160+1=161
Private Key	160 (801 with system parameters)	2048(or 2560 with CRT information)

D. Performance

With regard to the speed of implementation of these cryptosystems the situation is still very unclear. The basic elliptic curve operations are in fact quite complicated 3 (more complicated in fact than the operations required for RSA) and so if elliptic curve cryptosystems ever require the same size of parameters as does an implementation of RSA then the elliptic curve cryptosystem can be expected to be slower. In fact it is possible to envisage situations where even if the elliptic curve implementation uses smaller parameters than some implementation of RSA, the latter might remain the more efficient in terms of practical use. At present however, the current parameter advantages for elliptic curve cryptosystems are such that the speed of implementation can compare favourably with the performance of RSA.

Putting quantitative data into this part of the note is very difficult. We know of no figures or benchmarks with which we can compare an optimized version of RSA on one platform with an optimized version of some elliptic curve cryptosystem. However we make an attempt to qualitatively compare the performance of the various systems to the speed of RSA for the relevant operation and these results are presented in Table 2. These figures should be taken as a guide only and in making these comparisons we have assumed that one elliptic curve addition takes roughly the same effort as 10 modular multiplications. We feel that, for the purposes of this note, this figure will give a rough but fair comparison between cryptosystems. All techniques for pre computation that apply to discrete logarithm cryptosystems will apply equally to systems based on elliptic curves. It is interesting to note that even with the smaller keys required for elliptic curve cryptosystems, signature verification with RSA remains advantageous.

TABLE III
COMPARATIVE PERFORMANCE OF ECC AND RSA

	RSA 1024 bit n and e=2^{16}+1	**ECES and ECDSA on GF(q)**
Encryption	17	120
Decryption	384	60
Signing	384	60
Verification	17	120

Table 3 gives you the comparative performance of elliptic curve cryptosystems over GF(q) where q is 160 bits in length when compared with 1024-bit RSA and discrete logarithm cryptosystems for various cryptographic functions. Figures in the table are the number of time units required to complete the given operation if we assume that one 1024-bit modular multiplication requires one unit of time. Not included in this table is that Diffie-Hellman key agreement requires 480 time units for each party. These figures do not take account of any of the various optimizations possible and they should be viewed as a rough comparison only.

ACKNOWLEDGEMENTS

The Authors wish to thank Mr. N.M.S Desai for his comments, which helped in improving presentation of this paper. A Vardy wishes to thank Ch. Srinivasulu for his invaluable support.

REFERENCES

[1] Whitfield Diffie and Martin E. Hellman "New Directions in Cryptography-*Invited Paper"*

[2] R.L. Rivest, A. Shamir, and L. Adleman, "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems", 1977.

[3] Ranbir Soram and Memeta Khomdram ,"Juxtaposition of RSA and Elliptic Curve Cryptosystem",2009.

[4] Kumanduri and Romero, Number Theory with Computer Applications, Prentice- Hall of India, New Delhi, 2001.

[5] Blake,Seroussi, and Smart, Elliptic Curves in cryptography,Cambridge University Press,1999.