Juxtaposition of RSA and Elliptic Curve Cryptosystem

Ranbir Soram [†] *and* ***Memeta Khomdram*** [††]

[†] ***Manipur Institute of Technology, Takyelpat, Imphal -795004, India***

[††] ***Department of Electronics Accreditation of Computer Courses Centre, Akampat, Imphal-795008, India***

Summary

This paper presents an in-depth juxtaposition of RSA and Elliptic Curve Cryptosystem (ECC) and provides an overview of the different trade-off involved in choosing between cryptosystems based on them. We offer ECC as a suitable alternative to RSA. We also present experimental results quantifying the benefits of using ECC for public cryptosystems.

Key words:

RSA, Integer Factorization Problem, ECC, ECDLP, Timing Attack.

1. Introduction

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 **R**ivest, **S**hamir and **A**dleman 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 indepently 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 subexponential-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.

Manuscript received September 5, 2009
Manuscript revised September 20, 2009

2. Security goals

Cryptography aims at analyzing what problems can be made very easy while making others extremely hard. Careful examination of the scenarios in network security reveals the following fundamental objectives of secure communications [3,4]:

Confidentiality: keep information private and secret so that only the authorized recipient see it. Messages sent by Alice to Bob should not be readable by Eve.

Data integrity: ensure that information is not tempered with during its transit or its storage on the network. Bob should be able to detect when data sent by Alice has been modified by Eve.

Data origin authentication: provide proof of identity of the sender to the recipient, so that the recipient can be assured that the person sending the information is who or what he or she claims to be. Bob should be able to verify that data purportedly sent by Alice indeed originated with Alice.

Entity authentication: corroborate the identity of an entity. Bob should be convinced of the identity of the other communicating entity.

Non-repudiation: preventing an entity from denying previous commitments or actions. Once the non-repudiation process is in place, the sender cannot deny being the originator of the information. When Bob receives a message purportedly from Alice, not only is Bob convinced that the message originated with Alice, but also Bob can convince a neutral third party.

3. Public Key Cryptosystem

Public key cryptosystem is based on the idea of separating the key used to encrypt a message from the one used to decrypt it. Anyone who wants to send a message to Bob can encrypt that message using Bob's public key but only Bob can decrypt the message using his private key. In implementing a public-key cryptosystem, it is understood

that Bob's private key should be kept secret at all times. Furthermore, even though Bob's public key is publicly available to everyone, including his adversaries, it is impossible for anyone, except Bob, to derive the private key in any reasonable amount of time.

4. 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).

One of the beauties of the RSA algorithm is that the public and private keys can be interchanged without affecting the security of the system. It means that the public key can be the order pair (n, d) and private key can be (n, e).

4.1 Encryption

Divide the message into blocks, so that each plaintext block, P, lies in the interval $0 \le P < n$. 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

$$C=P^e \pmod n$$

Alice sends C, the cipehertext, to Bob.

4.2 Decryption

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

$$P=C^d \pmod n$$

4.3 RSA Design Example

Assume Alice wants to send the message "**http://www.ijcsns.org**" to Bob. Alice chooses two integers **p** and **q**.

The integer **p** is a 77-digit number.

p=73680054488656110405523153970293730091326981370921628773735365768551633318757.

The integer **q** is also a 78-digit number.

q=115318060910582303052969666756729180055703925467022497453321651161405436738911.

Alice calculates **n** and it has 154 digits.

n=84966410114178683644595197219008702230891206642480386293950938951556413926969792091998017694111389267466709828166468029197507035665682753935649024805362 7.

Alice also calculates **z** and it has 154 digits.

z=84966410114178683644595197219008702230891206642480386293950938951556413926967909228045809385276553998539400753715176493851371262304297705224626533177995960.

Alice chooses **e**=65537 and calculates **d**. It has 154 digits.

d=57866407510814522571873323964680019155188643320269001660805441340979431149065194630279027970496570047008676217118928050209226924053435826708269509111901 93.

Alice's message is "**http://www.ijcsns.org".** It is changed to numeric values (*using java*) as shown below.

P=15266100987464762024566676191962879910276857 1708007.

The ciphertext calculated by Alice is $\mathbf{C}=P^e \pmod n$ is

C=80493809573092512227038054900711923766211501458095719917110815595662086399649801260359590792933522533206494411535502431138111424883051430829276079284455.

Bob recover the plaintext from the ciphertext using $\mathbf{P}=C^d \pmod n$, which is

P=15266100987464762024566676191962879910276857 1708007. After converting the numeric numbers to text, the message becomes "**http://www.ijcsns.org".**

5. Security of RSA

We can identify three techniques for attacking the security of RSA:

1. Factor **n** into its prime factors. This enables us to calculate z=(p-1) x (q-1), which, in turn, enable us to determine d and e.
2. Determining **z** without first determining **p** and **q**.

This enables us to determine d and e.

3. Determine **d** directly, without first determining **z**.

In most literatures, the activities of the cryptanalysis of RSA are focused on the task of factoring **n** into two prime factors **p** and **q**. Determining **z** is the same as determining the two factors of **n**. With presently known algorithms, determining **d** given **e** and **n** is time consuming as the factoring algorithm [3]. Hence, the integer factorization performance can be used as the benchmark against which to evaluate the security of RSA.

In number theory [5], integer factorization is the breaking down of a composite number into smaller non-trivial divisors, which when multiplied together equal the original integer. When the numbers are very large, no efficient integer factorization algorithm is publicly known.

In cryptography, we use a term from *computational complexity theory* describing the difficulty of solving a computational problem. A fast integer factorization algorithm would mean that the RSA public-key algorithm is not secure. We need an encryption and decryption algorithm to have a low level of complexity (efficient); we need an algorithm to be used by hackers to have a high level of complexity (inefficient).

The complexity of an algorithm is based on two factors: space and time [4]. The space factor of complexity refers to the memory required to run the algorithm. The time factor of complexity is the time required to run the algorithm. The time complexity of an algorithm depends on the particular computer on which the algorithm is executed. To make complexity (normally the complexity of an algorithm means time complexity) independent of the machine, the bit-level operation complexity is defined. A bit-level operation is the time needed for a computer to add, subtract, multiply, divide two bits or shift / rotate one single bit.

Based on the bit-level operation complexity, the following hierarchy of complexity can be arrived at.

Table 1: complexity hierarchy

Hierarchy	Big- O Notation
Logarithmic	O(log b)
Linear	O(b)
Polynomial	$O(b^c)$
Sub exponential	$O(2^{p(\log b)}$ and p=*f*(logb)
Exponential	$O(2^b)$
Super exponential	$O(b^b)$ or $O(2^x)$ and $x=2^b$

An algorithm with constant or logarithmic or polynomial complexity is considered efficient for any size of ***b***. An algorithm with *sub exponential* complexity is feasible if ***b*** is not very large. But an algorithm with *exponential* and *super exponential* is considered infeasible for large value of ***b*** [4].

If a large, ***b***-bit number is the product of two primes that are roughly the same size, then no algorithm has been published that can factor in polynomial time, *i.e.*, that can factor it in time $O(b^k)$ for some constant *k*. There are published algorithms that are faster than $O((1+\varepsilon)^b)$ for all positive ε, *i.e.*, sub-exponential.

The best published asymptotic running time for the general number field sieve (GNFS) algorithm, which, for a *b*-bit number n, is:

$$O\left(\exp\left(\left(\tfrac{64}{9}b\right)^{\frac{1}{3}}(\log b)^{\frac{2}{3}}\right)\right).$$

For an ordinary computer, GNFS is the best published algorithm for large *n* (more than about 100 digits). For a quantum computer, however, Peter Shor discovered an algorithm in 1994 that solves it in polynomial time. This will have significant implications for cryptography if a large quantum computer is ever built.

It is not known exactly which *complexity classes* contain the decision version of the integer factorization problem. It is known to be in both **NP** (*nondeterministic polynomial*) and **co-NP**(*complementary **NP***). This is because both *YES* and *NO* answers can be trivially verified given the prime factors (we can verify their primality using the ***cyclotomic AKS*** *primality test* [10], and that their product is N by multiplication). In fact, provided we require the factors to be listed in order, the fundamental theorem of arithmetic will guarantee that there is only one possible string that will be accepted; this shows that the problem is in both **UP**(undecidable problem) and **co-UP**(complementary **UP**). It is also suspected to be outside of all three of the complexity classes **P**(polynomial), NP-complete, and co-NP-complete.Many people have tried to find classical polynomial-time algorithms for it and failed.

6. Elliptic Curve Cryptography

Elliptic curve has a rich and beautiful history and mathematicians have studied them for many years. They have been used to solve a diverse range of problems. 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, Neal Koblitz and Victor Miller proposed, in the mid 1980s, the elliptic curve public-key cryptographic systems. Since then an abundance of research has been published on the security and efficient implementation of elliptic curve cryptography. In the late 1990s, elliptic curve systems started receiving commercial acceptance when accredited standard organizations specified elliptic curve protocols, and private companies included these protocols in their security products.

An elliptic curve is a cubic equation of the form:

$$y^2 = x^3+ax+b \quad (1)$$

or

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

or

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

where ***a*** and ***b*** are constants and ***x*** and ***y*** are variables. In fact, *x* and *y* can be complex, real, integers, polynomial basis, optimal normal basis, or any other values from any field. That's part of what makes the math so complex and interesting.

7. Elliptic curve over prime Galois Fields

An elliptic group over the prime Galois Field $E_p(a,b)$ uses a special elliptic curve of the form

$$y^2 \pmod p = x^3+ax+b \bmod p$$

where $a,b \in F_p$, $0 \le x < p$ and $-16(4a^3+27b^2) \bmod p \neq 0$. The constants ***a*** and ***b*** are non-negative integers smaller than the prime p. The condition that $-16(4a^3+27b^2) \bmod p \neq 0$ implies that the curve has no "singular points", which will be essential for the applications we have in mind.

8. Operations required by ECC

The scalar multiplication or repeated addition of elliptic curve points is the main operation required by ECC schemes, although other operations such as division may also be needed. The addition of two elliptic curve points is illustrated geometrically in figure 1 given below. The line connecting the two points P and Q intercepts the curve at a point called –R. We reflect –R on the x-axis to get R. This point R is the sum of P and Q i.e. R=P+Q. To double a point P, first we draw the tangent line and find the other point of intersection -R. We reflect –R on the x-axis to get R. Now, R=P+P=2P. See figure 2 given below.

As it is the scalar multiplication operation that dominates the actual execution timing of ECC schemes, its efficient implementation is crucial.

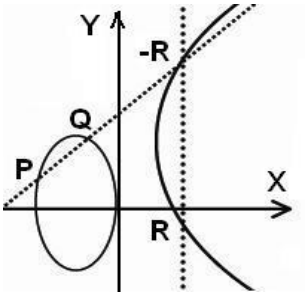

Figure 1. Adding two points

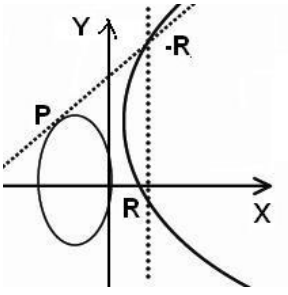

Figure 2. Doubling a point

The actual mathematics depends on the chosen curve and underlying field, however there is a clear hierarchy of underlying mathematical operations, as shown in figure 3 given below.

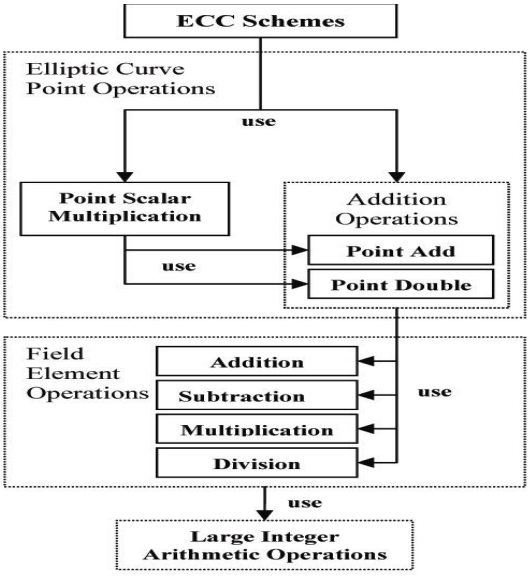

Figure 3: Hierarchy of required underlying mathematical operations

9. The Group Law of Elliptic Curve

Let E be an elliptic curve defined over the field **K**. There is a *chord-and-tangent rule* for adding two points in E (K) to give a third point in E (K). Together with this addition operation, the set of points E(K) forms an abelian group with **0** serving as its identity. It is this group that is used in the construction of elliptic curve cryptographic systems. Algebraic formulas for the group law can be derived from the geometric description. These formulae are established and presented here for non-supersingular elliptic curves E of the form $y^2=x^3+ax+b$.

Group law for E/K : $y^2 = x^3 + ax + b$.

1. *Identity.* P +0=0+ P = P for all P ∈ E (K).

2. *Negative*. If P = (x , y) ∈ E(K),then(x , y) + (x ,-y) =*0.*

The point (x ,-y) is denoted by -P and is called the negative of P; note that -P is indeed a point in E(K). Also, -0 = 0.

3. Point addition. Let $P = (x_1 , y_1) \in E(K)$ and $Q = (x_2, y_2) \in E(K)$,where $P \neq \pm Q$ .Then $P+Q = R = (x_3, y_3)$, where $x_3=\lambda^2-x_1-x_2$, $y_3=\lambda(x_1-x_3)-y_1$ and $\lambda=(y_2-y_1)/(x_2-x_1)$

4. *Point doubling*. Let $P = (x_1 , y_1) \in E(K)$, where $P \neq -P$. Then $2P = R = (x_3 , y_3)$, where $x_3=\lambda^2-2x_1$, $y_3=\lambda(x_1-x_3)-y_1$ and $\lambda=(3x_1^2+a)/(2y_1)$

To establish the algebraic formulae given above,

we pick an elliptic curve equation $y^2 = x^3 + ax + b$. Consider the line $y=\lambda x+c$ and examine the points that lie on the intersection of this line with our curve. These are the points that obey both equations. We can therefore use the linear equation to substitute for ***y*** in our equation, to obtain

$$(\lambda x)^2 + 2c\lambda x + c^2 = x^3 + ax + b \quad (4)$$

This is a cubic equation in x, and we assume it will have three real roots. Two of these are $P(\mathbf{x_1},y_1)$ and $Q(\mathbf{x_2},y_2)$, so we will obtain another, $-R(\mathbf{x_3},-y_3)$. We rewrite our equation (4) as

$$x^3-\lambda^2x^2 + (a-2c\lambda)x + b-c^2 = 0 \quad (5)$$

If $\mathbf{x_1}$, $\mathbf{x_2}$ and $\mathbf{x_3}$ are the three solutions, then by intuition, we compare the given equation (5) with the equation (6) given below:-

$$(x-x_1)(x-x_2)(x-x_3)=0 \quad (6)$$

Hence, $x^3-\lambda^2x^2 + (a-2c\lambda)x + b-c^2 =(x-x_1)(x-x_2)(x-x_3)$

Simplifying the right hand side and comparing the coefficients of $\mathbf{x^2}$ terms on both sides, we get

$$\lambda^2 = x_1+x_2+x_3 \quad (7)$$

We know already x_1 and x_2 in equation (7). So, $x_3=\lambda^2-x_1-x_2$. To get y_3, we proceed as follows. The slope of a straight line passing two points $P(x_1,y_1)$ and $-R(x_3,-y_3)$ is the same as the slope of a straight line passing two points $P(x_1,y_1)$ and $Q(x_2,y_2)$. So, $(-y_3-y_1)/(x_3-x_1)=(y_2-y_1)/(x_2-x_1)= \lambda$. Solving for y_3 we get, $y_3=-y_1+\lambda (x_1-x_3)$.

As $-R(x_3,-y_3)=R(x_3,y_3)$, so $y_3=-y_1+\lambda (x_1-x_3)$.

In the case of point doubling formula, slope is calculated differently, others are same as point addition. In this case to find slope, λ, we find derivative of the curve $y^2 = x^3 + ax + b$ w.r.to x at the point $P(x_1,y_1)$.

So, slope=dy/dx at point $P(x_1,y_1)$.

$$\lambda = (3x_1^2+a)/(2y_1).$$

10. Multiplication over an elliptic curve group

The multiplication over an elliptic curve group Ep(a,b) is the equivalent operation of the modular exponentiation in RSA. The multiplication of points by a scalar is a series of additions and doubling of points.

Let $P =(3, 10) \in E_{23}(1, 1)$. Then $2P =(x_3,y_3)$ is equal to:

$2P = P + P =(x_1,y_1)+(x_1,y_1)$

Since P = Q, the values of λ , x_3 and y_3 are given by:

$\lambda=((3x_1^2+a)/(2y_1)) \mod p$

$= ((3 \times 3^2+1)/(2 \times 10)) \mod 23 = 5/20 \mod 23$

$=4^{-1} \mod 23=6$

$x_3 =(\lambda^2 - x_1 - x_2) \mod p =(6^2 - 3 - 3) \mod 23$

$=30 \mod 23 = 7$

$y_3 = (\lambda (x_1 - x_3) - y_1) \mod p =(6\times (3 - 7) – 10) \mod 23$

$= -34 \mod 23 = 12$

Therefore, $2P =(x_3,y_3)=(7,12)$.

The multiplication *kP* is obtained by repeating the elliptic curve addition operation k times by following the same additive rules i.e. group law. See figure 4 given below.

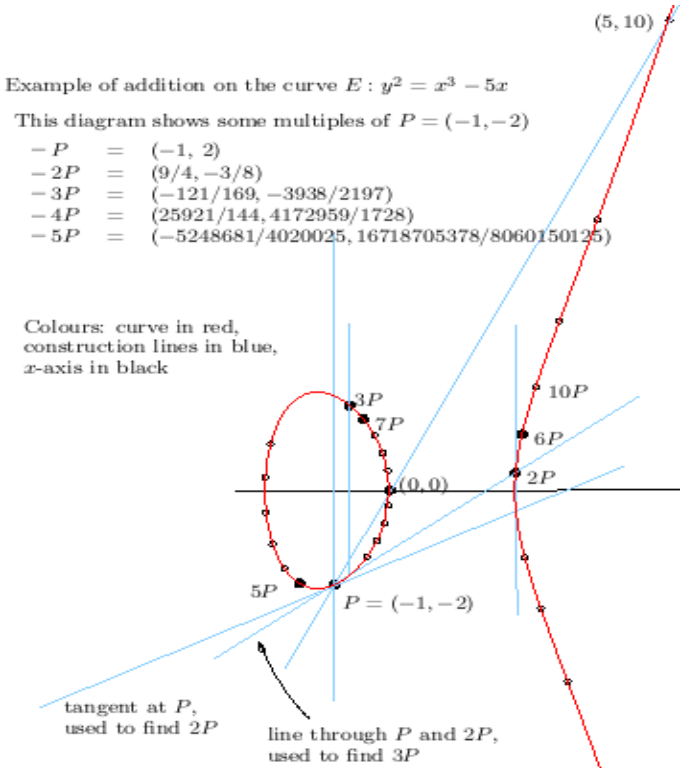

Figure 4: Group Law of Elliptic Curve

Here's an example of an elliptic curve, with some construction lines for the group law, which we have just studied. Some points on this curve, such as (0,0) and (-1,-2) are easy to find, but others such as (-5248681/4020025,16718705378/8060150125) would be difficult to find without using the group law.

11. ECC Encryption/Decryption

ECC can be used to encrypt plaintext messages, **M** , into ciphertexts, C, and decrypt ciphertexts into plaintext messages. The plaintext message **M** is to be encoded into a point P_m 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 $y^2 \pmod p = x^3+ax+b \pmod p$. We first convert the plaintext message **M** into a sequence of integers. Let **m** be a message (in fact integer) such that $0 \le m < p/100$. Calculate $x_i=100\mathbf{m}+\mathbf{i}$ for $\mathbf{i} \in [0,100]$. Compute each $s_i=x_i^3+ax_i+b$ for **i** in the range [0,100]. It is possible to test whether s_i is a square and compute its square root if it is. If s_i is a square, we have done it and we can use the point $P=(x_i,y_i)$ on our curve, where y_i is the root of s_i . The message **m** can then be obtained from P by simply taking

$[x_i]$. The probability that each s_i is a square is ½. Therefore, the probability that some s_i is a square is $1-2^{-100}$,which is extremely high. We could have used 10^k for some k>2 in place of 100 to increase this probability, but that is unnecessary.

11.1 Key Generation

1.Alice and Bob agree on a generator point $G=(x_g,y_g)$ and an elliptic group $E_p(a,b)$.
2.Alice chooses an integer n_a and calculates $P_a=n_aG=(x_a,y_a)$ according to addition law given in section 9.
3. Alice's public key is $P_a=(x_a,y_a)$ and his private key is n_a.
4.Bob also chooses an integer n_b and calculates $P_b=n_bG=(x_b,y_b)$ according to addition law given in section 9.
5.Bob's public key is $P_b=(x_b,y_b)$ and his private key is n_b.

11.2 Encryption

Alice wishes to send a message $P_m=(x_m,y_m)$ to Bob. He carries out the following steps.

1. Alice chooses a random number k.
2. He calculates $c_1=kG$ and $c_2=P_m+kP_b$.
3. Alice sends the $C_m=\{c_1,c_2\}$ as cipher text to Bob.

11.3 Decryption

Upon receiving the ciphertext pair (c_1,c_2) from Alice, Bob recovers the message as follows:

He multiplies c_1 by his private key n_b and subtract it from c_2. That is, he calculates $c_2-n_bc_1=(P_m+kP_b)-n_b(kG)=(P_m+kn_bG)-n_bkG$
$=P_m=(x_m,y_m)$

11.4 ECC Design Example

Alice chooses an elliptic curve
$y^2=x^3+ax+b \bmod p$
$y^2=x^3+537680305x+1059676324 \bmod 3946183951$
that is, a=537680305,b=1059676324, and p=3946183951.

The elliptic curve group generated by the above elliptic is $E_p(a,b)=E_{3946183951}(537680305,1059676324)$.

Assume Alice wants to send the message "http://www.ijcsns.org" to Bob. He uses Bob's public key to encrypt it. Alice assume the generator point G=(1152222263, 3133703258) $\in E_p(a,b)$. Suppose that Bob's secret key is n_b=2759936539 then his public key is $P_b=n_bG=(3539395206, 1802765602)$.

Alice chooses a random integer k=100 and computes the ciphertext pair of points P_c using Bob's public key P_b:

$P_c=[(kG), (P_m+kP_b)]$

The ciphertext calculated by Alice is $P_c=$
26101211923763490231111424883051438634068980126035950792933522533206494411563550211431013228111424889305143030829215595649509721001222702380545031972268584620086399648264696530085816237159595529021715838.

Bob recover the plaintext from the ciphertext using $P_m=(P_m+kn_bG)-[n_b(kG)]$, which is mapped to text message "http://www.ijcsns.org".

12. Security of ECC

Let *E* be an elliptic curve defined over a finite field and let, *P* be a point (called base point) on *E* of order *n* and k is a scalar. Calculating the point Q=*kP* from P is very easy and *Q=kP* can be computed by repeated point additions of P. However, it is very hard to determine the value of *k* knowing the two points: *kP and P.* This leads to the definition of **E**lliptic **C**urve **D**iscrete **L**ogarithm **P**roblem (ECDLP), which is defined as: "*Given a base point P and the point Q = kP, lying on the curve, find the value of scalar k, provided that such an integer exists*". The integer *k* is called the *elliptic curve discrete logarithm of Q to the base P*, denoted as $k=\log_P Q$.

The hardness of the elliptic curve discrete logarithm problem is essential for the security of all elliptic curve cryptographic schemes. The best general-purpose attack known on the ECDLP is the Pollard's rho algorithm, which has a fully-exponential running time of $O(\sqrt{p})$ where *p* is the largest prime divisor of *n*. To resist this attack, the elliptic curve parameters should be chosen so that *n* is divisible by a prime number *p* sufficiently large so that $\sqrt{p}$ steps is an infeasible amount of computation. If, in addition, the elliptic curve parameters are carefully chosen to defeat all other known attacks, then the ECDLP is believed to be infeasible given the state of today's computer technology. It should be noted that there is no mathematical proof that the ECDLP is intractable. That is, no one has proven that there does not exist an efficient algorithm for solving the ECDLP. Indeed, such a proof would be extremely surprising. For example, the nonexistence of a polynomial-time algorithm for the ECDLP would imply that P≠NP thus settling one of the fundamental outstanding open questions in computer science. Furthermore, there is no theoretical evidence that the ECDLP is intractable. For example, the ECDLP is not known to be NP-hard, and it is not likely to be proven to be NP-hard since the decision version of the ECDLP is known to be in both NP and co-NP.

13. Comparison of RSA and ECC

This section compares the public key sizes, activities involved in setting up cryptosystems, space complexities, digital signature sizes, signature signing

and verification running times, standards and interoperability for the ECC and RSA algorithms, encryption and decryption implementation speeds, and many more.

13.1 Public Key Size of RSA and ECC

An RSA public key pair consists of an ordered pair (n,e) where n is a composite number, called the modulus, and e is the public exponent. In a 1024-bit RSA system, n will have 1024 bits. A common value for the public exponent is $e=2^{16} + 1(=65537)$. Thus, an RSA public key would require 128 bytes for the modulus and (2+1=) 3 bytes for the public exponent. The total size is then 131 bytes.

An ECC public key consists of a point on the elliptic curve. Each point is represented by an ordered pair of element (x, y). For a 192-bit elliptic curve, the public key is then represented by two 24-byte numbers, giving a total key size of 48 bytes.

A method exists to reduce the size of the ECC public keys by almost a factor of 2. This method is called point compression and if we use point compression, a public key could be represented by using one 192-bit value and one additional bit. This would then require (24+1=) 25 bytes.

As can be seen from above, ECC provides a significant reduction in public key size. This reduction can be crucial in many constrained environments where large public keys are not possible.

The following table 2 gives the key sizes in bit that are said to be equal in terms of security.

Table 2: Relative public key sizes of RSA and ECC

Security Level	RSA	ECC
80 bit	1024	160
112 bit	2048	224
128 bit	3072	256
140 bit	4096	280
192 bit	7680	384
300 bit	21000	600

ECC with 600 bits practical, but RSA with 21000 bits not.

13.2 Setting Up of a Cryptosystem

In ECC a few system parameters are to be created as each ECC public key is only valid in the context of certain parameters. These parameters must be specified and transferred with the public key to the recipient. Creating the system parameters consists of selection of an underlying finite field for the cryptosystem and a representation for the elements in the field. Then an appropriate elliptic curve has to be chosen together with a base point on the curve.

There are very many approaches for selecting an appropriate elliptic curve for cryptography at least in theory [6]. They all tend to be mathematically very complicated and they have some limitations. So, it is perhaps most important mentioning at this stage that implementing elliptic curve cryptosystems can in fact be quite challenging without a good understanding of the mathematics of number theory and 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 and each user has his or her public/private key pair. The beauty of ECC is that these key pairs are easy to generate.

By way of comparison, the RSA cryptosystem requires few system parameters. The first stage of computing a public/private key pair consists of the user generating two primes of appropriate size and computing the public modulus **n** as their product. This part of the product can be computationally intensive. 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 Information). The calculation of the secret exponent or related information is insignificant when compared to the time required to generate the primes. The various system parameters, with sizes in bit, for the two cryptosystems are given in table 3.

Table 3: System parameters of RSA and ECC.

Parameter Name	RSA 1024-bit and e=65537	ECC 160-bit
System parameters	0	4x160+1=641
Public key	1027+17=1041	160+1=161
Private key	2048(2560 with CRT information)	160(801 with system parameters)

13.3 Space Complexity of RSA and ECC

The space complexity of an algorithm is a measure of how much storage is required for a computation. For different input size, there will be different amount of space. Here, we give the storage requirements in bytes of RSA with a 1024-bit modulus and an elliptic curve cryptosystem over GF(p) where p is 160 bits in length when making a rough comparison between the above two systems.

Table 4: Space complexity in byte when making a comparison between RSA and ECC.

Parameter Name	RSA with e=65537	ECC
System	0	81

parameter		
Public key	131	21
Private key	256 (320 with CRT information)	20 (101 with system parameters)

13.4 Signature Block Size

An RSA signature created with a 1024-bit public key consists of a single 1024-bit value. Thus, it can be represented in 128 bytes. Similarly, an RSA signature created with a 512 bit public key requires 64 bytes.

An ECDSA signature created using a 160-bit value consists of two 160-bit values. Thus, it can be represented using two 20-byte values. It is to be noted here that signature sizes cannot be reduced using point compression. Similarly, a signature created using a 192-bit curve requires 48 bytes.

Table 5: Signature block size

Algorithm	RSA 1024 bit	ECDSA 160 bit
Signature size	1024 bits	320 bits

13.5 Signing and Verification Running Time

This section compares the time taken to perform RSA signature signing and verification operations with that of ECC.

The first issue to consider is whether the implementation is in the software of hardware. For hardware implementation, even characteristic curves mostly allow the fastest implementations [4]. This is due to the fact that the underlying arithmetic for even characteristic curves can be implemented using fewer gates than the arithmetic for the odd characteristic curves or for RSA. Odd characteristic curves and RSA however can take advantage of the integer mathematical routines available on most computers and therefore they should be used for software implementation.

Using a small public exponent value of **e** (=65537), the RSA public key operation can be made very fast. The RSA private key operations (signature generation and decryption) are generally slower than the public key operations. The ECC private key operations are generally faster than the public key operations. This situation can be summarized in the table 6 given below.

Table 6: Signing and Verification Time

Activity	RSA 1024 bit	ECDSA 160 bit
Signing	384 ms (slow)	60 ms (fast)
Verification	17 ms (very fast)	120 ms (slower)

In addition, public and private-key operations differ in efficiency: for RSA, public-key operations are significantly more efficient than private-key ones, whereas for ECC, private-key operations are slightly more efficient than public-key ones. Therefore, the performance of one algorithm relative to another depends upon the profile of operations used by the application. Profiles that involve significantly more public-key operations than private-key and key generation operations will favor RSA over ECC. Other mixes will tend to favor ECC.

13.6 Software Attack on RSA and ECC

The level of effort for factoring integers and computing elliptic curve discrete logarithms is measured in a unit called MIPS year. The term MIPS year denotes the computational power of a MIPS computer utilized for one year; a million-instruction-per-second processor running for one year, which is about $3x10^{13}$ instructions executed [4]. It is worthy to note that a software attack on ECC appears to be relatively more difficult than that of software attack on RSA.

The following figure in table 7 shows the level of effort required for various values of **n** in bits to factor with current version of the GNFS and to compute a single elliptic curve discrete logarithm using the Pollard-rho method.

Table 7: Software Attack on RSA and ECC

RSA	ECC	MIPS years to attack
1024	160	10^{12}
2048	224	10^{24}
3072	256	10^{28}
4096	280	10^{31}
7680	384	10^{47}
21000	600	10^{81}

In November 2002, a 109-bit ECC encryption key was data mined with 10,000 computers running 24 hours a day for 549 days. For the binary field case, it was broken in April 2004 using 2600 computers for 17 months. A Certicom white paper reports that breaking a 160-bit key, which is the standard applied to most commercial ECC applications that Certicom uses, would be a hundred million times harder than breaking the 109-bit key.

13.7 Timing Attack on RSA and ECC

Recently, a new class of cryptanalysis aimed at a cryptosystem's implementation-specific weaknesses has attracted great interest. This kind of cryptanalysis attempts to compromise a cryptosystem by analyzing the time taken to execute cryptographic algorithms. Every

logical operation in a computer takes time to execute. The time required for different inputs may vary, forming a timing distribution. If this timing distribution is related to the secret key bits in the system, an attacker can work backwards to the input and thus the information can lead to reveal the secret keys.

A timing attack is an example of an attack that exploits the implementation of an algorithm rather than the algorithm itself. Information can leak from a system through measurement of time it takes to respond to certain queries. Reasons include performance optimizations to bypass unnecessary operations, branching and conditional statements, RAM cache hits, processor instructions such as multiplication division that run in non-fixed time, and a wide variety of other causes. This type of attack is primitive in the sense that no specialized equipment is needed. Timing attacks are often overlooked in the design phase because they are so dependent on the implementation.

The execution time for the square-and-multiply algorithm and modular exponentiation depends linearly on the number of "1" bits in the key. While the number of "1" bits alone is not nearly enough information to make finding the key trivially easy, repeated executions with the same key and different inputs can be used to perform statistical correlation analysis of timing information to recover the key completely, even by a passive attacker. Observed timing measurements often include noise. Nevertheless, timing attacks are practical against a number of encryption algorithms such RSA.

The most time consuming steps in ECC is the process of adding two elliptic curve points or doubling an elliptic curve point. Multiplication is implemented using the above operations. A widely used method for performing a scalar multiplication is the celebrated double -and–add method. The double-and-add method for multiplication certainly has computational time dependent upon the bit strings multiplied. An implementation of ECC using this algorithm for multiplications must be vulnerable to timing attacks.

The most obvious way to prevent timing attack is to make all operations take exactly the same amount of time. Unfortunately this is often difficult to implement. Fixed time implementations are likely to be slow; many performance optimizations are to be avoided since all operations must take as long as the slowest operation.

Another method proposed to counter timing attack is the timing equalization of multiplication and squaring. The time taken by the unit for the performance of multiplication and for the performance of exponentiation actions should be similar. Due to this quality, an attacker will not be able to learn if, when and how many multiplications and exponentiations are performed. The equalization can be caused by always performing both operations i.e. multiplication and exponentiation, regardless of the operation that is required at any given time. At any stage where one of the operations is required to run, both should be executed and the aftermath of the unnecessary operation is to be silently ignored. This technique prevents timing attacks against the exponentiation operations that are performed as a part of asymmetric encryption operations and which are subject to the most common attacks.

To avoid timing attack in the implementation of ECC, we have to look other algorithm that does not make use of the shift-and-add algorithm for performing multiplication. One such algorithm is found in the implementation of ECC written by Rosing [9]. The algorithm used is based upon table lookups in a fixed size array.

By intuition, both RSA and ECC are equally vulnerable to timing attacks

13.8 Management for the Future

According to Moore's law, the computing power increases exponentially. So, cryptographic key sizes have to be increased considerably. This makes it unlikely that today's 1024-bit RSA keys will still be considered secure 30 years from now. Taking the Moore's law into consideration and baring any unforeseen developments, RSA key sizes will increase at a faster rate than those of ECC.

As key sizes increase, so do the sizes of signatures and public keys, and so does the time required to perform cryptographic operations on a particular computing platform. This rate of increase will be considerably faster for RSA than it is for ECC.

Clearly, RSA cannot satisfy this requirement, and we are constrained to consider ECC as an alternative.

13.9 Platform Consideration

Before settling down to any cryptosystem, the computing power available on the target platforms must be taken into consideration. Given the security level given in table 2, we should ask our self whether is it possible for the platforms to support the required RSA key size? For example, if the required security level corresponds to 7680 bits of RSA, and the platform includes a server that generates keys for many users, then it may not be feasible to choose RSA. Similarly, if the required security level corresponds to 2048 bits of RSA and the platform is a constrained device, then it may take hours for it to generate an RSA key of that size. If RSA is not feasible for the given key size, then ECC is the best choice. The security level required for medium to long-term use will disqualify RSA for many applications.

Currently, ECC tends to be more useful in dedicated applications that involve resource-constrained platforms, do not require external interoperability and do not share a common infrastructure with other applications.

13.10 Encryption and Decryption Operating Speed

In this part, we compare the RSA and ECC algorithms in terms of encryption and decryption operating speeds for key sizes that are said to be equal in terms of security as stated in table 2 above. The message (="http://www.ijcsns.org") size was 21 bytes and encrypted output was 152 bytes in case of RSA and 203 bytes in case of ECC.

We performed five test runs of our experiment on our system Intel 1.6 GHz and 1GB of RAM in Sun Java SE 7 (Codename Dolphin). Absolute timings can vary widely. So, we give here relative timings only. The results of our test are tabulated in table 8 given below.

Table 8: Encryption and decryption operating speed

Algorithms	Encryption	Decryption
RSA 1024 ECC 160	03.04 ms 81.16 ms	31.51 ms 62.06 ms
RSA 2048 ECC 224	15.21 ms 111.08 ms	203.65 ms 98.71 ms
RSA 3072 ECC 256	16.86 ms 131.11 ms	703.21 ms 115.43 ms
RSA 4096 ECC 280	18.51 ms 145.00 ms	1594.01ms 195.08 ms
RSA 7680 ECC 384	31.96 ms 180.21 ms	10093.05 ms 244.80 ms

From the table, we conclude that the encryption process in RSA is optimal even for large key sizes such as 7680 bits. However, for decryption the time taken raises considerably. Both the encryption and decryption speeds of the ECC are optimal even for large key sizes. RSA with 21000 bit key size may no be practical to implement; thus forcing us to use ECC. So we conclude that the use of ECC will offer significant benefits over RSA when more security needs increase as operating speed of RSA with large key size increases exponentially.

13.11 Standard

Lately a number of standards have been published, each making various recommendations on the usage of ECC algorithm [8]. The elliptic curve cryptosystems have been considered as part of the various standard bodies including ANSI X9 and the IEEE. They are also included as a key agreement protocol in an Internet IETF draft [11].

RSA is used in many published and proposed standards. It is also used in many Internet protocols such as S/MIME, S-HTTP, and SSL.

14. Conclusion

This paper presented an in-depth comparison of RSA and ECC. Elliptic curves are believed to provide good security with smaller key sizes, something that is very useful in many applications. Smaller key sizes may result in faster execution timings for the schemes, which is beneficial to systems where real time performance is a critical factor.

We gave estimates of parameter sizes providing equivalent levels of security for RSA and ECC systems. These comparisons illustrate the appeal of elliptic curve cryptography especially for applications that have high security.

Acknowledgment

The first author would like to thank his daughter, *Java Compiler Soram* and son, *Chandrayan One Soram* for not disturbing him while he was engaged in coding and implementing the entire work in Java 7 in his system.

References

[1]W. Diffie and M. Hellman, "New Directions in cryptography", IEEE Transactions on Information Theory, volume 22, 1976.
[2]R.L. Rivest, A. Shamir, and L. Adleman, "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems",1977
[3]Tanenbaum A.S, Computer Networks, Prentice-Hall of India, New Delhi, 2006.
[4]Stalling W,Cryptography and Network Security,Prentice-Hall of India, New Delhi, 2001
[5]Kumanduri and Romero, Number Theory with Computer Applications, Prentice- Hall of India, New Delhi, 2001.
[6]Blake,Seroussi, and Smart, Elliptic Curves in cryptography,Cambridge University Press,1999.
[7]N.P. Smart,The Discrete logarithm Problem on Elliptic curves of Trace One, Journal of Cryptology, Vol. 12,1999
[8] http://www.certicom.com/
[9] Michael Rosing, "Implementing Elliptic Curve Cryptography", from "http://www.manning.com/rosing/"
[10]Agrawal, Kayal, Saxena, "PRIMES is in P", retrieved from "http://www.cse.iitk.ac.in/"
[11]http://en.wikipedia.org/

Ranbir Soram, studied at Coimbatore Institute of Technology, Coimbatore, Tamil Nadu, is working as a lecturer in Computer Science and Engineering at Manipur Institute of Technology, Takyelpat, Imphal, India. His field of interest includes network security, neural network, genetic algorithm etc. In his spare time, he writes programs in Java.

Memeta Khomdram is working at Department of Electronics Accreditation of Computer Courses Centre, Akampat, Imphal, India. She diligently read and commented on every page of this paper, at least twice while her two small children were engaged in watching Pogo and Arirang.