The Simulation of 256-bit Elliptic Curve Cryptosystem

Nur Azman Abu, Nur Azlina Zahari, Ida Zuraida Zakaria
++606-252 3578
azman@mmu.edu.my
Faculty of Information Science and Technology
Multimedia University
Jalan Ayer Keroh Lama,
75450 Bukit Beruang,
Melaka, Malaysia

Abstract

Elliptic Curve Cryptosystem have recently come into strong consideration, particularly by standards developers, as alternatives to established standard cryptosystem such as RSA public key. Elliptic Curve Cryptosystem is “the next generation” of public key cryptography, providing greater strength, higher speed, smaller keys than established systems.

An elliptic curve operation involves a sequence of elliptic curve additions, and each addition consists of several arithmetic operations in the finite field. The premise is that an elliptic curve 256-bit cryptosystem offers better security than of 1024-bit RSA.

The opportunity to conveniently generate Elliptic Curve Cryptosystem using commercial Mathematical Software has now become feasible. The purpose of this paper is to show how the powerful computer algebra system Maple V can be used to explore and visualize Elliptic Curve Cryptosystem.

The paper begins with some general background of Elliptic Curve Cryptosystem. Cryptographic operations tend to be highly CPU intensive, particularly public key operations. Then this paper discusses the simulation on Elliptic Curve Cryptosystem for 256-bit key length.

1 General Background Of Elliptic Curve Cryptosystem(ECC)

Elliptic Curve Cryptosystems is categorized as a public key cryptosystems. In a public-key cryptosystem, the abilities to perform encryption and decryption are separated. The encryption rule employs a public key E (that is $E = k$), while the decryption rule requires a different (but mathematically related) private key D (that is $D = r$). Knowledge of the public key allows encryption of plaintext but does not allow decryption of the ciphertext. Once a person publishes his/her public key, then anyone can use the public key to encrypt messages especially intended for that person. The private key is kept secret so that only the intended individual can decrypt the ciphertext.

For users to be able to understand the concept of ECC, they must first look into the background of elliptic curve. Elliptic curve cryptosystems were introduced in the papers of Koblitz [9] and Miller [10]. This part will provide an intuitive introduction to Elliptic

Curves and how they are used to create a secure and powerful cryptosystem. The readers are suggested to see [7] [8] [9] [10] for further overview.

2 General Background Of Elliptic Curve

An elliptic curve is simply the locus of points in the x-y plane that satisfy an algebraic equation $y^2 = x^3 + bx + c$ of the form. Each choice of numbers b and c yields different elliptic curves. The value of x, y, b and c may be from any field, namely complex number, real number, finite number and so on [2]. An example of elliptic curve is as shown below in Figure 1.

2.1 Elliptic Curve Groups Over $\mathbf{F}_p$

Calculations over the real numbers are slow and inaccurate due to round-off error. Cryptographic applications require fast and precise arithmetic; thus elliptic curve groups over the finite fields of $\mathbf{F}_p$ and $\mathbf{F}_{2^m}$ are used in practice.

The finite field $\mathbf{F}_p$ is comprised of the set of integers $\{0, 1, 2, \ldots, p-1\}$. Each such integer is represented by a binary string of length exactly $t = \lceil \log_2 p \rceil$ consisting of the binary representation of the integer padded on the left with the appropriate number of 0's.

An elliptic curve with the underlying field of $\mathbf{F}_p$ can be formed by choosing the variables b and c within the field of $\mathbf{F}_p$. The elliptic curve includes all points (x, y), which satisfy the elliptic curve equation modulo p (where x and y are numbers in $\mathbf{F}_p$).

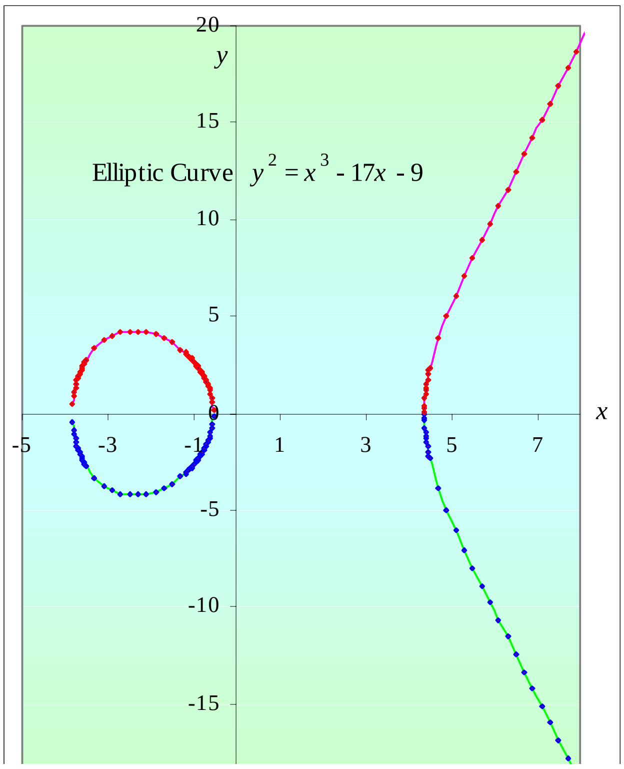

Figure 1. *Elliptic Curve* $y^2 = x^3 - 17x - 9$

If $x^3 + bx + c$ contains no repeating factors (or, equivalently, if $4b^3 + 27c^2 \bmod p \neq 0$), then the elliptic curve can be used to form a group. An elliptic curve group over $\mathbf{F}_p$ consists of the points on the corresponding elliptic curve, together with a special point $\boldsymbol{O}$, called the point at infinity. There are finitely many points on such an elliptic curve [3].

3 ECC Engine

ECC engine can be divided into three parts, which are:

- *Public Key Generation* – this include prime number p generation, elliptic curve generation, initial point $\boldsymbol{P}$ generation, public key k and generation of point $k\boldsymbol{P}$.

- *Encryption* – this include the generation of secret key r, generation of point $r\boldsymbol{P}$ and point $rk\boldsymbol{P}$.
- *Decryption* – including generation of point $rk\boldsymbol{P}$ (this value should be same as point $rk\boldsymbol{P}$ generated during encryption) and also involved in solving linear equations (refer section 3.3).

3.1 Public Key Generation

This process is will be done by the receiver. Firstly the receiver have to generate prime number 256-bit pseudo-randomly. This can be done by choosing a number, x, randomly using rand function. In order to generate 256- bit number, p, a simple formula is used:

$$p = 2^{256+128x} \bmod 2^{256}$$

To check whether the number is prime or not, function isprime is used. If the number is not prime, function netxprime may be used to find next prime number.

```
> flag: = 0;
> while flag < > 1 do
>   x: = rand()/10.0^12;
>   randp: = trunc(2^(256 + 128*x)) mod 2^256;
>      if isprime(randp) < > true then
>         nextprime(randp);
>         n: = %;
>         p: = n;
>      else
>         p: = randp;
>      fi;
```

The prime number 256-bit p, should fulfill a criteria, where p = -1 mod 4.

```
>   test: = p mod 4;
>   if test  = 3 then
>       flag: = 1;
>       break;
>   else
>     flag: = 0;
>   fi;
> od;
```

Sample output:

```
> p: = 55755856259966275371575697195005993278284027256745433338915789625709987627063.
```

The next process is generating elliptic curve, $y^2 = x^3 + bx + c$. Value b and c are generated randomly using rand function. It is also recommended to take prime numbers for b and c.

```
> b: = rand(p);
> b();
> c: = rand(p);
> c();
```

Sample output:

```
> b: = 17890289192135612607352510730917250551662575378703260594827665952843024596403
> c: = 42138790892025581984217012130179718041978275120415341335913239520254182879379
```

If for the chosen values *b* and *c* result in $4b^3 + 27c^2$ not equal to zero then they are accepted. Otherwise, search for the next suitable pair.

```
> ans: = 4*(b^3) + 27*(c^2);
> while ans = 0 do
>     b();
>     c();
>     ans: = 4*(b^3) + 27*(c^3);
> od;
```

The next step is finding the initial point, $\boldsymbol{P}(x_{0,} y_0)$. The computations involved are as follows:
Step 1: Find the *x* value randomly (in the range of *p*, $0< x< p$).
Step 2: Test *x* value to see if $z = x^3 + bx + c$ mod *p* is ***quadratic residue*** (*q*) by applying Euler's criterion.
Step 3: Using Euler's criterion, find $\boldsymbol{q_r}$ as follows:

$$\boldsymbol{q_r} = z^{(p-1)/2} \textbf{ mod } \boldsymbol{p}$$

If $\boldsymbol{q_r} = 1$ then it is a quadratic residue modulo *p*.
Step 4: Then, the formula to have the square roots of a quadratic residue *z* (also refers to y^2) is applied.

$z^{(p+1)/4}$ ***mod p*** (this is due to the nature of prime, $p = -1$ mod 4).
Step 5: Solve for ***y*** using the formula $z^{(p+1)/4}$ ***mod p***. [5][6]

In Maple V, the computation is as follows:

```
> # Find x randomly
> flag: = 1;
> while flag < > 0 do
>     x1: = rand(p);
>     x: = x1();
>     z: = x^3 + b*x + c mod p;
>     qr: = z&^((p-1)/2) mod p;
>     if qr = 1 then
>        flag: = 0;
>     fi;
> od;
>
> #To check the nature of prime
> s: = p mod 4;
> if s = 3 then
>    ex: = (p + 1)/4;
> fi;
>
> #To find the value of y
> y: = z&^ex mod p;
```

Sample output:

```
> # To find the initial point on the curve P (x0, y0).
> x0 : = 25310347367171583082854772584115503678456365680274536128411346757957478329205
> y0 : = 46751032657002880765202883369103632744069530285461100546191761999841908459679
```

The fourth step is generating the secret key, *k*. The value k is generated randomly using rand function.

```
> k: = rand(9999);
```

Sample output:

```
> k := 4527
```

The fifth step is calculating ***kP***. To do this a function named scalarmult is defined.

```
> scalarmult:=proc(x,y,k)
> global p,b,c;
> local flag,rx,ry,tx,ty,i;
>
> tx:=x;
> ty:=y;
>
> #To add point repetitively
> for i from k by -1 to 2 do
>      (rx,ry) :=addpoint(x,y,tx,ty);
>      tx:=rx;
>      ty:=ry;
> od;
> RETURN(tx,ty);
> end:
```

Sample output:

```
> #Generate kP (kPx0 , kPy0)
> kPx0:= 55399931627495512984169137377907510175521866970968028926605306710404685111880
> kPy0 :=14256969885540287872114700569909650995703486166745002948833053080582261726557
```

Function addpoint in scalarmult function is a user-defined function. This function (addpoint) adds two points on the curve in order to get the third point, which is also on the curve.

```
> addpoint:=proc(tempX,tempY,x,y)
> global p,b,c;
> local lamda,semX,semY,resultx,resulty,xans,yans,mult;
>
> #To check the 2nd point is not the inverse of the 1st point
> #and also the 1st point is not (x,0)
> if tempX=x and tempY=-y and not(tempY=0)then
>    printf("Point at infinity");
>
> #Doubling the point when both points are the same
> elif tempX=x and tempY=y then
>      mult:=multiplyInverse(2*tempY,p);
>      lamda:= (3*tempX^2 + b)* mult;
>      lamda:= lamda mod p;
>
>      #Find the value for x
>      semX:= lamda^2 -tempX-x;
>      semX:= semX mod p;
>      resultx:=semX;
>      #Find the value for y
>      semY:=  lamda* (tempX-resultx)-tempY;
>      semY:= semY mod p;
>      resulty:= semY;
> else
>     #Adding two distinct point
>       xans:= tempX-x;
>       yans:= tempY-y;
>
>       lamda:= yans * (multiplyInverse(xans,p));
>       lamda:= lamda mod p;
>
>       #Find the value x
>       semX:= lamda^2 - tempX - x;
>       semX:= semX mod p;
>       resultx:=semX;
>
>       #Find the y value
>       semY:= lamda * (tempX-resultx)-tempY;
>       semY:= semY mod p;
>       resulty:= semY;
>  fi;
>
>   RETURN(resultx,resulty);
>
>   end:
```

Function multiplyinverse in addpoint function is a user-defined function. This function (multiplyinverse) finds multiplicative inverse of the number using formula $x^{(p - 2)}$mod p. Finally, p, b, c, $\boldsymbol{P}$ and $k\boldsymbol{P}$ that are the public keys will be passed to the sender to do the encryption [4]

3.2 Encryption

The sender will do this process. The steps involved are as follows;
Step 1: Generate secret key r.
Step 2: Calculate $r\boldsymbol{P}$ ($\boldsymbol{P}$ is the initial point).
Step 3: Calculate $rk\boldsymbol{P}$.
Step 3: Get the plain text a_1 and a_2.
Step 4: Generate cipher text d and $e_.$ using the formula given

$$d\ (\ d = rkx_0 * a_1)$$
$$e\ (\ e = rky_0 * a_2)$$ [4]

```
>  encryption:=proc()
>
>   global p,b,c,rPX,rPY,
>          x,y,d,e,
>          rkPX,rkPY;
>
>   local a1,a2,r;
>
>   d:=0;
>   e:=0;
>
>   #Step1: Generate random number r := rand(9999)
>   r:=keyr();
>
>   #Step2: Calculate rP
>  scalarmult(x,y,r);
>  rPX,rPY:=%;
>
>  #Step3: Calculate rkP
>  scalarmult(kPX,kPY,r);
>  rkPX,rkPY:=%;
>
>  #Step4:To read the message
>  message();
>  a1,a2:=%;
>
>  #Step5:To encrypt data
>  d:=rkPX*a1 mod p;
>  e:=rkPY*a2 mod p;
>
>  RETURN(d,e);
>  end:
```

Sample output:

```
> r :=6459.

> #To calculate rP( rPx0, rPy0)
> rPx0:=37619123447363204419848893204707252554559986357917974750508586519571899861509
> rPy0:=14093420069777191955170789677369508162053042007492963489365378717677341268050

> #To calculate rP( rkPx0, rkPy0)
> rkPx0:=50595393987755189892258981015940966687614245775209437530830285060998411146289
> rkPy0 :=21854521793814014707227837590700606941317251445886882671669683288918913499812

> #Read the  secret message  a1 and a2
> a1:=635709568337115114395456457424789965157
> a2 :=62517676918298663777362322669257586755

 > #Encrypt the message…results in d and e
> d:=24563675628938377003564243352417299208337453806978857651979551486638942429244
> e:=44131538599835625289098055726587201067331396076879321899194512077774738690628
```

3.3 Decryption

This process is done by the receiver. The steps involved are as follows:

Step1 : Calculate $kr\boldsymbol{P} = (krx_0, kry_0)$. This value should be the same as rkP calculated by sender during encryption process. This value is calculated by decrypt procedure and then passed to this procedure.

Step 2: Get the cipher text. These values are referred as d and e.

Step 3: Solve for the equation

$$kr\boldsymbol{P}X * a_1 \equiv d \pmod p$$
$$kr\boldsymbol{P}Y * a_2 \equiv e \pmod p$$ [4]

```
> decryption:=proc()
>
> global d,e,rPX,rPY,p,k;
> local krPX,krPY,tempx,tempy,resultx,resulty;
>
```

```
> #Calculate krP
> scalarmult(rPX,rPY,k);
> (krPX,krPY):=%;
>
> #solve the equation for decryption
> tempx:= krPX&^(p-2) mod p;
> resultx:=(tempx * d) mod p;
> tempy:=krPY&^(p-2) mod p;
> resulty:=(tempy * e) mod p;
>
> RETURN (resultx,resulty);
> end:
```

Sample output:

```
> #To calculate rP( krPx0, krPy0)
> krPx0:=5059539398775518989225898101594096668761424577520943753083028506099841146289
> krPy0:=2185452179381401470722783759070060694131725144588688267166968328891891349812

> #Decrypt the message back to plain text c1, c2 which is the secret message pair a1, a2.
> c1:=6357095683371151143954564574247899651 57
> c2 := 6251767691829866377736232266925758675 5
```

4 Efficiency of ECC

Public-key cryptographic systems have proven to be effective and more manageable than symmetric key systems in a large number of scenarios. When talking about the efficiency of a public-key cryptographic system, there are three factors to be considered:

- *Computational overheads*: how much computation is required to perform the public key and private key transformation.
- *Key size*: How many bits are required to store the key pairs and associated system parameters.
- *Bandwidth*: How many bits must be communicated to transfer an encrypted message or a signature.

ECC offers significant efficiency savings due to its added strength-per-bit. These savings are advantageous in many applications, particularly when computational power, bandwidth, or storage space is limited.[1] Advantages of ECC in a constrained environment include the following:

- Shorter keys - 161-bit ECC is about or at least better than 1024-bit RSA.
- Shorter signatures – 322-bit ECC is or at least better than about 1024-bit RSA.
- Shorter certificates – 256-byte RSA is about 62-byte ECC.
- Simpler generation of key pair, given a valid set of domain parameters.
- Large proportion of the signature generation and encrypting transformations can be pre-computed such that computational overhead can be saved. [12][13]

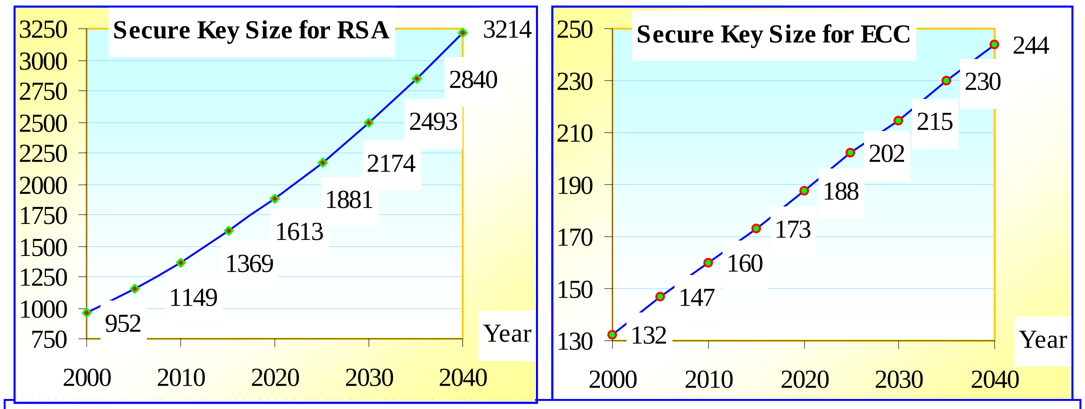

Figure 2 shows the equivalent key length(in the number of bits) to be considered sufficiently secure for RSA and ECC.

5 Security of ECC

The emphasis in terms of security is placed on theoretical security – breaking a public key system in general. The best algorithm known to date for the ECC with DLP(Discrete Logarithm Problem)in general is the Pollard rho-method[14] which takes about $O(\sqrt{\pi\ p/2})$ steps, where a *step* here is an elliptic curve addition.[15]

Then there is also the general–purpose algorithms that always succeed in solving the integer factorization problem(thus solving RSA) run in *subexponential* time, which means the problem should still be considered hard but not as hard as those problem which admit only fully exponential time. Let n, the product of 2 large primes, be the modulus of RSA. Currently, the running time for the best general algorithms to factor n is:

$O(\exp\{(c + o(1))(\ln n)^{1/3}(\ln \ln n)^{2/3}\})$ for constant $c = 1.9229$.

Simply put, the elliptic curve discrete logarithm problem is currently considered harder than integer factorization problem.

In [12], key size recommendations are presented for several classical cryptosystems based on the hyphotheses:

i. Relative to the breaking a 56-bit DES key would cost 5*105 Mips Years(or equivalent to using USD 50 millions machine within 2 days) in 1982
ii. Moore's Law projects that computing power per chip doubles every 18 months
iii. The amount of computing power and RAM would double every 18 months per USD 1
iv. Monetary budget available to the ones breaking cryptographic keys doubles every ten years due to the trend of US GNP doubling on average every ten years
v. On average it takes 18 months for cryptanalytic developments affecting classical asymmmetric system to become twice as efective without any major breakthrough,

.

The graph in Figure 2 depicts the summary of the key length to be considered sufficiently secure for RSA and ECC.

6 Conclusions

ECC provides greater efficiency than either integer factorization systems or discrete logarithm systems, in terms of computational overheads, key sizes, and bandwidth. In implementations, these savings mean higher speeds, lower power consumption, and code size reductions.

The theory behind ECC is mathematically complex compared to RSA. RSA is a public-key system based on integer factorization problem. On the other hand, ECC is a public-key system based on elliptic curve discrete logarithm problem, which is much harder than integer factorization problem. Since ECC provides a greater efficiency compared to other public-key systems, it is beneficial to learn the mathematical background of ECC.[12]

The use of Maple V has eliminated the problem of using large numbers since Maple V supports large numbers which is especially structured for decimal base. Besides, it also can do these large computations in reasonable time. Apart from that, Maple V provides a range of functions that allows more practical example to stimulate ECC.[11]

7 References

[1] Certicom Corp., “Current Public-Key Cryptography System”, Certicom Whitepaper 2, April 1997.
http://www.certicom.com/research/download/paper2wd.zip

[2] Charles Daney, “Elliptic Curves and Elliptic Functions”, 1996.
http://www.best.com/~cgd/home/flt/flt03.htm

[3] Certicom Corp., “ECC Tutorials”, Certicom Tutorial, 1997.
http://www.certicom.com/ecc/enter/index.htm

[4] “An Introduction Course to Elliptic Curve Cryptosystems”, 1998.
http://ge.ge.kochi-ct.ac.jp/%&Etakagi/crypto/

[5] Waterloo Maple Inc., “Waterloo Maple Advancing Mathematics”, Maple Home.
http://www.maplesoft.com

[6] M.B. Monagan, K.O. Geddes et al, “Maple V Programming Guide”, Springer Waterloo Maple Inc, 1998.

[7] Randall K. Nichols, “ICSA Guide to Cryptography”, McGraw-Hill Companies, Inc., 1999.

[8] Douglas R. Stinson, “Cryptography, Theory and Practise”, CRC Press LLC, 1995.

[9] Neal Koblitz, “Introduction to Elliptic Curves and Modular Form”, Second Edition, Springer, 1993.

[10] V. Miller, "Uses of elliptic curves in cryptography", *Advances in Cryptology CRYPTO '85,* Lecture Notes in Computer Science, **218** (1986), Springer-Verlag, 417-426.

[11] Ida Zuraida, Nur Azlina, “Elliptic Curve Cryptography Multimedia System (Simulation)”, Thesis for Bachelor of Information Technology, Multimedia University, 28th May 2000.

[12] Arjen K. Lenstra, Eric R. Verheul, “Selecting Cryptographic Key Sizes”,Public Key Cryptography Conference (PKC2000) Melbourne, Australia, January 2000.

[13] Don B. Johnson(Certicom) "ECC, Future Resiliency, and High Security Systems", PKS '99, March 30, 1999
http://www.certicom.com/ecc/wpaper.htm/ECCFut.zip

[14] J. Pollard, “Monte Carlo methods for index computation mod *p*”, *Mathematics of Computation*, Volume 32, pages 918-924, 1978.

[15] Aleksandar Jurisic, Alfred J. Menezes, “Elliptic Curves and Cryptography”, Certicom Whitepaper 4,
http://www.certicom.com/research/download/ paper3wd.zip