Implementation of Elliptic Net Scalar Multiplication Computation for NIST P-192 Curve using Python

Zuren Razali¹, Norliana Muslim*², Saliyah Kahar¹

¹Faculty of Communication, Visual Art and Computing, Universiti Selangor.
²Faculty of Engineering and Life Sciences, Universiti Selangor.

*norliana_muslim@unisel.edu.my

Abstract. Elliptic curve cryptography is one of the most efficient public-key cryptosystems compared to the Rivest-Shamir-Addleman scheme. One of the methods to compute elliptic curve scalar multiplication is division polynomials which utilize the non-linear recurrence relation also known as the elliptic net. Previous studies were related to elliptic curve scalar multiplication via elliptic net which implemented the use of C++, GP/PARI or Java programming. This study aims to implement the Python programming in the SageMath compiler for computing elliptic net scalar multiplication based on the National Institute of Standards and Technology curve of the type P-192. The implementation could be seen at three main processes which are the scalar decomposition, the blocks of elliptic net generation namely the initial and final blocks, as well as scalar multiplication computation. The base point and prime field of the short Weierstrass curve with a large scalar are the parameters used in the process. The results showed that the implemented programming was easier while working with a large prime number field, vulnerable to errors and can be easily verified. The implemented programming can also be used to compute scalar multiplication on other standardizations of elliptic curve cryptography such as Brainpool 384-bit prime field or NUMS 256-bit prime field curves.

1. Introduction

Elliptic curve cryptography (ECC) is asymmetric cryptography, or also known as the public-key cryptosystem where every user uses two keys: public and private keys for encryption and decryption. ECC provides equal security to the Rivest-Shamir-Addleman (RSA) scheme with smaller key sizes, as shown in Figure 1. Thus, ECC is faster and requires less hardware than RSA, as mentioned in [1]. However, researchers from [2] stated that the security of ECC also depends on the parameters and not only on the key length. A previous study in [3] highlighted the basic requirement for generating a secure parameter of the elliptic curve (EC) with the order of the curve must be prime or nearly prime, and the curve must be immune to special attacks. According to [4], the security of RSA is based on the integer factorization problem that takes sub-exponential time to be solved [5] while ECC is based on the EC discrete logarithm problem which takes full exponential time, which leads to less memory and suitable in-memory constrained devices.

ECC offers more protection than RSA with smaller key sizes [4]. Figure 1 shows the comparison between RSA and ECC with the National Institute of Standards and Technology (NIST) recommended security bit let level [6].
Figure 1 depicts that ECC used fewer keys compared to RSA with the same security. Scalar multiplication (SM) is a major operation in ECC applications and protocols, and SM is defined as follows: Definition 1 [7]: SM is the operation to compute n-multiple of points in the EC group. The process is denoted by \( Q = nP = P + P + P + \ldots + P \) (for n times), where n is the positive integers while P and Q are points on the EC. In the literature, several implementations of SM computations were conducted using C++ [8], GP/PARI [9] and Java [10]. However, the implementation using Python has not yet been explored. Thus, the main purpose of this paper was to implement the computation of EC SM via elliptic net (EN) using SageMath [11] which is a mathematical software that utilizes Python language, and it is actively developed and optimized by hundreds of skilled software engineers. This paper is organized as follows. Section 2 reviews the EC Weierstrass together with the arithmetic operation. Section 3 presents the methodology then, the Python implementation of SM computation via the EN is depicted in Section 4, including the implementation result. Finally, the study outcomes are concluded in Section 5.

2. Literature Review

This section presents several significant concepts related to the EC Weierstrass and the point arithmetic.

2.1. Elliptic Curve Weierstrass

A short Weierstrass curve is denoted by \( y^2 = x^3 + Ax + B \) over the prime field or \( F_p \), where \( p \) is a large prime number or known as mod \( p \). The modular value illustrates the key size for the EC system. The choice of the base point \( G \) in ECC is the prime step for its security [12]. The value of the order gives the core security levels and this defines the number of possible values before its cycle [13]. Therefore, it is best if the order of the curve is a large prime number [14]. The algorithm of the base point was discussed in [15]. Example 1:

Let \( E \) be the curve \( y^2 = x^3 - 8x + 4 \) over \( F_{29} \). The command to plot the EC over \( F_{29} \) is depicted below and the output is illustrated in Figure 2.

```
sage: E=EllipticCurve(GF(29),[-8,4]);E
Elliptic Curve defined by y^2 = x^3 + 21*x + 4 over Finite Field of size 29
sage: Ep = plot(E)
sage: t1 = text("P = (22,23)", (22.2,23.6)); t2 = text("10P = (10,5)", (10.2,5.6)); t3 = text("32P = infinity", (0.2,1.6)); t4 = text("P+Q = (9,9)", (9.2,9.6))
sage: show(Ep+t1+t2+t3+t4)
```

2.2. Arithmetic Operations

There are operations of addition and doubling a point on the EC by using the rule of chord and tangent. Figure 3 shows an example of operations on the EC over the real number field.
Figure 2. Curve $y^2 = x^3 - 8x + 4$ over $F_{29}$

Figure 3. Geometric description

From Figure 3, when a line connects two points, $P$ and $Q$ on the curve, it will intersect the curve at the third point, $R$ and the reflects of the x-axis is the point $P+Q$. The doubling of a point, $S$ can be done by drawing a tangent line that passes through the point, $T$ and it will intersect at another point on the curve, and the reflects is the point doubling labeled by $2S$.

3. Methodology

This section explains how the elliptic net SM (ENSM) can be computed using division polynomials. Apart from SM computation, division polynomials are also used to study the field generated by torsion points [16] and implemented in Schoof’s algorithm for counting points on the EC [17].

3.1. Division Polynomials of Weierstrass

Let $P = (x_1, y_1)$ be a point of the Weierstrass curve, then the first two initial values of the EN are $W_0 = 0$ and $W_1 = 1$. The next three terms of the EN follow the equation from division polynomials, such that:

$$W_2 = 2y_1$$  \hspace{1cm} (1)
$$W_3 = 3x_1^3 + 6ax_1^2 + 12bx_1 - a^2$$ \hspace{1cm} (2)
$$W_4 = 4y_1(x_1^6 + 5ax_1^4 + 20bx_1^3 - 5a^2x_1^2 - 4abx_1 - 8b^2 - a^3)$$ \hspace{1cm} (3)

For $n \geq 5$, $\psi_n$ can be defined recursively using:

$$W_{2n+1} = W_{n+2}W_n^3 - W_{n+1}W_{n-1}^3(n \geq 2)$$ \hspace{1cm} (4)
$$2y_1W_{2n} = W_n(W_{n+1}W_{n-1}^2 - W_{n-2}W_{n+1}^2)(n \geq 3)$$ \hspace{1cm} (5)
3.2. Elliptic Net Scalar Multiplication

The following theorem represents the ENSM upon the Weierstrass curve: Theorem 1 [18]: Let \( W_\eta \) be the elliptic net. If \( P = (x_1, y_1) \) is a point on short Weierstrass of the type \( y^2 = x^3 + Ax + B \) over \( F_p \), then the ENSM is generated by,

\[
nP = \left( x_1 - \frac{w_{n+1}w_{n+1} - w_{n+2}w_{n+1}^2}{4y_1w_n^2}, \frac{w_{n+1}w_{n+1} - w_{n+2}w_{n+1}^2}{4y_1w_n^2} \right)
\]

\( n = 622810902072934271145388709891269731448017030948024451072 \)

\( B = 6277101735386680763835789423207666416083908700390324961279 \)

\( x = 6024628237568865675821348058752611191669897636884684818 \)

\( y = 17405032293220314045885352280219410364023488927386650641 \)

\( p = 245515554600843817740293915197451784769108058161191238065 \)

\( \eta = 17405032293220314045885352280219410364023488927386650641 \)

\( n = 622810902072934271145388709891269731448017030948024451072 \)

The running environmental requirements of this research are described as follows: Intel Core i-7 8565 CPU 1.80 GHz, 64-bits and memory of 8 GB.

4. Implementations Results

This section shows the implementation of ENSM computation using Python language. Using the value of \( n \) stated in Section 3.4, the Python command for the scalar decomposition is as follows:

```python
sage: n = 622810902072934271145388709891269731448017030948024451072
sage: def initial_time(n,t=0):
    ...
    if n==1:
    ...
    return t
    ...
    else:
    ...
    return initial_time(n//2,t+1)
    ...
    t = initial_time(int(n))
    ...
    m = [n]
    ...
    for j in range(1,t+1):
    ...
    m.append(i)
    ...
    m[j]=floor(m[j-1]/2)
```
The command to generate Weierstrass’s division polynomials using NIST P-192 parameters is shown below:

```sage
A = -3
B = 2455155546008943817740293915197451784769108058161191238065
p = 6277101735386680768733578942320766416083908700930324961279
x = 60246282375688656785213480587526111916698976636884684818
y = 17405032939262575552280219410364023488927386650641
W = [0%p,1%p,(2*y)%p,(3*x^4+6*A*x^2+12*B*x-A^2)%p,(4*y)*(x^6+5*A*x^5+4+20*B*x^3-A^3))%p,0]
W[5] = ((W[4]*W[2]^3)-(W[1]*W[3]^3))%p
aa = (W[2])^-1
```

Using values generated previously, the EN block centered at \( n \) can be computed as:

```sage
S = [0,0,0,0,0,0]
P = [0,0,0,0,0,0]
V = [-W[2],-W[1],W[0],W[1],W[2],W[3],W[4],W[5]]
for j in range(1,t+1):
    add = m[t-j]%2
    for i in range(0,6):
        S[i]=(V[i+1]^2)%p
        P[i]=(V[i]*V[i+2])%p
    for i in range(1,5):
        if add==0:
            V[2*i-2]=((S[i-1]*P[i])-(S[i]*P[i-1]))%p
            V[2*i-1]=(((S[i-1]*P[i+1])-(S[i+1]*P[i-1]))*aa)%p
        elif add==1:
            V[2*i-2]=(((S[i-1]*P[i+1])-(S[i+1]*P[i-1]))*aa)%p
            V[2*i-1]=((S[i]*P[i+1])-(S[i+1]*P[i]))%p
```

The command to produce \( nP \) using EN block center at \( n \) and Equation (6) is depicted below:

```sage
Xnp=(x-((V[2]*V[4])/V[3]^2))%p
Ynp=((V[2]^2*V[5]-V[4]^2*V[1])/(4*y*V[3]^3))%p
print(n,"P=",(Xn,Yn))
```

Finally, the multiple points can be verified using the following coding:

```sage
(Yn^2)%p===(Xn^3+A*Xn+B)%p
```

The SM algorithm has been implemented using Python with a scalar \( n \) of 192-bits with Hamming weight 100. The parameters used was mentioned in Section 3.4. The first eight terms of scalar that were decomposed for \( t = 0,1,2,\ldots,191 \) and \( \mu_0 = 1 \) referred to the initial block centered at \( W_t \). The Boolean value was “0” when \( \mu_t \) was even and “1” when \( \mu_t \) was odd. Through the process of double and double add it will reach the final block center at \( W_n \) as shown in Table 1.

| \( W_{n-3} \) | 20253818502892235071350649477180558242705731159642014634 |
| \( W_{n-2} \) | 564907704230981388716698781112881815333364006800057 |
| \( W_{n-1} \) | 116471486034088276343538746801748293142142569454846013632 |
| \( W_n \) | 4772841145887749764610636770332606771865874030264361544 |
| \( W_{n+1} \) | 523798811342961512563684332806072378268075533689879214604498 |
| \( W_{n+2} \) | 259465651626436588995879874889779547001994634143574960 |
| \( W_{n+3} \) | 953096182057793846753497068965069860265630310479675171760 |
| \( W_{n+4} \) | 230383915483270042704471700193078608769690120632480512094 |
From Table 1, only five terms denoted by \( W_{n-2}, W_{n-1}, W_n, W_{n+1}, W_{n+2} \) were used in the multiple point formulae in Equation (6). Then, \( 6228109020272934271145388709891269731448017030948024451072p \) are
\[
\begin{align*}
x & \equiv 220822597778521054005972346426102517868443540208626439220 \mod p \\
y & \equiv 109757376476806880843861453881037644563847016045849499602 \mod p
\end{align*}
\]

5. Conclusion and Future Development

The implementation of the SM computation using Python language in SageMath is demonstrated for the scalar decomposition, EN blocks and the ENSM computation. The results indicated that the SageMath compiler is simpler compared to the manual calculation which is prone to mistakes and can be easily verified. The presence implementations can be conducted to other ECC standardizations with different parameters value. For instance, Brainpool of the type 384-bit prime field curve or NUMS of the type 256-bit prime field curve [20].

Acknowledgements

This work received financial support from the Ministry of Education under the grant code FRGS/1/2019/ICT05/UNISEL/03/1. We would also like to thank Universiti Selangor for providing research facilities.

References

[1] K. Rabah, “Elliptic curve cryptography over binary finite field GF(2^m),” Inf. Technol. J., vol. 5, no. 1, pp. 204–229, 2006.
[2] A. J. Zargar, M. Manzoor, and T. Mukhtar, “Encryption / decryption using elliptic curve cryptography,” Int. J. Adv. Res. Comput. Sci., vol. 8, no. 7, pp. 48-51, 2017.
[3] N. Torii and K. Yokoyama, “Elliptic curve cryptosystem,” Fujitsu Sci. Tech. J., vol. 36, no. 2, pp. 140–146, 2000.
[4] D. Mahto and D. K. Yadav, “RSA and ECC: A comparative analysis,” Int. J. Appl. Eng. Res., vol. 12, no. 19, pp. 9053–9061, 2017.
[5] T. M. Aung and N. N. Hla, “Implementation of elliptic curve arithmetic operations for prime field and binary field using java biginteger class,” Int. J. Eng. Res. Tech. Res., vol. 6, no. 8, pp. 454–459, 2017.
[6] E. Barker, “Recommendation for Key Management – Part 1: General,” NIST Spec. Publ. 800-57, vol. 4, pp. 1–142, 2016.
[7] R. K. A. Ajeena and H. Kamarulhaili, “Mathematical analysis of the computational complexity of integer sub-decomposition algorithm,” J. Phys. Conf. Ser., vol. 622, no. 012032, 2015.
[8] B. Chen, C. Hu, and C. A. Zhao, “A note on scalar multiplication using division polynomials,” IET Inf. Secur., vol. 11, no. 4, pp. 195–198, 2017.
[9] S. R. SubramanyaRao, Z. Hu, and C. A. Zhao, “Division polynomial-based elliptic curve scalar multiplication revisited,” IET Inf. Secur., vol. 13, no. 6, pp. 614–617, 2019.
[10] N. Kanayama, Y. Liu, E. Okamoto, K. Saito, T. Teruya, and S. Uchiyama, “Implementation of an elliptic curve scalar multiplication method using division polynomials,” IEICE Trans. Fundam. Electron. Commun. Comput. Sci., vol. E97-A, no. 1, pp. 300–302, 2014.
[11] “Sage,” SageMath Mathematical Software System. [Online]. Available: https://www.sagemath.org/development.html. [Accessed: Jan. 31, 2021].
[12] M. Roy, N. Deb, and A. J. Kumar, “Point generation and base point selection in ECC: An overview,” Int. J. Adv. Res. Comput. Commun. Eng., vol. 3, no. 5, pp. 6711–6713, 2014.
[13] L. K. Jie and H. Kamarulhaili, “Comparison study on point counting algorithms of elliptic curves over prime field,” Eur. J. Sci. Res., vol. 61, no. 4, pp. 538–548, 2011.
[14] H. Kamarulhaili and L. Khang, “Elliptic curve cryptography and point counting algorithms,” in Cryptography and Security in Computing, J Sen, Ed. London, UK: In Teach, 2012, pp. 91-116. Accessed on: Jan. 31, 2021. [Online], doi: 10.5772/34042.
[15] Y. Hu, Y. Y. Cui, and T. Li, “An optimization base point choice algorithm of ECC on GF(p),” ICCMS 2010 - 2010 Int. Conf. Comput. Model. Simul., vol. 4, pp. 103–105, 2010.
[16] D. Kohel, “The geometry of efficient arithmetic on elliptic curves,” AMS Contemp. Math., 637, pp. 95-110, 2015.

[17] A. J. Menezes, S. A. Vanstone, and R. J. Zuccherato, “Counting Points on Elliptic Curves Over $F_{2^m}$,” Math. Comput., vol. 60, no. 201, p. 407, 1993.

[18] N. Muslim, F. Yunos, Z. Razali, and M. R. M. Said, "Elliptic net scalar multiplication upon Koblitz curves," Malaysian Journal of Mathematical Sciences, vol. 14, no. 3, pp. 373–388, 2020.

[19] G. Locke and P. Gallagher, FIPS PUB 186-3 Digital Signature Standard (DSS), Computer Security Resource Center, June 2009. Accessed on: Jan. 31, 2021. [Online]. Available: https://csrc.nist.gov/publications/detail/fips/186/3/archive/2009-06-25

[20] “Standard curve database,” 2020. [Online]. Available: https://neuromancer.sk/std/methods/. [Accessed: Jan. 31, 2021].