Comparison of scalar multiplication algorithms in a low resource device

Mohamed Ramdani
LRDSI Laboratory, Computer Science Dept
University of Blida1, Blida, Algeria
moumouhr@yahoo.fr
med.ramdani@yahoo.com

Mohamed Benmohammed
LIRE Laboratory, Computer Science Dept
University of Constantine2, Constantine, Algeria
ben_moh123@yahoo.com

Nadjia Benblidia
LRDSI Laboratory, Computer Science Dept
University of Blida1, Blida, Algeria
benblidia@gmail.com

Abstract

Scalar multiplication of a point is the central operation in the elliptic curves cryptography (ECC). It's a complex operation that requires a lot of optimization especially on execution times to reduce resource consumption in systems with low computing power and memory such as embedded systems. Several works on the acceleration of computation and many others on the reduction of the complexity of mathematical methods used in the computations on the elliptic curves have been realized. Many algorithms for computing scalar multiplication are proposed, each using their own calculation technique and mathematical methods. In this paper, we combine the techniques of accelerated computations with optimized mathematical methods and we implement them on algorithms for scalar multiplication of a point. This work aims to distinguish the most efficient computation algorithm with the best acceleration technique. The results show that the Joye algorithm combined with the co-Z Doubling-Addition acceleration technique gives better results and saves more than one second of computation time compared to the other algorithms implemented in this paper.

Keywords – Elliptic curves cryptography (ECC), Scalar multiplication, embedded systems, acceleration computations, co-Z addition

In: Proceedings of the 3rd Edition of the International Conference on Advanced Aspects of Software Engineering (ICAASE18), Constantine, Algeria, 1,2-December-2018, published at http://ceur-ws.org

1 Introduction

Cryptography has emerged as a reliable and powerful solution for maintaining data confidentiality. To make information eligible, cryptographic mechanisms use complex mathematical methods that often involve intensive computations. This intensity is often a problem for low-resource systems like embedded systems.

There are two ways to encrypt a message; symmetric method (Secret Key Cryptography SKC) based on shared private keys, and asymmetric method (Public-Key Cryptography PKC) based on a couple of public and private keys. The advantage of the latter is that all exchanges are public and its security is based on the difficulty of finding the secret from information exchanged publicly.

One of the most recently used asymmetric cryptosystems is Elliptic Curve Cryptography (ECC). ECC uses short keys for equal security to other asymmetric cryptosystems [MKY16]. It is recommended for systems in which electronic devices have low computing power and very limited memory such as embedded systems. However, despite recent optimizations on ECC, essentially the reduction of computational complexity, this cryptosystem remains complex and requires further optimization.

The important and costly operation in Elliptic Curve Cryptography for encryption, decryption, digital signature, key exchange, etc is scalar multiplication of a point. This is a complex operation that requires more optimizations for the acceleration of cryptographic computations. The execution time of any ECC operation depends on the execution time of the scalar multiplication.

An embedded system is an autonomous system, generally dedicated to specific tasks, having a limited size and low resources. There are two important constraints of an embedded system: optimized computing power while respecting the temporal and spatial constraints, and an essential security to ensure data confidentiality

especially for sensitive applications. The system used in this work is an Arduino card very limited in memory resources and computing power. It is an open source system, open hardware and open source bootloader.

In this paper, we present an implementation of scalar multiplication algorithms on a low-resource system to compare and derive the most optimized computation method for such systems. We define the scalar multiplication of a point and present some algorithms of its computation, in section 2. Section 3 is dedicated to acceleration techniques of cryptographic computations on a scalar multiplication and to the system of coordinates used. In Section 4, we present the software implementation of acceleration techniques in scalar multiplication algorithms and present the comparison results in Section 5.

2 Scalar multiplication

The basic operation performed in protocols based on Elliptic Curve Cryptography is the scalar multiplication of a point on the curve. Each scalar multiplication requires several thousand operations in a finite field. Let k be a scalar of n bits, P and Q two points of an elliptic curve defined on a finite field F by the Weierstrass equation [Gui13][BJ02]:

$$E : Y^2 + a_1XY + a_3 = X^3 + a_2X^2 + a_4X + a_6 \quad (1)$$

where a_1, a_2, a_3, $a_4 \in$ F. In this work, we consider the finite prime fields F_p. Equation (1) then becomes:

$$E : Y^2 = X^3 + aX + b \quad (2)$$

where a, b $\in F_p$. The multiplication of the scalar k by the point P of the curve is another point Q s.t Q = k · P = P + P + . . . + P (k times).

Two operations are necessary to perform a scalar multiplication: the point doubling P + P = 2P = P' and the point addition P + P'. The computations of doubling and addition are given by formulas (3) and (4) below. Let P (X_1, Y_1) and Q (X_2, Y_2) be two points of the elliptic curve s.t P, Q ∈ E (F_p) and P ≠ Q : P + P = 2P = (X_3, Y_3) is computed as follows:

$$\begin{aligned} X_3 &= \left(\frac{3X_1^2 + a}{2Y_1}\right)^2 - 2X_1 \\ Y_3 &= \left(\frac{3X_1^2 + a}{2Y_1}\right)^2 (X_1 - X_3) - Y_1 \end{aligned} \quad (3)$$

and P + Q = (X_4, Y_4) is the result of an addition of the two points P and Q is computed as follows:

$$\begin{aligned} X_4 &= \left(\frac{Y_2 - Y_1}{X_2 - X_1}\right)^2 - X_1 - X_2 \\ Y_4 &= \left(\frac{Y_2 - Y_1}{X_2 - X_1}\right)^2 (X_1 - X_4) - Y_1 \end{aligned} \quad (4)$$

The standard algorithm called Dbl-and-add [Knu97] for calculating scalar multiplication of a point is given by algorithm 1.

Algorithm 1 Standard Dbl-and-add algorithm

Require: k = (k_{d-1},..., k_1, k_0)$_2$, P ∈ E(F_p)
Ensure: Q = k·P
  $R_0 = \emptyset$
  R_1 = P
  for $i \leftarrow$ 1 to d-1 **do**
    if k_i = 1 **then**
      $R_0 \leftarrow R_0 + R_1$
    end if
    $R_1 \leftarrow 2 \times R_1$
  end for
  return R_0;

This algorithm has a huge disadvantage for the security of the private key used because it is very vulnerable to attacks SPA [MS00] and DPA [KJJ99]. Indeed, by analyzing either the energy consumed or the number of clock cycles performed per operation, an attacker can reconstruct the private key. This fault is corrected by several works [Mon87][Joy07] [LS01][DSF16][FGDM$^+$10], we choose here to work on the algorithms of Mongomery ladder [Mon87] and the algorithm of Joye Dbl-and-add [Joy07] (see algorithms 2 and 3).

Algorithm 2 Montgomery ladder algorithm

Require: k = (k_{d-1},..., k_1, k_0)$_2$, P ∈ E(F_p)
Ensure: Q = k·P
  $R_0 = \emptyset$
  R_1 = P
  for $i \leftarrow$ d-1 downto 0 **do**
    $b \leftarrow k_i$
    $R_{1-b} \leftarrow R_{1-b} + R_b$
    $R_b \leftarrow 2 \times R_b$
  end for
  return R_0;

Algorithm 3 Joye Dbl-and-add algorithm

Require: k = (k_{d-1},..., k_1, k_0)$_2$, P ∈ E(F_p)
Ensure: Q = k·P
  $R_0 = \emptyset$
  R_1 = P
  for $i \leftarrow$ 0 to d-1 **do**
    $b \leftarrow k_i$
    $R_{1-b} \leftarrow 2 \times R_{1-b} + R_b$
  end for
  return R_0;

Table 1: Calculation of point doubling and addition operations with Jacobean coordinates

	P + Q	2P
A	$X_1Z_2^2$	$4X_1Y_1^2$
B	$X_2Z_1^2$	$3X_1^2 + AZ_1^4$
C	$Y_1Z_2^3$	/
D	$Y_2Z_1^3$	/
E	B – A	/
F	D – C	/
X_3	$-E^3 - 2AE^3 + F^2$	$-2A + B^2$
Y_3	$-CE^3 + F(AE^2 - X_3)$	$-8Y_1^4 + B(A - X_3)$
Z_3	Z_1Z_2E	$2Y_1Z_1$

3 Related work

The complexity of the computations in scalar multiplication of a point and the large number of point doubling and point addition calculated to perform this operation required a lot of optimization works reduce computations. In this section, we present some algorithms that allow computation accelerations in a scalar multiplication, but before that, we will focus first on the choice of coordinate systems used.

In the Affine coordinate system, the addition and doubling formulas involve point inversion operations in F_p which is considered very expensive on the finite fields. In order to avoid the inversion of points [KS17], we chose to work on the Jacobian coordinates where a point of the curve is represented by three coordinates (X : Y : Z) which corresponds to the Affine point $(\frac{X}{Z^2}, \frac{Y}{Z^3})$. Equation (2) then becomes:

$$E : Y^2 = X^3 + aXZ^4 + bZ^6 \tag{5}$$

The computation of the doubling P + P = 2P = (X_3 : Y_3 : Z_3) and the addition P + Q = (X_3 : Y_3 : Z_3) is given by the table 1.

An additional optimization of the addition, called co-Z addition, has been proposed by Meloni [Mel07] where he considers that two entry points share the same Z coordinate. Let P = (X_1 : Y_1 : Z) and Q = (X_2 : Y_2 : Z), the addition co-Z of P and Q (with P $\neq$ Q) is defined by P + Q = (X_3 : Y_3 : Z_3) so that :

$$\begin{aligned} X_3 &= D - (B + C) \\ Y_3 &= (Y_2 - Y_1)(B - X_3) - E \\ Z_3 &= Z(X_2 - X_1) \end{aligned} \tag{6}$$

where A = $(X_2 - X_1)^2$, B = X_1A, C = X_2A, D = $(Y_2 - Y_1)^2$ and E = Y_1(C-B)

In [Riv11], several algorithms are proposed to perform a scalar multiplication of a point in Jacobian coordinates with Standard Jacobian Formulae and co-Z addition Jacobian Formulae. In Standard Jacobian Formulae, a point doubling 2P is performed by the Jacobian Doubling algorithm, a P + Q point addition by Jacobian Addition and a Doubling-Addition operation (DA) 2P + Q by the Mixed Jacobian Affine algorithm. This last algorithm uses mixed coordinates; a point in Jacobian coordinates and another point in Affine coordinates. In co-Z Jacobian Formulae, a point doubling is computed by the Initial Doubling algorithm (XYcZ-IDBL), an addition by (X;Y)-only co-Z addition (XYcZ-ADD) and a DA by (X;Y)-only co-Z Doubling-Addition (XYcZ-DA).

4 Software implementation

The various techniques of computational acceleration for point doubling and addition are implemented on the different scalar multiplication algorithms presented above in order to compare the performance of each algorithm with each technique. The algorithms obtained and used as support for our comparisons are presented in the following:

- Standard Dbl-and-add with Jacobian doubling and Jacobian addition (see algorithm 4)
- Standard Dbl-and-add with co-Z Initial doubling and co-Z addition (see algorithm 5)
- Montgomery ladder with Jacobian doubling and Jacobian addition (see algorithm 6)
- Montgomery ladder with co-Z Initial doubling and co-Z addition (see algorithm 7)
- Joye Dbl-and-add with Mixed Jacobian Affine Doubling-Addition (DA) (see algorithm 8)
- Joye Dbl-and-add with co-Z Doubling-Addition (DA) (see algorithm 9)

Algorithm 4 Standard Dbl-and-add with Jacobian doubling and Jacobian addition

Require: k = $(k_{d-1},..., k_1, k_0)_2$, P $\in$ E(F_p)
Ensure: Q = k·P
  $R_0 = \emptyset$
  R_1 = P
  for $i \leftarrow$ 1 to d-1 **do**
    if k_i = 1 **then**
      (R_0) $\leftarrow$ Jacobian_addition(R_0,R_1)
    end if
    (R_1) $\leftarrow$ Jacobian_doubling(R_1)
  end for
  return R_0;

Algorithm 5 Standard Dbl-and-add with co-Z Initial doubling and co-Z addition

Require: k = $(k_{d-1},..., k_1, k_0)_2$, P ∈ E($F_p$)
Ensure: Q = k·P
  $R_0 = \emptyset$
  R_1 = P
  for $i \leftarrow$ 1 to d-1 **do**
    if $k_i = 1$ **then**
      (R_1,R_0) ← XYcZ-ADD(R_0,R_1)
    end if
    (R_0,R_1) ← XYcZ-IDBL(R_0)
  end for
  return R_0;

Algorithm 6 Montgomery ladder with Jacobian doubling and Jacobian addition

Require: k = $(k_{d-1},..., k_1, k_0)_2$, P ∈ E($F_p$)
Ensure: Q = k·P
  $R_0 = \emptyset$
  R_1 = P
  for $i \leftarrow$ d-1 downto 0 **do**
    $b \leftarrow k_i$
    (R_{1-b}) ← Jacobian_addition(R_{1-b},R_b)
    (R_b) ← Jacobian_doubling(R_b)
  end for
  return R_0;

Algorithm 7 Montgomery ladder with co-Z Initial doubling and co-Z addition

Require: k = $(k_{d-1},..., k_1, k_0)_2$, P ∈ E($F_p$)
Ensure: Q = k·P
  $R_0 = \emptyset$
  R_1 = P
  for $i \leftarrow$ d-1 downto 0 **do**
    $b \leftarrow k_i$
    (R_{1-b},R_b) ← XYcZ-ADD(R_b,R_{1-b})
    (R_b,R_{1-b}) ← XYcZ-IDBL(R_b)
  end for
  return R_0;

Algorithm 8 Joye Dbl-and-add with Mixed Jacobian Affine DA

Require: k = $(k_{d-1},..., k_1, k_0)_2$, P ∈ E($F_p$)
Ensure: Q = k·P
  $R_0 = \emptyset$
  R_1 = P
  for $i \leftarrow$ 0 to d-1 **do**
    $b \leftarrow k_i$
    (R_{1-b}) ← Mixed_Jacobian_Affine_DA(R_{1-b},R_b)
  end for
  return R_0;

Algorithm 9 Joye Dbl-and-add with Mixed Jacobian Affine DA

Require: k = $(k_{d-1},..., k_1, k_0)_2$, P ∈ E($F_p$)
Ensure: Q = k·P
  $R_0 = \emptyset$
  R_1 = P
  for $i \leftarrow$ 0 to d-1 **do**
    $b \leftarrow k_i$
    (R_{1-b},R_b) ← XYcZ-DA(R_{1-b},R_b)
  end for
  return R_0;

5 Results and comparison

The results presented in this comparison are the result of experiments carried out on a low-resource device with a very low computing capacity and a very limited memory size. The device used is an Arduino Uno R3 board whose characteristics, shown in Table 2, are close to most embedded devices. The curves used are those recommended in [Qu99]. The size of the keys used is 192 and 256 bits.

Table 2: Features of Arduino Uno R3

Feature	Type/Value
Chipset	Atmega328P
Clock Frequency	8/16 Mhz
Voltage	1.8 - 5.5 V
Bit Width	8 bits
Static RAM	2 kB
Program Memory	32 kB

Table 3 shows the execution time (in ms) of point doubling, point addition and doubling-addition operations in Standard Jacobian Formulae and co-Z Jacobian Formulae.

We find that the addition and doubling-addition operations in co-Z Jacobian Formulae are faster and require less computations, unlike the Initial Doubling operation where the treatments are heavier than Standard Jacobian Formulae.

The execution time of the scalar multiplication algorithms (algorithm 4 to algorithm 9) is given by table 4 and 5.

The standard Dbl-and-add algorithm gives better results, its resource consumption is very low because the number of point addition is optimized to log(d/3) unlike the other two algorithms where at each iteration, an addition is performed regardless of the value of the bit. However, its vulnerability to SPA and DPA attacks makes it almost inappropriate. In the other results, we

Table 3: Execution time (in ms) of point doubling, addition and doubling-addition operations

Formulae	**Algorithm**	**P-192**	**P-256**
Point Doubling (2P)			
Standard Jacobian	Jacobian Doubling	6.064	11.732
co-Z Jacobian	XYcZ-IDBL	9.708	18.592
Point Addition (P + Q)			
Standard Jacobian	Jacobian Addition	10.056	19.332
co-Z Jacobian	XYcZ-ADD	3.628	6.916
Doubling-Addition (2P + Q)			
Standard Jacobian	Mixed DA	15.501	29.712
co-Z Jacobian	XYcZ-DA	10.916	20.808

Table 4: Execution time of the scalar multiplication algorithms (P-192)

Algorithm	P-192	
	Standard Jacobian	co-Z Jacobian
Standard Dbl-and-add	Algorithm 4	Algorithm 5
	1841	2260
Montgomery ladder	Algorithm 6	Algorithm 7
	2960	2641
Joye Dbl-and-add	Algorithm 8	Algorithm 9
	3085	2213

Table 5: Execution time of the scalar multiplication algorithms (P-256)

Algorithm	P-256	
	Standard Jacobian	co-Z Jacobian
Standard Dbl-and-add	Algorithm 4	Algorithm 5
	4770	5724
Montgomery ladder	Algorithm 6	Algorithm 7
	7471	6630
Joye Dbl-and-add	Algorithm 8	Algorithm 9
	7815	5528

notice that the algorithm 9 gives very interesting results with a reduced execution time of 428 ms compared to the algorithm 7 for a 192-bit key and 1102 ms for a 256-bit key.

the graph below (see figure 1) gives a global comparison of all implemented algorithms.

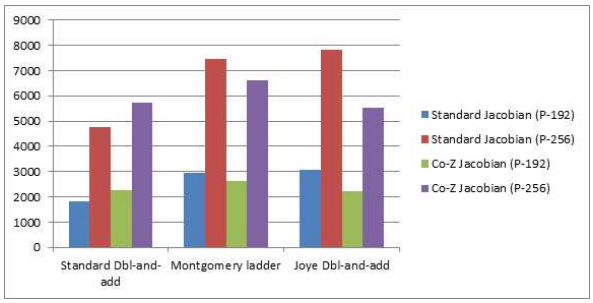

Figure 1: A global comparison of implemented algorithms

6 Conclusion

In this work, we compared the different computational acceleration techniques for scalar multiplication by a point on an embedded system in order to compare them and to distinguish the best method of computing a scalar multiplication in a low-resource system. The hardware used is an Arduino embedded system with a microcontroller limited in resources including memory and computing power. We used Jacobian coordinate system with Standard Jacobian Formulae and co-Z Jacobian Formulae, an improved version of the addition called co-Z addition, since this type of coordinates provides less complex and more efficient computation operations. We

have implemented acceleration techniques on three algorithms widely used in practice: the standard Dbl-and-add algorithm, the Montgomery ladder algorithm and the Joye Dbl-and-add algorithm. We found that the Joye Dbl-and-add algorithm widh co-Z Doubling-Addition gives better results if we take into account the level of security offered, unlike the standard algorithm Dbl-and-add which offers a better time but it is very vulnerable to SPA and DPA attacks. The time saved for a 256-bit key is more than one second (1102 ms) for scalar multiplication using the Joye Dbl-and-add algorithm coupled with the technique co-Z Doubling-Addition compared to other algorithms implemented in this work.

References

[BJ02] Eric Brier and Marc Joye. Weierstraß elliptic curves and side-channel attacks. In *International Workshop on Public Key Cryptography*, pages 335–345. Springer, 2002.

[DSF16] Giovanni Di Sirio and Giovanni Fontana. Method for protecting a cryptographic device against spa, dpa and time attacks, August 30 2016. US Patent 9,430,188.

[FGDM+10] Junfeng Fan, Xu Guo, Elke De Mulder, Patrick Schaumont, Bart Preneel, and Ingrid Verbauwhede. State-of-the-art of secure ecc implementations: a survey on known side-channel attacks and countermeasures. In *Hardware-Oriented Security and Trust (HOST), 2010 IEEE International Symposium on*, pages 76–87. IEEE, 2010.

[Gui13] Aurore Guillevic. *Arithmetic of pairings on algebraic curves for cryptography*. PhD thesis, Ecole Normale Supérieure de Paris-ENS Paris, 2013.

[Joy07] Marc Joye. Highly regular right-to-left algorithms for scalar multiplication. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 135–147. Springer, 2007.

[KJJ99] Paul Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In *Annual International Cryptology Conference*, pages 388–397. Springer, 1999.

[Knu97] Donald Ervin Knuth. *The art of computer programming: sorting and searching*, volume 3. Pearson Education, 1997.

[KS17] K Keerthi and B Surendiran. Elliptic curve cryptography for secured text encryption. In *Circuit, Power and Computing Technologies (ICCPCT), 2017 International Conference on*, pages 1–5. IEEE, 2017.

[LS01] P-Y Liardet and Nigel P Smart. Preventing spa/dpa in ecc systems using the jacobi form. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 391–401. Springer, 2001.

[Mel07] Nicolas Meloni. New point addition formulae for ecc applications. In *International Workshop on the Arithmetic of Finite Fields*, pages 189–201. Springer, 2007.

[MKY16] Dindayal Mahto, Danish Ali Khan, and Dilip Kumar Yadav. Security analysis of elliptic curve cryptography and rsa. In *Proceedings of the World Congress on Engineering WCE*, volume 1, 2016.

[Mon87] Peter L Montgomery. Speeding the pollard and elliptic curve methods of factorization. *Mathematics of computation*, 48(177):243–264, 1987.

[MS00] Rita Mayer-Sommer. Smartly analyzing the simplicity and the power of simple power analysis on smartcards. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 78–92. Springer, 2000.

[Qu99] Minghua Qu. Sec 2: Recommended elliptic curve domain parameters. *Certicom Res., Mississauga, ON, Canada, Tech. Rep. SEC2-Ver-0.6*, 1999.

[Riv11] Matthieu Rivain. Fast and regular algorithms for scalar multiplication over elliptic curves. *IACR Cryptology ePrint Archive*, 2011:338, 2011.