Fast Point Quadrupling on Elliptic Curves

Duc-Phong Le
Information Security Group
Temasek Laboratories
National University of Singapore
Singapore 119260
tslld@nus.edu.sg

Binh P. Nguyen
School of Information and Communication Technology
Hanoi University of Science and Technology
Hanoi, Vietnam
binhnp@soict.hut.edu.vn

ABSTRACT

Ciet *et al.* (2006) proposed an elegant method for trading inversions for multiplications when computing $[2]P+Q$ from two given points P and Q on elliptic curves of Weierstrass form. Motivated by their work, this paper proposes a fast algorithm for computing $[4]P$ with only one inversion in affine coordinates. Our algorithm that requires $1\mathbf{I} + 8\mathbf{S} + 8\mathbf{M}$, is faster than two repeated doublings whenever the cost of one field inversion is more expensive than the cost of four field multiplications plus four field squarings (i.e. $\mathbf{I} > 4\mathbf{M} + 4\mathbf{S}$). It saves one field multiplication and one field squaring in comparison with the Sakai-Sakurai method (2001). Even better, for special curves that allow "a = 0" (or "b = 0") speedup, we obtain $[4]P$ in affine coordinates using just $1\mathbf{I} + 5\mathbf{S} + 9\mathbf{M}$ (or $1\mathbf{I} + 5\mathbf{S} + 6\mathbf{M}$, respectively).

Categories and Subject Descriptors

F.2.1 [**Analysis of Algorithms and Problem Complexity**]: Numerical Algorithms and Problems—*Computations in finite fields*; G.4 [**Mathematics of Computing**]: Mathematical software—*Algorithm design and analysis, Efficiency*

General Terms

Algorithms

Keywords

Elliptic curve cryptography, fast arithmetic, quadrupling, affine coordinates

1. INTRODUCTION

The use of elliptic curves in cryptography was suggested independently by Miler in [8] and Koblitz in [7]. Since then, Elliptic Curve Cryptography (ECC) have received a lot of attention due to the fast group law on elliptic curves and no sub-exponential attack on the discrete logarithm problem defined over elliptic curves. Recall that the best known discrete logarithm algorithm on elliptic curves is the parallelized Pollard rho algorithm [11, 10], which has running time $O(\sqrt{r})$, where r is the prime order of the largest subgroup of points on an elliptic curve E. Meanwhile, the best algorithm for discrete logarithm computation in a finite field $\mathbb{F}_q$ is the index calculus attack [9] which has sub-exponential running time in the field size (i.e., q). Thus to achieve the same level of security in both groups, q must be significantly larger than r. For example, at 80-bit of security level, it is required that $r \geq 2^{160}$ and $q \geq 2^{1024}$ (see NIST recommendations [2]). In other words, ECC can provide the same security level as RSA (or Diffie-Hellman) but with much shorter keys. This is mainly relevant for small embedded devices.

In [4], Ciet *et al.* introduced a fast algorithm trading one inversion for several multiplications to compute $[2]P + Q$ from two given points P and Q on an elliptic curve. Their algorithm, which requires $1\mathbf{I} + 2\mathbf{S} + 9\mathbf{M}$, is faster than the Eisenträger *et al.* method [5] whenever the cost of one field inversion is more expensive than the cost of six field multiplications.

The idea of trading field inversions for field multiplications when computing $[4]P$ has been appeared in a work of Sakai and Sakurai [12], in which the authors presented a general formula for computing $[2^k]P$ in both affine and projective coordinates. In affine coordinates, their method requires one field inversion, $(4k+1)$ field multiplications and $(4k+1)$ field squarings. For $k = 2$, the cost of their algorithm is one inversion, 9 multiplications and 9 squarings.

In this paper, motivated by the Ciet *et al.*'s method, we introduce new formulas for quadrupling of points on elliptic curves over finite fields of odd characteristic $p > 3$ in affine coordinates. Our algorithm requires one field inversion, 8 field multiplications and 8 field squarings on general curves. In comparison to the Sakai-Sakurai method [12], our algorithm saves one field multiplication and one field squarings. It is also faster than two repeated doublings if the cost of one field inversion is more expensive than the cost of four field multiplications plus four field squarings. Even better, for curves that are sensibly chosen with small parameters so that multiplications by those parameters have negligible cost, $[4]P$ can be computed by using just only $1\mathbf{I} + 7\mathbf{S} + 7\mathbf{M}$. Moreover, for curves that allow "a = 0" speedup which can be found in many pairing-friendly elliptic curves, we obtain $[4]P$ in affine coordinates by using just $1\mathbf{I} + 5\mathbf{S} + 6\mathbf{M}$.

The rest of the paper is organized as follows. Definitions of elliptic curve cryptography are briefly recalled in Section 2. Section 3 presents our algorithms. We also give some

SoICT 2012, August 23-24, 2012, Ha Long, Vietnam.

analysis in this section. Section 4 concludes the paper.

2. PRELIMINARIES

In this section, we first recall some basic definitions in elliptic curve cryptography. For much more material on elliptic curve cryptography we refer to [6]. Then we review the algorithm of Ciet al et. [4] for computing $[2]P + Q$ from two given points P and Q on an elliptic curve E.

2.1 Elliptic curves over finite fields

For a prime p with $p > 3$, let $\mathbb{F}_q = \mathbb{F}_{p^m}$ $(m \geq 1)$ be a finite field of characteristic p having q elements. An elliptic curve E defined over the finite field $\mathbb{F}_q$ (denoted $E/\mathbb{F}_q$) in short Weierstrass form is the set of solutions (x, y) to the following equation:

$$E : y^2 = x^3 + ax + b, \tag{1}$$

together with an extra point $\mathcal{O}$ which is called the *point at infinity*, where $a, b \in \mathbb{F}_q$ such that the *discriminant* $\Delta = -16(4a^3 + 27b^2)$ is non-zero.

We usually use the notation $E(\mathbb{F}_q)$ for the set of points (x, y) with coordinates in the field $\mathbb{F}_q$ together with the point $\mathcal{O}$, the identity element of the group. Points on an elliptic curve can be represented in several coordinate systems, such as affine coordinates, and projective coordinates.

We also use the notation $\#E(\mathbb{F}_q)$ for the number of points on the elliptic curve E. The curve $E/\mathbb{F}_q$ can have at most $2q + 1$ points, *i.e.* the point at infinity $\mathcal{O}$ along with $2q$ pairs (x, y) with $x, y \in \mathbb{F}_q$, satisfying the equation 1. This is because, for each $x \in \mathbb{F}_q$, there are at most two possible values of $y \in \mathbb{F}_q$, satisfying 1. But in reality, not all elements of $\mathbb{F}_q$ has a square root. Approximatively, only half of the elements in $\mathbb{F}_q = \mathbb{F}_q \backslash \{0\}$ have square roots. Therefore the expected number of points on $E/\mathbb{F}_q$ is about $q+1$. If we set

$$\#E(\mathbb{F}_q) = q + 1 - t,$$

where the value t is called the trace of Frobenius at q. The following well-known theorem of Hasse gives a tight bound of the number of points on an elliptic curve over a finite field.

THEOREM 1 (HASSE, 1933). *The trace of Frobenius satisfies*

$$|t| \leq 2\sqrt{q}.$$

The Group Law

The set of points on an elliptic curve forms a group under a certain addition rule. Let $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$ be two points on elliptic curve E. The negative of the point P_1 is given $-P_1 = (x_1, -y_1)$. Assume that $P_1 \neq -P_2$. Then the coordinates of $P_3 = P_1 + P_2 = (x_3, y_3)$ can be computed as follows:

$$x_3 = \lambda^2 - x_1 - x_2, \tag{2a}$$

$$y_3 = \lambda(x_1 - x_3) - y_1, \tag{2b}$$

where

$$\lambda = \begin{cases} \frac{3x_1^2 + a}{2y_1} & \text{if } P_1 = P_2, \\ \frac{y_2 - y_1}{x_2 - x_1} & \text{otherwise.} \end{cases} \tag{3}$$

The doubling and addition costs are $1\mathbf{I} + 2\mathbf{M} + 2\mathbf{S}$ and $1\mathbf{I} + 2\mathbf{M} + 1\mathbf{S}$, respectively, in affine coordinates, where $\mathbf{I}$, $\mathbf{M}$ and $\mathbf{S}$ denote field inversion, field multiplication and field squaring, respectively. For every $n \in \mathbb{N}$, let $[n]$ denote scalar multiplication by n, *i.e.,* $[n] : P \mapsto [n]P = \underbrace{P + \cdots + P}_{\text{n terms}}$.

Graphically, this addition law can be interpreted as in the Figure 1. The straight line passing P_1 and P_2 will have the third intersection with the curve at the point $-P_3$, where $-P_3 = (-x_3, y_3)$.

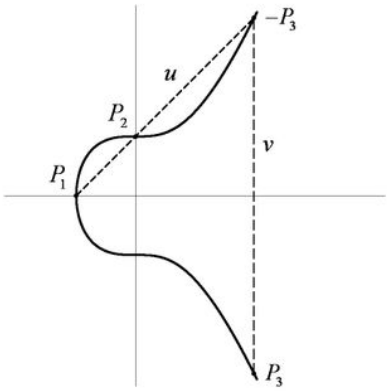

Figure 1: Addition Law of Points on Elliptic Curves

2.2 Ciet et al. Algorithm

In [4], the authors gave formulas to directly compute $[2]P + Q$ from given points P and Q on elliptic curves with one field inversion, 2 field squarings and 9 field multiplications. The key idea is to perform two additions $(P + (P + Q))$ instead of one doubling and then one addition. Given two points $P = (x_1, y_1)$, $Q = (x_2, y_2)$, the sum $R = (P + (P + Q))$ is obtained by:

$$\lambda_1 = \frac{y_2 - y_1}{x_2 - x_1}, \quad x_3 = \lambda_1^2 - x_1 - x_2, \quad y_3 = \lambda_1(x_1 - x_3) - y_1,$$

$$\lambda_2 = \frac{y_3 - y_1}{x_3 - x_1}, \quad x_4 = \lambda_2^2 - x_1 - x_3, \quad y_4 = \lambda_2(x_1 - x_4) - y_1.$$

Eisenträger *et al.* observed that the y-coordinate of $(P + Q)$ can be omitted and thus one field multiplication is saved (see [5]). From this observation, Ciet *et al.* [4] went further and showed that the x-coordinate of the point $(P + Q)$ can also be omitted and two divisions from calculation of λ_1 and λ_2 can obtained by one inversion. Their algorithm is described as in Table 1.

3. THE ALGORITHM

Our algorithm performs a quadrupling $[4]P$ on an elliptic curve E in affine coordinates using only one field inversion, 8 field multiplications, and 8 field squarings.

3.1 Description of the Algorithm

Let $P = (x_1, y_1)$, we need to compute $[4]P = (x_4, y_4)$. Let

$$d = (3x^2 + a)(12x_1y_1 - 3(x^2 + a)^2) - 8y_1^4. \tag{4}$$

It is easy to see that $d = 8y_1y_3$, where y_3 is y-coordinate of the point $[2]P$. The computation of the value d requires $1\mathbf{M}$

Table 1: $([2]P+Q)$ algorithm, for Weierstrass elliptic curves over a field $\mathbb{F}_q$

Input: $P=(x_1,y_1)\neq O,\ Q=(x_2,y_2)\neq\mathcal{O}$ **Output**: $T=[2]P+Q=(x_4,y_4)$	
if $(x_1=x_2)$ **then**	
if $(y_1=y_2)$ **then return** $[3]P$ **else return** P;	
$A\leftarrow(x_2-x_1)^2$;	
$B\leftarrow(y_2-y_1)^2$;	
$d\leftarrow A(2x_1+x_2)-B$;	2**S**
if $(d=0)$ **then return** $\mathcal{O}$;	
$D\leftarrow(x_2-x_1)d$;	1**M**
$I\leftarrow D^{-1}$;	1**I**
$\lambda_1\leftarrow dI(y_2-y_1)$;	2**M**
$\lambda_2\leftarrow 2y_1A(x_2-x_1)I-\lambda_1$;	3**M**
$x_4\leftarrow(\lambda_2-\lambda_1)(\lambda_2+\lambda_1)+x_2$;	1**M**
$y_4\leftarrow\lambda_2(x_1-x_4)-y_1$;	1**M**
return (x_4,y_4);	
Total:	1**I** + 2**S** + 9**M**

+ 5**S**. If a and b are small, then d can be computed by 4**S** as follows (we assume that a^2, a^3 and $27b^2$ are precomputed and cached):

$$d=y_1^4+3ax_1^4-6a^2x_1^2+18by_1^2-24abx_1-a^3-27b^2. \quad (5)$$

Defining $D=(2y_1)d$ and $I=D^{-1}$, we have:

$$\tfrac{1}{2y_1}=dI, \quad \tfrac{1}{2y_3}=8y_1^4I.$$

The algorithm works as in Table 2. The total cost is 1**I** + 8**S** + 8**M**. If a and b are small, our algorithm can perform a quadrupling of point using just 1**I** + 7**S** + 7**M**.

Quadrupling on curves with $b=0$: In the case of $b=0$, d should be set to

$$d=(x_2-a)((x_2+a)^2+4ax^2). \quad (6)$$

That is because, in this setting, $y_3=(x_1^2-a)((x_1^2+a)^2+4ax_1^2)/8y_1^3$. Thus, we only need 2**M** + 2**S** to compute d. By performing similarly as in Table 2, the quadrupling computation of a point on an elliptic curve requires only 1**I** + 5**S** + 9**M**.

Quadrupling on curves with $a=0$: In the case of $a=0$, the slopes $\lambda_1=3x_2/2y_1$ and $\lambda_2=3x_3/2y_3$ are particularly simple. The value d in the Equation (4) can be replaced by

$$d=y_1^4+18by_1^2-27b^2, \quad (7)$$

Assume that $27b^2$ is precomputed and cached, the equation (7) can be computed using just two squarings. In comparison with the equation (4), we save one multiplication and 3 squarings.

In this setting, $x_3=\lambda_1^2-2x_1=x_1(y_1^2-9b)/(4y_1^2)$, which will be used for computing λ_2. We have:

$$\lambda_2=\frac{3x_3^2}{2y_3}=\frac{3x_1^2(y_1^2-9b)^2}{32y_1^4y_3} \quad (8)$$

$$=3x_1^2\frac{(y_1^4-18by_1^2+81b^2)}{2D} \quad (9)$$

$$=\frac{3}{2}(x_1^2(y_1^4-18by_1^2+81b^2)I), \quad (10)$$

Table 2: Quadrupling algorithm of a point on curves of form $y^2=x^3+ax+b$, where $ab\neq 0$

Input: $P=(x_1,y_1)\neq\mathcal{O}$ **Output**: $T=[4]P=(x_4,y_4)$	
if $(y_1=O)$ **then return** P;	
$A\leftarrow x_1^2$; $B\leftarrow 3x_1^2+a$;	1**S**
$C\leftarrow 2y_1^2$; $E\leftarrow C^2$;	2**S**
$F\leftarrow(x_1+C)^2-A-E$;	1**S**
$d\leftarrow B(3F-B^2)-2E$;	1**S** + 1**M**
if $(d=0)$ **then return** $\mathcal{O}$;	
$D\leftarrow(2y_1)d$;	1**M**
$I\leftarrow D^{-1}$;	1**I**
$\lambda_1\leftarrow dIB$;	2**M**
$x_3\leftarrow\lambda_1^2-2x_1$;	1**S**
$y_3\leftarrow\lambda_1(x_1-x_3)-y_1$;	1**M**
$H\leftarrow 3x_3^2+a$;	1**S**
$\lambda_2\leftarrow 2EIH$;	2**M**
$x_4\leftarrow\lambda_2^2-2x_3$;	1**S**
$y_4\leftarrow\lambda_2(x_3-x_4)-y_3$;	1**M**
return (x_4,y_4);	
Total:	1**I** + 8**S** + 8**M**

where $D=(2y_1)d=16y_1^4y_3$ and $I=D^{-1}$.

Table 3 performs quadrupling of a point in 1**I** + 5**S** + 6**M**.

3.2 Analyze

From the operation count we see that the algorithm is faster than two repeated doublings if one field inversion is more expensive than 4**M** + 4**S** on general curves. In comparison with the Sakai-Sakurai [12] algorithm, our method saves one field multiplication and one field squaring.

The advantage of a respective method depends on **I**/**M** ratios and **S**/**M**-ratios over prime fields. In this analysis, the ratio of a field squaring to a field multiplication is set to be **S** = 0.8**M** as commonly used in the literature, see [3]. The **S**/**M**-ratios deeply depend on many factors such as the implementations, hardware architecture, the prime characteristic of finite fields, the size of finite fields, etc. For example, the inversion-to-multiplication (**S**/**M**) ratio is bigger than 100 on smart cards (see [13]). On workstations, for NIST-recommended elliptic curves over prime fields chosen to either be a Mersenne prime, or a Mersenne-like prime for fast modular reduction and multiplication, this ratio is roughly 80 (see benchmarks reported in [3]). In the cases when the Mersenne prime cannot used (e.g. pairing-based cryptography), the **S**/**M**-ratio is often reported to be 13 on 32-bits Intel processors (see benchmarks reported in [1]).

In this setting, our algorithm is better than two repeated doubling in the case 1**I** > 7.2**M**.

On special elliptic curves, our algorithms are even better. For curves that are sensibly chosen with small parameters so that multiplications by those parameters have negligible cost, we compute $[4]P$ using just only 1**I** + 7**S** + 7**M**, thus it is faster than two repeated doublings if 1**I** > 5.4**M**. For curves with $b=0$, our algorithm requires only 1**I** + 5**S** + 9**M**, it is faster than two repeated doublings if 1**I** > 5.8**M**. Even better, for curves with $a=0$ (this form of curves can be found in pairing-friendly elliptic curves admitting a twist of degree 2, 3, and 6), our algorithm requiring only 1**I** + 5**S**

Table 3: Quadrupling algorithm of a point on curves of form $y^2 = x^3 + b$

Input: $P = (x_1, y_1) \neq \mathcal{O}$	
Output: $T = [4]P = (x_4, y_4)$	
if $(y_1 = O)$ **then return** P;	
$A \leftarrow x_1^2$; $B \leftarrow A^2$;	$2\mathbf{S}$
$C \leftarrow 3x_1^2$; $d \leftarrow B + 18bA - 27b^2$;	$1\mathbf{S}$
if $(d = 0)$ **then return** $\mathcal{O}$;	
$D \leftarrow (2y_1)d$;	$1\mathbf{M}$
$I \leftarrow D^{-1}$;	$1\mathbf{I}$
$I' \leftarrow CI$;	$1\mathbf{M}$
$\lambda_1 \leftarrow I'd$;	$1\mathbf{M}$
$\lambda_2 \leftarrow I'(y_1^4 - 18by_1^2 + 81b^2)/2$;	$1\mathbf{M}$
$x_3 \leftarrow \lambda_1^2 - 2x_1$;	$1\mathbf{S}$
$y_3 \leftarrow \lambda_1(x_1 - x_3) - y_1$;	$1\mathbf{M}$
$x_4 \leftarrow \lambda_2^2 - 2x_3$;	$1\mathbf{S}$
$y_4 \leftarrow \lambda_2(x_3 - x_4) - y_3$;	$1\mathbf{M}$
return (x_4, y_4);	
Total:	$1\mathbf{I} + 5\mathbf{S} + 6\mathbf{M}$

$+ 6\mathbf{M}$ is faster than two repeated doublings if $1\mathbf{I} > 2.8\mathbf{M}$. Let us recall that in pairing-friendly cryptography, elliptic curves that have twists of degree $d = 2$, 3, and 6 are defined by the equation $y^2 = x^3 + b$.

Table 4 sumarizes the computational complexity and make a comparison between our method and other methods when computing a point quadrupling on elliptic curves of Weierstrass form.

Table 4: Comparison of point quadrupling on Weierstrass curves $y^2 = x^3 + ax + b$ defined over $\mathbb{F}_q$

Curves	Traditional method	Sakai-Sakurai method	Our method
$a, b \in \mathbb{F}_q$	$2\mathbf{I} + 4\mathbf{S} + 4\mathbf{M}$	$1\mathbf{I} + 9\mathbf{S} + 9\mathbf{M}$	$1\mathbf{I} + 8\mathbf{S} + 8\mathbf{M}$
a, b small	$2\mathbf{I} + 4\mathbf{S} + 4\mathbf{M}$	$1\mathbf{I} + 9\mathbf{S} + 8\mathbf{M}$	$1\mathbf{I} + 7\mathbf{S} + 7\mathbf{M}$
$b = 0$	$2\mathbf{I} + 4\mathbf{S} + 4\mathbf{M}$	$1\mathbf{I} + 9\mathbf{S} + 9\mathbf{M}$	$1\mathbf{I} + 5\mathbf{S} + 9\mathbf{M}$
$a = 0$	$2\mathbf{I} + 4\mathbf{S} + 4\mathbf{M}$	$1\mathbf{I} + 9\mathbf{S} + 8\mathbf{M}$	$1\mathbf{I} + 5\mathbf{S} + 6\mathbf{M}$

4. CONCLUSION

In this paper, we presented the fast algorithms for quadrupling of point on three forms of elliptic curves that offer a better performance than a repeated doubling if $1\mathbf{I} > 7.2\mathbf{M}$, $1\mathbf{I} > 5.8\mathbf{M}$, and $1\mathbf{I} > 2.8\mathbf{M}$, respectively. This can be helpful to speedup the scalar multiplication in elliptic curve cryptography as indicated in [12, 4]. Our quadrupling algorithm can be used to speed up pairing computation over elliptic curves of form $y^2 = x^3 + b$ in affine coordinates.

Acknowledgement

The authors thank anonymous referees for useful comments and suggestions.

5. REFERENCES

[1] T. Acar, K. Lauter, M. Naehrig, and D. Shumow. Affine pairings on ARM. Cryptology ePrint Archive, Report 2011/243, 2011. http://http://eprint.iacr.org/2011/243/.

[2] E. Barker, W. Barker, W. Burr, W. Polk, and M. Smid. NIST 800-57, Recommendation for Key Management, May 2011.

[3] M. Brown, D. Hankerson, J. López, and A. Menezes. Software implementation of the nist elliptic curves over prime fields. In *Proceedings of the conference on Topics in Cryptology: The Cryptographer's Track at RSA (RSA-CT 2001)*, pages 250–265. Springer-Verlag, 2001.

[4] M. Ciet, M. Joye, K. Lauter, and P. L. Montgomery. Trading inversions for multiplications in elliptic curve cryptography. *Designs, Codes and Cryptography*, 9(2):189–206, 2006.

[5] K. Eisenträger, K. Lauter, and P. L. Montgomery. Fast elliptic curve arithmetic and improved weil pairing evaluation. In *Proceedings of the 2003 RSA conference on the Cryptographers' Track (RSA-CT 2003)*, volume 2612 of *Lecture Notes in Computer Science*, pages 243–354. Springer Berlin / Heidelberg, 2003.

[6] D. Hankerson, A. Menezes, and S. Vanstone. *Guide to Elliptic Curve Cryptography*. Springer-Verlag New York, Inc., 2003.

[7] N. Koblitz. Elliptic curve cryptosystems. *Mathematics of Computation*, 48(177):203–209, 1987.

[8] V. S. Miller. Use of elliptic curves in cryptography. In *Lecture Notes in Computer Sciences; 218 on Advances in Cryptology – CRYPTO 85*, pages 417–426, New York, NY, USA, 1986. Springer-Verlag New York, Inc.

[9] A. Odlyzko. Discrete logarithms in finite fields and their cryptographic significance. In *Proc. of the EUROCRYPT 84 Workshop on Advances in Cryptology: Theory and Application of Cryptographic Techniques*, pages 224–314, New York, NY, USA, 1985. Springer-Verlag New York, Inc.

[10] P. C. V. Oorschot and M. J. Wiener. Parallel collision search with cryptanalytic applications. *Journal of Cryptology*, 12:1–28, 1996.

[11] J. M. Pollard. Monte Carlo Methods for Index Computation (mod p). *Mathematics of Computation*, 32(143):918–924, Jul 1978.

[12] Y. Sakai and K. Sakurai. Efficient scalar multiplications on elliptic curves with direct computations of several doublings. *IEICE Transactions on Fundamentals*, E84-A(1):120–129, 2001.

[13] M. Seysen. Using an RSA accelerator for modular inversion. In *Proceedings of the 7th International Conference on Cryptographic Hardware and Embedded Systems (CHES'05)*, volume 3659 of *Lecture Notes in Computer Science*, pages 226–236, Berlin, Heidelberg, 2005. Springer-Verlag.

APPENDIX

A. PSEUDO-CODE

In this section, we present the pseudo-code for the quadrupling algorithm described in Section 3. Let (x_1, y_1) be a point on a curve of short Weierstrass form defined over the finite field $\mathbb{F}_q$. The following algorithm updates (x_1, y_1) with $[4](x_1, y_1)$. (Field) registers are denoted by T_i.

Input: $P = (x_1, y_1) \neq \mathcal{O}$, $T_1 \leftarrow x_1$, $T_2 \leftarrow y_1$, a, b
Output: $T = [4]P = (x_4, y_4)$

if $(T_2 = O)$ **then return** P;

1.	$T_3 = T_1^2$;	$\{x_1^2\}$
2.	$T_4 = 3T_3$;	$\{3x_1^2\}$
3.	$T_4 = T_4 + a$;	$\{3x_1^2 + a\}$
4.	$T_5 = T_2^2$;	$\{y_1^2\}$
5.	$T_5 = 2T_5$;	$\{2y_1^2\}$
6.	$T_6 = T_5^2$;	$\{4y_1^4\}$
7.	$T_7 = T_1 + T_5$;	$\{x_1 + 2y_1^2\}$
8.	$T_7 = T_7^2$;	$\{(x_1 + 2y_1^2)^2\}$
9.	$T_7 = T_7 - T_3$;	$\{(x_1 + 2y_1^2)^2 - x_1^2\}$
10.	$T_7 = T_7 - T_6$;	$\{(x_1 + 2y_1^2)^2 - x_1^2 - 4y_1^4\}$
11.	$T_7 = 3T_7$;	$\{12x_1y_1^2\}$
12.	$T_3 = T_4^2$;	$\{(3x_1^2 + a)^2\}$
13.	$T_3 = T_7 - T_3$;	$\{12x_1y_1^2 - (3x_1^2 + a)^2\}$
14.	$T_3 = T_4 \times T_3$;	$\{(3x_1^2 + a)$ $(12x_1y_1^2 - (3x_1^2 + a)^2)\}$
15.	$T_6 = 2T_6$;	$\{8y_1^4\}$
16.	$T_3 = T_3 - T_6$;	$\{d = (3x_1^2 + a)(12x_1y_1^2 -$ $-(3x_1^2 + a)^2) - 8y_1^4\}$

if $(T_3 = 0)$ **then return** $\mathcal{O}$;

17.	$T_7 = 2T_2$;	$\{2y_1\}$
18.	$T_7 = T_7 \times T_3$;	$\{D = 2y_1d\}$
19.	$T_7 = 1/T_7$;	$\{I = D^{-1}\}$
20.	$T_5 = T_3 \times T_7$;	$\{dI\}$
21.	$T_5 = T_5 \times T_4$;	$\{\lambda_1 = dI(3x_1^2 + a)\}$
22.	$T_4 = T_5^2$;	$\{\lambda_1^2\}$
23.	$T_4 = T_4 - T_1$;	$\{\lambda_1^2 - x_1\}$
24.	$T_4 = T_4 - T_1$;	$\{x_3 = \lambda_1^2 - 2x_1\}$
25.	$T_1 = T_1 - T_4$;	$\{x_1 - x_3\}$
26.	$T_1 = T_1 \times T_5$;	$\{\lambda_1(x_1 - x_3)\}$
27.	$T_1 = T_1 - T_2$;	$\{y_3 = \lambda_1(x_1 - x_3) - y_1\}$
28.	$T_2 = T_4^2$;	$\{x_3^2\}$
29.	$T_2 = 3T_2$;	$\{3x_3^2\}$
30.	$T_2 = T_2 + a$;	$\{3x_3^2 + a\}$
31.	$T_2 = T_2 \times T_6$;	$\{8y_1^4(3x_3^2 + a)\}$
32.	$T_2 = T_2 \times T_7$;	$\{\lambda_2 = 8y_1^4(3x_3^2 + a)I\}$
33.	$T_3 = T_2^2$;	$\{\lambda_2^2\}$
34.	$T_3 = T_3 - T_4$;	$\{\lambda_2^2 - x_3\}$
35.	$T_3 = T_3 - T_4$;	$\{x_4 = \lambda_2^2 - 2x_3\}$
36.	$T_5 = T_4 - T_3$;	$\{x_3 - x_4\}$
37.	$T_5 = T_5 \times T_2$;	$\{\lambda_2(x_3 - x_4)\}$
38.	$T_5 = T_5 - T_1$;	$\{y_4 = \lambda_2(x_3 - x_4) - y_3\}$

return (T_3, T_5);