An x-Coordinate Point Compression Method for Elliptic Curves over $\mathbf{F}_p$

Alina Dudeanu, George-Răzvan Oancea, Sorin Iftene
Department of Computer Science
"Al. I. Cuza" University
Iasi, Romania
Email: {alina.dudeanu,george.oancea,siftene}@info.uaic.ro

***Abstract*—In this paper we propose an x-coordinate point compression method for elliptic curves over $\mathbf{F}_p$, where $p > 3$ is prime, as an alternative to the classical y-coordinate point compression method. A point $P = (x, y)$ will be compressed as $\tilde{P} = (\tilde{x}, y)$ where $\tilde{x}$ has only two bits and, thus, our method allows more compact representations when $\lceil \log_2 x \rceil > \lceil \log_2 y \rceil + 1$. Both our compression and decompression algorithms involve solving cubic equations or, in some cases, only computing cube roots modulo a prime, thus being of worst-case complexity $\mathcal{O}((\log_2 p)^4)$. For some particular cases, our compression algorithm can be significantly improved, requiring only two multiplications (thus, being of worst-case complexity $\mathcal{O}((\log_2 p)^2)$).**

***Index Terms*—elliptic curves, compact representation, cube roots, cubic equations.**

I. Introduction

Elliptic curve cryptography (*ECC*) has been introduced independently by Koblitz [1] and Miller [2]. According to Menezes [3], "elliptic curve cryptosystems offer the most security per bit of any known public-key scheme", and, thus, the elliptic curve cryptographic schemes represent a suitable solution for securing resource-constrained devices as smart cards, personal digital assistants (PDAs) or mobile phones.

Elliptic curve cryptographic protocols require storing and transmitting elliptic curve points. Usually, a point P on an elliptic curve over a finite field $\mathbf{F}_q$, is represented by its affine coordinates (x_P, y_P), thus requiring at most $2\lceil \log_2 q \rceil$ bits for its representation. In order to reduce the required memory/bandwidth, several point compression (and decompression) techniques have been proposed in the literature (the most important being [4], [5], [6], [7], [8]), the majority of them for elliptic curves over fields of characteristic two. For elliptic curves over $\mathbf{F}_p$, where $p > 3$ is a prime, the y-coordinate point compression ([4]) maps (x_P, y_P) to $(x_P, \overline{y}_P)$, where $\overline{y}_P \in \{0, 1\}$, thus requiring at most $1 + \lceil \log_2 p \rceil$ bits. The classical decompression algorithm requires the computation of one square root of a certain field element.

It is possible that the size of the x-coordinate is significantly greater than the corresponding y-coordinate. In these cases, an x-coordinate point compression may be a more suitable alternative. In this paper we propose an x-coordinate point compression method for elliptic curves over $\mathbf{F}_p$, where $p > 3$ is a prime, in which the size of the compressed point is at most $2 + \lceil \log_2 p \rceil$ bits. More exactly, a point (x_P, y_P) will be mapped into to $(\tilde{x}_P, y_P)$, where $\tilde{x}_P \in \{0, 1, 2\}$. On our best knowledge, no such compression technique has been mentioned in the literature. Both our compression and decompression involve solving cubic equations or, in some cases, only computing cube roots modulo a prime. Our compression and decompression algorithm have the worst-case complexity $\mathcal{O}((\log_2 p)^4)$. However, for some particular cases, our compression algorithm can be significantly improved ($\mathcal{O}((\log_2 p)^2)$).

The paper is organized as follows. In Section 2 we present a short background on elliptic curves over $\mathbf{F}_p$, where $p > 3$ is a prime, and the classical y-coordinate point compression method. In the next section, after presenting the required background on computational number theory (computing cube roots and solving cubic equations modulo a prime), we will present our x-coordinate point compression and decompression methods. Section 4 presents the results of our experiments and the comparisons between our dedicated algorithms and the algorithms used in *NTL* library [9]. The last section concludes the paper.

II. Elliptic Curves over $\mathbf{F}_p$

In this section we will present some basic facts on elliptic curves and their applications in cryptography (for more details on this topic, the reader is referred to [10]), focusing then on the compactness of the representation of the points on elliptic curves over $\mathbf{F}_p$.

Definition 1: ([10, Definition 3.1]) Let $\mathbf{F}_q$ be an arbitrary finite field. An *elliptic curve* over $\mathbf{F}_q$ is defined by its Weierstrass equation:

$$\mathbf{E}: \quad y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6,$$

where $a_1, a_2, a_3, a_4, a_6 \in \mathbf{F}_q$ and $\Delta \neq 0$, where Δ is the *discriminant* of $\mathbf{E}$ and is defined as:

$$\begin{aligned} \Delta &= -d_1^2 d_8 - 8d_4^3 - 27d_6^2 + 9d_2d_4d_6, \text{ where} \\ d_2 &= a_1^2 + 4a_2, \\ d_4 &= 2a_4 + a_1a_3, \\ d_6 &= a_3^2 + 4a_6, \\ d_8 &= a_1^2a_6 + 4a_2a_6 - a_1a_3a_4 + a_2a_3^2 - a_4^2. \end{aligned}$$

The set of the *points* on $\mathbf{E}$, denoted as $\mathbf{E}(\mathbf{F}_q)$ (or directly as $\mathbf{E}$ if the underlying field is clear from the context), is defined

978-0-7695-4324-6/10 $26.00 © 2010 IEEE
DOI 10.1109/SYNASC.2010.23

as

$$\{(x,y) \in \mathbf{F}_q^2 | x^3+a_2x^2+a_4x+a_6-y^2-a_1xy-a_3y = 0\} \cup \{O\},$$

where O is the *point at infinity*.

If p is a prime, $p > 3$, the above equation considered over $\mathbf{F}_p$ can be considerably simplified, using an admissible change of variables, into

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

where $a, b \in \mathbf{F}_p$ satisfy $4a^3 + 27b^2 \neq 0$.

Using the *chord-and-tangent rule* for adding two points, the set of the points on $\mathbf{E}$ defined over $\mathbf{F}_p$ can be structured as a group as follows:

- The identity is O;
- If $P = (x_P, y_P)$ then $-P = (x_P, -y_P)$;
- If $P = (x_P, y_P)$, $Q = (x_Q, y_Q)$, $P \neq \pm Q$, then $P+Q = R$, $R = (x_R, y_R)$, where:

$$x_R = (\frac{y_Q - y_P}{x_Q - x_P})^2 - x_P - x_Q,$$

$$y_R = (\frac{y_Q - y_P}{x_Q - x_P})(x_P - x_R) - y_P;$$

- If $P = (x_P, y_P)$, $P \neq -P$, then $P + P = Q$, $Q = (x_Q, y_Q)$, where

$$x_Q = (\frac{3x_P^2 + a}{2y_P})^2 - 2x_P,$$

$$y_Q = (\frac{3x_P^2 + a}{2y_P})(x_P - x_Q) - y_P.$$

The security of the majority of the cryptographic schemes based on elliptic curves relies on the difficulty of the *discrete logarithm problem*, i.e., finding a positive integer n such that $nP = Q$, where P and Q are given points on an elliptic curve over a certain field $\mathbf{F}_q$. According to [3], "elliptic curve cryptosystems offer the most security per bit of any known public-key scheme" - for example, 160 bit ECCs are believed to provide about the same level of security as 1024 bit RSA (see also [11]).

A. y-Coordinate Point Compression Method

Usually, a point P on an elliptic curve over the finite field $\mathbf{F}_q$, is represented by its affine coordinates (x_P, y_P), thus requiring at most $2\lceil \log_2 q \rceil$ bits for its representation. For elliptic curves over $\mathbf{F}_p$, where $p > 3$ is a prime, the y-coordinate point compression ([4]) maps (x_P, y_P) to $(x_P, \overline{y}_P)$, where $\overline{y}_P \in \{0, 1\}$, thus requiring at most $1 + \lceil \log_2 p \rceil$ bits. The y-coordinate point compression and decompression are presented in Figure 1, and, respectively, in Figure 2.

```
y-Coordinate Compression(a, b, x_P, y_P)
   input:   a point P = (x_P, y_P) on the elliptic curve
            E : y^2 = x^3 + ax + b;
   output:  the compressed point P̄ = (x_P, ȳ_P), ;
   begin
1.    ȳ_P := y_P mod 2;
2.    return (x_P, ȳ_P)
   end.
```

Fig. 1: y-Coordinate Compression

```
y-Coordinate Decompression(a, b, x_P, ȳ_P)
   input:   a compressed point P̄ = (x_P, ȳ_P);
   output:  the corresponding point P = (x_P, y_P) on
            the elliptic curve E : y^2 = x^3 + ax + b;
   begin
1.    temp := x_P^3 + ax_P + b;
2.    y_P :=Square Root(temp);
3.    if (y_P mod 2 ≠ ȳ_P) then y_P := −y_P;
4.    return (x_P, y_P)
   end.
```

Fig. 2: y-Coordinate Decompression

Remark 1: The correctness of the above compression/decompression algorithms is based on the fact that if y_P is a square root modulo p of $x_P^3 + ax_P + b$ then $p - y_P$ is the other square root of the same element. Moreover, because p is odd, $y_P \bmod 2 \neq (p - y_P) \bmod 2$, and, thus, the remainder modulo 2 uniquely identifies the correct square root.

Decompression requires the computation of one square root of a certain field element. The algorithms for computing square roots have worst-case complexity $\mathcal{O}((\log_2 p)^4)$ (see SectionIII-A1) and, thus, the algorithm y-Coordinate Decompression has the same complexity. The complexity of y-Coordinate Compression is $\mathcal{O}(1)$.

III. x-Coordinate Point Compression Method

In this section we propose an x-coordinate point compression method for elliptic curves over $\mathbf{F}_p$, where $p > 3$ is a prime, as an alternative to the classical y-coordinate point compression method. Both compression and decompression involve solving cubic equations or, in some cases, only computing cube roots modulo a prime. We present first the required background on computational number theory.

A. Cube Roots and Cubic Equations Modulo a Prime

1) Square/Cube roots : Will present next some basic facts on quadratic/cubic residues and computing square/cube roots.

Let p be a prime and $a \in \mathbf{F}_p$. We say that a is a *quadratic (cubic) residue (modulo p)* if there exists $b \in \mathbf{F}_p$ with the property $a = b^2$ ($a = b^3$). Otherwise, a is a *quadratic (cubic) non-residue (modulo p)*.

For an element $a \in \mathbf{F}_p^*$, the *Legendre symbol* of a modulo p, denoted as $\left(\frac{a}{p}\right)$, is defined to be equal to 1 if a is a quadratic residue modulo p, and -1, otherwise.

Euler's criterion states that, for any prime p and $a \in \mathbf{F}_p^*$, the following relation holds true:

$$a^{\frac{p-1}{2}} = \left(\frac{a}{p}\right).$$

Euler's criterion provides a method for computing the Legendre symbol of a modulo p using an exponentiation modulo p, whose complexity is $\mathcal{O}((log_2(p))^3)$.

If $b^2 = a$ ($b^3 = a$) then b will be referred to as a *square (cube) root* of a and we will simply denote this fact by $b = \sqrt{a}$ ($b = \sqrt[3]{a}$) (the reader must be aware that an element a may have 0 or 2 square roots (if b is a square root of a then $-b$ is also a square root of a) and 0, 1, or 3 cube roots).

Remark 2 presents a complete discussion on cubic residues and computing cube roots.

Remark 2: Because p is prime and $p > 3$ then we will have the following cases:

1) If $p \equiv 2 \ mod \ 3$, then every element $a \in \mathbf{F}_p$ is a cubic residue and there is a unique cube root of a, namely $\sqrt[3]{a} = a^{\frac{2p-1}{3}}$. Indeed, in this case, $(a^{\frac{2p-1}{3}})^3 = a^{2p-1} = a^{(2p-1) \ mod \ (p-1)} = a$ (the last but one equality follows from Fermat's Theorem);
2) If $p \equiv 1 \ mod \ 3$ and $a = 1$, then a (trivial) cube root of a is 1. In order to find a non-trivial cube root of 1, we have to focus on the equation $x^2 + x + 1 = 0$. This equation has solutions in $\mathbf{F}_p$ if and only if $\Delta = -3$ is a quadratic residue modulo p. If $p \equiv 1 \ mod \ 3$ then Legendre symbol of -3 modulo p is 1 and, thus, we can determine a non-trivial cube root of 1 as $\sqrt[3]{1} = (-1 + \sqrt{-3})2^{-1}$;
3) If $p \equiv 1 \ mod \ 3$ and $a \neq 1$, then a is a cubic residue modulo p if and only if $a^{\frac{p-1}{3}} = 1$. Thus, if $a^{\frac{p-1}{3}} \neq 1$ then a has no cube roots. If $a^{\frac{p-1}{3}} = 1$ then a has exactly three cube roots. We will indicate first how to compute one of them. We will consider three subcases:
   a) If $p \equiv 7 \ mod \ 9$ then $\sqrt[3]{a} = a^{\frac{p+2}{9}}$. Indeed, in this case, $(a^{\frac{p+2}{9}})^3 = a^{\frac{p+2}{3}} = a^{1+\frac{p-1}{3}} = a$;
   b) If $p \equiv 4 \ mod \ 9$ then $\sqrt[3]{a} = (a^{\frac{p-4}{9}})^{-1}$. Indeed, in this case, $((a^{\frac{p-4}{9}})^{-1})^3 = (a^{\frac{p-4}{3}})^{-1} = (a^{\frac{p-1}{3}}a^{-1})^{-1} = (a^{-1})^{-1} = a$;
   c) If $p \equiv 1 \ mod \ 9$ then use the cubic variant of Tonelli-Shanks algorithm (presented below);

   The other two cube roots can be obtained as follows. Let ϵ be a non-trivial cube root of 1. If $\sqrt[3]{a}$ is a cube root of a then $\epsilon\sqrt[3]{a}$ and $\epsilon^2\sqrt[3]{a}$ are also cube roots of a.

Tonelli-Shanks algorithm ([12], [13]) is the most popular algorithm for computing square roots in finite fields. This algorithm has been extended by Adleman, Manders, and Miller [14] to the computation of r-th root. The cubic variant of Tonelli-Shanks algoritm is presented in Figure 3 (see also Remark 3).

```
Cubic Tonelli-Shanks(p, a)
   input:   p prime such that p ≡ 1 mod 3 and
            a ∈ F_p, a cubic residue;
   output:  b, a cube root of a
   begin
1.   find s, t such that p − 1 = 3^s t and t ≡ ±1 mod 3;
2.   repeat
3.     generate randomly d ∈ F_p
4.   until d^((p−1)/3) ≠ 1
5.   c := d^t; r := a^t; h := 1; c' := c^(3^(s−1)); c := c^(−1);
6.   for i = 1 to s − 1 do
7.     begin
8.       temp := r^(3^(s−i−1));
9.       if temp = c' then
10.        begin
11.          h := h · c;
12.          r := r · c^3;
13.        end
14.      else if temp ≠ 1 then
15.             begin
16.               h := h · c^2;
17.               r := r · (c^3)^2;
18.               c := c^3;
19.             end
20.    end{for}
21.  if t ≡ 2 mod 3 then b := a^((t+1)/3) · h;
22.                  else b := (a^((t−1)/3) · h)^(−1);
23.  return b
   end.
```

Fig. 3: Cubic Tonelli-Shanks Algorithm

Remark 3: For the clarity of the presentation, we have to make some comments:

- Step 1 decomposes $p - 1$ as $p - 1 = 3^s t$, $s \geq 1$ and t is not multiple of 3 (by repeated divisions by 3);
- A cubic non-residue $d \in \mathbf{F}_p$ is generated in Steps 2-4 (because in case $p \equiv 1 \ mod \ 3$ two thirds of elements from $\mathbf{F}_p$ are cubic non-residues, 1.5 iterations are required on average for obtaining such an element);
- Elements c and r from Step 5 satisfy the property that there exists a positive integer k such that $r = c^k$, or, equivalently, $a^t = d^{kt}$. Moreover, this element k is a multiple of 3 and, thus, $\sqrt[3]{a}$ can be computed as $a^{\frac{t+1}{3}}(d^t)^{-\frac{k}{3}}$ if $t \equiv 2 \ mod \ 3$ or as $(a^{\frac{t-1}{3}}(d^t)^{-\frac{k}{3}})^{-1}$ if $t \equiv 1 \ mod \ 3$. In Steps 6-20, the element k is determined using Pohlig-Hellman discrete logarithm algorithm [15] and the element $(d^t)^{-\frac{k}{3}}$ is stored in variable h;
- The final result is generated in Steps 21-22.

Worst-case complexity of `Cubic Tonelli-Shanks` algorithm is $\mathcal{O}((\log_2 p)^4)$.

2) Cubic Equations Modulo a Prime: Solving polynomial equations is strongly related to factoring polynomials. The most popular polynomial factorization algorithms are Berlekamp [16] and Cantor-Zassenhaus [17] (the reader is

referred to [18, Section 20] for a survey on this topic). Berlekamp algorithm requires $\mathcal{O}(n^3 + n^2\lceil\log_2 n\rceil\lceil\log_2 p\rceil)$ operations in $\mathbf{F}_p$ and Cantor-Zassenhaus algorithm requires $\mathcal{O}(n^3\lceil\log_2 p\rceil)$ operations in $\mathbf{F}_p$, where n is the degree of the polynomial. These algorithms are used in the functions `FindRoots` and `FindRoot` from *NTL* [9].

We have focused on finding dedicated (and, thus, faster) methods for solving cubic equations modulo a prime. Fortunately, there is an elegant, comprehensive paper on this topic due to Sun [19]. We include next the most important results from this paper.

Let us consider the equation $x^3 + a_1x^2 + a_2x + a_3 = 0$, where $a_1, a_2, a_3 \in \mathbf{F}_p$, for p prime, $p > 3$. The *discriminant* of this equation, denoted by D, is defined as $D = a_1^2a_2^2 - 4a_2^3 - 4a_1^3a_3 - 27a_3^2 + 18a_1a_2a_3$. A first important theorem, that states the number of solutions for the case $D \neq 0$, is presented next.

Theorem 1: ([19, Theorem 1.1]) If p is a prime, $p > 3$, $a_1, a_2, a_3 \in \mathbf{F}_p$, and the discriminant $D \neq 0$, then the number of solutions in $\mathbf{F}_p$, denoted by $N_p(x^3+a_1x^2+a_2x+a_3 = 0)$ (or, simply, as N_p), is

$$N_p = \begin{cases} 0 \text{ or } 3, & \text{if } D \text{ is a quadratic residue modulo } p; \\ 1, & \text{otherwise.} \end{cases}$$

In case $D = 0$, the following lemma presents how to compute the solutions.

Lemma 1: ([19, Lemma 4.1]) If p is a prime, $p > 3$, $a_1, a_2, a_3 \in \mathbf{F}_p$, and the discriminant $D = 0$, then $N_p(x^3 + a_1x^2+a_2x+a_3 = 0) = 3$. Moreover, the solutions (x_0, x_1, x_2) in $\mathbf{F}_p$ of the equation $x^3 + a_1x^2 + a_2x + a_3 = 0$ are[1]:

$$\begin{cases} (-\frac{a_1}{3}, -\frac{a_1}{3}, -\frac{a_1}{3}), \text{ if } a_1^2 - 3a_2 = 0; \\ (-a_1 + \frac{a_1a_2-9a_3}{a_1^2-3a_2}, \frac{-a_1a_2+9a_3}{2(a_1^2-3a_2)}, \frac{-a_1a_2+9a_3}{2(a_1^2-3a_2)}), \text{otherwise.} \end{cases}$$

The case of a single solution (D is a quadratic non-residue modulo p) is presented in the next theorem.

Theorem 2: ([19, Theorem 2.3])

Let p be a prime, $p > 3$, $a_1, a_2, a_3 \in \mathbf{F}_p$, and $a = (a_1^2 - 3a_2)^3$, $b = -2a_1^3 + 9a_1a_2 - 27a_3$ such that $ab(b^2 - 4a) \neq 0$ (remark that $D = -\frac{1}{27}(b^2 - 4a)$). The equation $x^3 + a_1x^2 + a_2x + a_3 = 0$ has one solution if and only if

$$x_0 = \frac{1}{3}((a_1^2 - 3a_2)^{-(\frac{p}{3})}v_{(p+2(\frac{p}{3}))/3}(a, b) - a_1)$$

is the solution of the equation, where the Lucas sequence $\{v_n(a, b)\}_{n\geq 0}$ is defined as $v_0(a, b) = 2$, $v_1(a, b) = b$, and $v_{n+1}(a, b) = bv_n(a, b) - av_{n-1}(a, b)$ $(n \geq 1)$.

There is an efficient algorithm (of worst-case complexity $\mathcal{O}((\log_2 p)^3)$) for evaluating Lucas sequences (see Postl [20]), based on the following properties:

$$v_{2k}(a, b) = v_k(a, b)^2 - 2a^k,$$

$$v_{2k+1}(a, b) = v_k(a, b)v_{k+1}(a, b) - ba^k.$$

The case of three solutions (D is a quadratic residue modulo p), for $p \equiv 1 \ mod\ 3$, is considered in the next theorem.

[1]The notation $\frac{x}{y}$ denotes $x \cdot y^{-1}$, for any $x, y \in \mathbf{F}_p$, $y \neq 0$.

Theorem 3: ([19, Theorem 4.5]) Let p be a prime such that $p \equiv 1 \ mod\ 3$ and $a_1, a_2, a_3 \in \mathbf{F}_p$. Let $a = (a_1^2 - 3a_2)^3$ and $b = -2a_1^3 + 9a_1a_2 - 27a_3$ and $a(b^2 - 4a) \neq 0$. Then the $x^3+a_1x^2+a_2x+a_3 = 0$ has three solutions in $\mathbf{F}_p$ if and only if $b^2 - 4a$ is a quadratic residue modulo p (there is $y \in \mathbf{F}_p$ such that $y^2 = b^2 - 4a$) and $4(b - y)$ is a cubic residue in $\mathbf{F}_p$ (let z_0, z_1, z_2 be the cube roots of $4(b - y)$). Moreover, in this case, the solutions (x_0, x_1, x_2) in $\mathbf{F}_p$ are

$$x_i = \frac{(z_i - a_1)^2 + 3(a_1^2 - 4a_2)}{6z_i}, \quad i \in \{0, 1, 2\}.$$

The case of three solutions (D is a quadratic residue modulo p), for $p \equiv 2 \ mod\ 3$, is considered in [19, Theorem 4.4] but it requires evaluating third-order recurring sequences. This part is rather complicated (and leads to inefficient implementation) and thus we omit it (for more details, the reader is referred to [19, Section 3]). In this case, in our tests, we will use the function `FindRoots` from *NTL* library [9].

We focus now on the equations from elliptic curves (see Equation 1 from Section II). Thus, we will consider next cubic equations with a least one solution, in which $a_1 = 0$, i.e., equations of form $x^3 + a_2x + a_3 = 0$ with one or three solutions. We will assume also that $a_2 \neq 0$ - otherwise, solving the obtained equation is equivalent with computing cube roots of $-a_3$, and $a_3 \neq 0$ - otherwise, 0 is a solution and the other ones are the square roots of $-a_2$. All the theoretical results presented above are combined in the algorithm `Cubic Equation`, presented in Figure 4 (see also Remark 4).

```
Cubic Equation(p, a2, a3)
   input:   p prime, p > 3, and a2, a3 ∈ F_p^*
            such that the equation x^3 + a2x + a3 = 0
            has at least one solution in F_p;
   output:  (x0, x1, x2), the solutions in F_p of the
            equation x^3 + a2x + a3 = 0
   begin
1.    a := −27a3^2; b = −27a3; d := b^2 − 4a; D = −(1/27)d;
2.    if (D = 0) then return (3a3/a2, −3a3/2a2, −3a3/2a2)
3.    if (D^((p−1)/2) ≠ 1) then
4.       begin
5.          x0 := (1/3) · (v_((p+2(p/3))/3)(a, b)) · (−3a2)^−(p/3);
6.          return (x0, x0, x0)
7.       end
8.    if p ≡ 1 mod 3 then
9.       begin
10.         y :=Square Root(d);
11.         c := 4(b − y);
12.         ε := (−1+Square Root(−3))2^−1;
13.         z0 :=Cube Root(c); z1 := z0 · ε; z2 := z1 · ε;
14.         return ((z0^2−12a2)/6z0, (z1^2−12a2)/6z1, (z2^2−12a2)/6z2)
15.      end
   end.
```

Fig. 4: Solving Cubic Equations Modulo a Prime p

Remark 4: For the clarity of the presentation, we have to make some comments on `Cubic Equation` algorithm:

- As the reader may notice, `Cubic Equation` algorithm always returns three values x_0, x_1, x_2 even in the case that the equation has only one solution x_0 (in this case we return $x_1 = x_0$ and $x_2 = x_0$). The purpose of this choice will be clear when we will introduce the new compression/decompression algorithms (in Section III-B);
- Step 5 requires computing an element of a certain Lucas sequence - this can be done in $\mathcal{O}((\log_2 p)^3)$ (according to [20]);
- We have to remark that Theorem 3 ([19, Theorem 4.5]) (which includes in fact the classical cubic formula) can be also applied for computing the unique solution in case $p \equiv 2 \ mod \ 3$ (in this case, the discriminant D is a quadratic non-residue). Indeed, in this case, $b^2 - 4a$ is a quadratic residue because $b^2 - 4a = -27D$ and -3 is a quadratic non-residue in case $p \equiv 2 \ mod \ 3$ and, moreover, any element in $\mathbf{F}_p$ is a cubic residue; for simplicity of the presentation we have not included the case $p \equiv 2 \ mod \ 3$ and D is a quadratic residue;
- The most time-consuming part of `Cubic Equation` is computing square roots (Steps 10 and 12) and cube roots (Step 13). Thus, the worst-case complexity of the `Cubic Equation` algorithm is $\mathcal{O}((\log_2 p)^4)$.

B. x-Coordinate Point Compression Method

We propose next an x-coordinate point compression method for elliptic curves over $\mathbf{F}_p$, p prime, $p > 3$, in which the size of the compressed point is at most $2 + \lceil \log_2 p \rceil$ bits. More exactly, a point (x_P, y_P) will be mapped into to $(\tilde{x}_P, y_P)$, where $\tilde{x}_P \in \{0, 1, 2\}$. The main idea is to sort (with respect to an arbitrary total ordering over $\mathbf{F}_p$, e.g., the usual ordering of positive integers or the lexicographic ordering of the binary representations) the solutions of the cubic equation $x^3 + ax + (b - y_P^2) = 0$ and identify the component x_P by its index (denoted as $\tilde{x}_P$) in the mentioned sorted list.

x-`Coordinate Compression`$(a,\ b,\ x_P,\ y_P)$

`input:` a point $P = (x_P, y_P)$ on the elliptic curve $\mathbf{E} : y^2 = x^3 + ax + b$;
`output:` the compressed point $\tilde{P} = (\tilde{x}_P, y_P)$;
`begin`
1. $(x_0, x_1, x_2) :=$`Cubic Equation`$(p, a, (b - y_P^2))$;
2 $(x_0, x_1, x_2) :=$`Sort`(x_0, x_1, x_2);
3. `find` the least element $\tilde{x}_P \in \{0, 1, 2\}$ such that $x_P = x_{\tilde{x}_P}$;
4. `return` $(\tilde{x}_P, y_P)$

`end`.

Fig. 5: x-Coordinate Compression

x-`Coordinate Decompression`$(a,\ b,\ \tilde{x}_P,\ y_P)$

`input:` a compressed point $\tilde{P} = (\tilde{x}_P, y_P)$;
`output:` the corresponding point $P = (x_P, y_P)$ on the elliptic curve $\mathbf{E} : y^2 = x^3 + ax + b$;
`begin`
1. $(x_0, x_1, x_2) :=$`Cubic Equation`$(p, a, (b - y_P^2))$;
2 $(x_0, x_1, x_2) :=$`Sort`(x_0, x_1, x_2);
3. $x_P = x_{\tilde{x}_P}$;
4. `return` (x_P, y_P)

`end`.

Fig. 6: x-Coordinate Decompression

In our compression algorithm, because $\tilde{x}_P \in \{(00)_2, (01)_2, (10)_2\}$, we may use $(11)_2$ to signal an additional information. For example, if the discriminant of the resulted cubic equation is a quadratic non-residue, we may set $\tilde{x}_P = (11)_2$ in order to indicate that the equation has a single solution - thus, computing the discriminant and testing if it is a quadratic residue are not required in the decompression phase.

The worst-case complexities of x-`Coordinate Compression` and x-`Coordinate Decompression` algorithm are the same, namely $\mathcal{O}((\log_2 p)^4)$.

We have to remark that in case of elliptic curves over $\mathbf{F}_p$, p prime, $p \equiv 1 \ mod \ 3$, given by equations of form

$$\mathbf{E} : y^2 = x^3 + b \ \ (a = 0),$$

Step 1 in y-`Coordinate Compression` and y-`Coordinate Decompression` can be directly replaced by

1.1. $temp :=$`Cube Root`$(y_P^2 - b)$;
1.2. $(x_0, x_1, x_2) = (temp, temp \cdot \epsilon, temp \cdot \epsilon^2)$;

where ϵ is a non-trivial cube root of 1.

Moreover, in this case, some additional savings can be performed in the compression phase - because x_P is a cube root of $(y_P^2 - b)$, Step 1.1 can be directly replaced by $temp := x_P$. In case p is a priori known, ϵ can be precomputed and stored in order to speed up the compression/decompression. Thus, in this case, our compression algorithm requires only two multiplications, being of complexity $\mathcal{O}((\log_2 p)^2)$.

We illustrate our x-coordinate point compression and decompression through a series of examples. We will use the elliptic curves over prime fields used in Elliptic Curve Digital Signature Algorithm (ECDSA) from Digital Signature Standard (DSS) [21], these curves being referred to as `P-xxx`.

In our first example, the discriminant D and -3 ($p \equiv 2 \ mod \ 3$) are quadratic non-residues, so we can use the cubic formula to generate the solution of the resulted cubic equation.

Example 1: Let us consider $p = 2^{192} - 2^{64} - 1$ and the elliptic curve `P-192` given by the equation $\mathbf{E} : y^2 = x^3 + ax + b$, where $a = -3$, and b is given in hexadecimal as $b =$ 64210519 e59c80e7 0fa7e9ab 72243049 feb8deec c146b9b1.

We generate a point $P = (x_P, y_P)$ on $\mathbf{E}$ with $x_P = 970995038558776078368432206432355657290010540141409250 56$, $y_P = 372897787406025110559828689773387838153511823458$.

The classical y-coordinate compression of P leads to: $\overline{P} = (970995038558776078368432206432355657290010540141409250 56, (0)_2)$, where the bit 0 states that y_P is the even solution of the equation $y^2 = x_P^3 + ax_P + b$. Thus, we get a compression to 187 bits since x_P has 186 bits.

The discriminant of the cubic equation $x^3+ax+(b-y_P^2) = 0$ is $D = 212397354386813571354971160934705649054273274564969730 9865$, that is a quadratic non-residue, so there is only one solution, the x-coordinate of P. Thus, our x-coordinate compression of P leads to $\tilde{P} = ((11)_2, 372897787406025110559828689773387838153511823458)$, where $(11)_2$ signify that the resulted cubic equation has only one solution, namely x_P. As we have y_P of only 162 bits, the compressed point is only 164 bits long and, therefore, it is reasonably smaller than the first one and far smaller than the 348-bits uncompressed representation of P.

Next, we consider $p \equiv 1 \ mod \ 3$ and, therefore, there are two main cases: $a \neq 0$ and $a = 0$.

Case 1. $a \neq 0$

In the next example, the discriminant D of the equation $x^3 + ax + (b - y^2) = 0$ is a quadratic residue, meaning there are three distinct points on the elliptic curve which have the same y-coordinate.

Example 2: Let us consider $p = 2^{256} - 2^{224} + 2^{192} + 2^{96} - 1$ and the elliptic curve `P-256` given by the equation $\mathbf{E} : y^2 = x^3 + ax + b$, where $a = -3$, and b is given in hexadecimal as b = 5ac635d8 aa3a93e7 b3ebbd55 769886bc 651d06b0 cc53b0f6 3bce3c3e 27d2604b.

We generate a point $P = (x_P, y_P)$ on $\mathbf{E}$ with $x_P = 409139172738441523365145960668656499644234251828792862899070481274490163134 40$, and $y_P = 3028798963818015963944739704565766856702837070$.

The classical y-coordinate compression of P is: $\overline{P} = (409139172738441523365145960668656499644234251828792862899070481274490163134 40, (0)_2)$, where the bit 0 states that y_P is the even solution of the equation $y^2 = x_P^3 + ax_P + b$. Thus, we get a compression of 256 bits since x_P has 255 bits.

Computing the discriminant of the cubic equation $x^3 + ax + (b - y_P^2) = 0$ we obtain the value $D = 115327517097851565109154984766540763323715048682373328754677978223828683203114$, that is a quadratic residue, and thus, there are three solutions of the mentioned equation, namely

$x_0 = 193633579762358416315220192624372333478720586777044396270888386326183704562 77$,

$x_1 = 409139172738441523365145960668656499644234251828792862899070481274490163134 40$, and

$x_2 = 555148139602762547946608316201046902177906595547065882785377445487997110842 34$ (we have already sorted the solutions). We notice that $x_P = x_1$, and, thus, our x-coordinate compression of P leads to $\tilde{P} = ((01)_2, 302879896381801596394473970456576685670231576497267214205750904837070)$, where $(01)_2$ signify the index of x_P among the sorted solutions.

As we have y_P of only 228 bits, the compressed point is only 230 bits long and, therefore, it is reasonably smaller than the first compression and far smaller than the 483 uncompressed representation of P.

Case 2. $a = 0$

Example 3: Let us consider the prime $p = 2^{224} - 2^{96} + 1$ and the elliptic curve given by equation $\mathbf{E} : y^2 = x^3 + b$, with b randomly generated: $b = 5767724673982617133747126870444140963248276118734545338098933632417$. Let $P = (x_P, y_P)$ on $\mathbf{E}$ with $x_P = 2395405760686290323856732945809111999924373746187422477256033352218 3$ and $y_P = 49408731706668410068153578971347434436875566061947142537691 4$.

Since x_P is 224 bits long and y_P is even, the y-coordinate compression of P has 225 bits: $\overline{P} = (2395405760686290323856732945809111999924373746187422477256033352218 3, (0)_2)$.

Next, in order to determine the x-coordinate compression, we compute a cube root of 1 in $\mathbf{F}_p$, $\epsilon = 113518326235439584354877412922381102907197250630999745267807984808 55$, and the other two solutions of $x^3 = y_P^2 - b$ as $x_P \cdot \epsilon$ and $x_P \cdot \epsilon^2$, obtaining the values $147023515948129702631307670787931282591724857423165028236114069484 48$ and $152634841326254060876359336371550130886996093158618886908484392127131$. We notice that x_P is the greatest solution among all, and, thus, the compressed x-coordinate will be $\tilde{x}_P = (10)_2$. Moreover, y has a 199 bit representation and facilitates a x-coordinate compression of only 201 bits, significantly shorter than the previous one:

$\tilde{P} = ((10)_2, 4940873170666841006815357897134743443687556606194714253769 14)$,

IV. Experiments

The implementation is written in Visual Studio 2008 on a Intel Core 2 Duo, running on a 2.53 GHz Dell laptop, under Windows 7 operating system. For operations with large integers we have used the *NTL* library [9] but we have implemented our own functions for computing square/cube roots and solving cubic equations. The code has not been completely optimized. We have compared our algoritms with the algorithms implemented in *NTL* in case of solving cubic equations or finding a cube root. The average times have been measured in milliseconds.

For the case of cubic equations with a single solution, we have implemented a binary algorithm for computing the Lucas number $v_n(a, b)$ and we have compared our `Cubic Equation` algorithm with `FindRoot` from *NTL*. The results are presented in Table I.

Algorithm	Curve				
	P-192	P-224	P-256	P-384	P-521
FindRoot	8.66	18.46	20.14	76.86	192.64
Cubic Equation	1.42	2.54	3.48	4.99	12.78

TABLE I

In case $p \equiv 1 \ mod \ 3$ and the discriminant is a quadratic residue (the cubic equation has three solutions), we have compared our `Cubic Equation` algorithm with `FindRoots` from *NTL*. The results are presented in Table II.

Algorithm	Curve		
	P-224	P-256	P-521
FindRoots	43.08	44.32	202.78
Cubic Equation	8.13	8.62	44.95

TABLE II

Finally, we have compared our `Cubic Tonelli-Shanks` algorithm (denoted as `CTS`) with `FindRoot` from *NTL* for the case of computing a cube root. The results are presented in Table III.

Algorithm	Size of p					
	192	224	256	384	521	1024
FindRoot	8.66	18.46	20.14	76.86	192.64	854.47
CTS	2.47	4.14	5.49	12.94	30.44	188

TABLE III

V. Conclusions and Future Work

In this paper we have proposed an x-coordinate point compression method for elliptic curves over $\mathbf{F}_p$, where $p > 3$ is prime, as an alternative to the classical y-coordinate point compression method. A point $P = (x, y)$ will be compressed as $\tilde{P} = (\tilde{x}, y)$ where $\tilde{x}$ has only two bits and, thus, our method allows more compact representations when $\lceil \log_2 x \rceil > \lceil \log_2 y \rceil + 1$. Both compression and decompression involve solving cubic equations or, in some cases, only computing cube roots modulo a prime - we have surveyed and tested the most important algorithms on this topic. Our decompression algorithm has the same complexity as the classical one ($\mathcal{O}((\log_2 p)^4)$). Although our compression algorithm is less time-efficient than the classical y-coordinate point compression, it can be used in applications in which the compactness of points is the deciding factor. However, for some particular cases, our compression algorithm can be significantly improved ($\mathcal{O}((\log_2 p)^2)$).

A very interesting topic is double point compression, that has been considered by Khabbazian et al. in [22] for situations that require transmitting/storing multiple points (e.g., the elliptic curve ElGamal encryption scheme). For example, Khabbazian et al. have remarked that two points $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$ may be simultaneously compressed as $(x_1, x_2, y_1 + y_2)$ - thus the two y-coordinates are combined into one. It will be interesting to find a method of simultaneously compress two points $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$ as $(x_1 \oplus x_2, y_1, y_2)$ and we will consider this problem in our future work.

References

[1] N. Koblitz, "Elliptic curve cryptosystems," *Mathematics of Computation*, vol. 48, no. 177, pp. 203–209, 1987.

[2] V. Miller, "Use of elliptic curves in cryptography," in *Advances in Cryptology - CRYPTO '85*, ser. Lecture Notes in Computer Science, H. Williams, Ed., vol. 218. Springer-Verlag, 1986, pp. 417–426.

[3] A. Menezes, "Elliptic curve cryptosystems," RSA Laboratories' CryptoBytes, Volume 1, Number 2, 1995.

[4] IEEE Std 2000-1363, "Standard specifications for public-key cryptography," 2000.

[5] G. Seroussi, "Compact representation of elliptic curve points over F_{2^n}," HP Labs Technical Report HPL-98-94R1, 1998, http://www.hpl.hp.com/techreports/.

[6] B. King, "A point compression method for elliptic curves defined over GF(2^n)," in *Public Key Cryptography - PKC 2004, 7th International Workshop on Theory and Practice in Public Key Cryptography*, ser. Lecture Notes in Computer Science, F. Bao, R. Deng, and J. Zhou, Eds., vol. 2947. Springer-Verlag, 2004, pp. 333–345.

[7] B. Brumley and K. Järvinen, "Fast point decompression for standard elliptic curves," in *Public Key Infrastructure, 5th European PKI Workshop: Theory and Practice, EuroPKI 2008*, ser. Lecture Notes in Computer Science, S. Mjølsnes, S. Mauw, and S. Katsikas, Eds., vol. 5057. Springer-Verlag, 2008, pp. 134–149.

[8] P. Eagle and S. Galbraith, "Point compression for Koblitz elliptic curves," Cryptology ePrint Archive, Report 2009/086, 2009, http://eprint.iacr.org/.

[9] V. Shoup, NTL: A Library for doing Number Theory (version 5.5.2) , http://www.shoup.net/ntl.

[10] D. Hankerson, A. Menezes, and S. Vanstone, *Guide to Elliptic Curve Cryptography*. Springer-Verlag, 2004.

[11] J. Bos, M. Kaihara, T. Kleinjung, A. Lenstra, and P. Montgomery, "On the Security of 1024-bit RSA and 160-bit Elliptic Curve Cryptography," Cryptology ePrint Archive, Report 2009/389, 2009, http://eprint.iacr.org/.

[12] A. Tonelli, "Bemerkung über die Auflösung quadratischer Congruenzen," *Göttinger Nachrichten*, pp. 344–346, 1891.

[13] D. Shanks, "Five number-theoretic algorithms," in *Proceedings of the second Manitoba conference on numerical mathematics*, ser. Congressus Numerantium, R. Thomas and H. Williams, Eds., vol. 7. Utilitas Mathematica, 1973, pp. 51–70.

[14] L. Adleman, K. Manders, and G. Miller, "On taking roots in finite fields," in *18th Annual Symposium on Foundations of Computer Science (FOCS 1977)*. IEEE Computer Society, 1977, pp. 175–178.

[15] S. Pohlig and M. Hellman, "An improved algorithm for computing logarithms over GF(p) and its cryptographic significance," *IEEE Transactions on Information Theory*, vol. 24, pp. 106–110, 1978.

[16] E. R. Berlekamp, "Factoring polynomials over finite fields," *Bell System Technical Journal*, vol. 46, pp. 1853–1859, 1967.

[17] D. Cantor and H. Zassenhaus, "A new algorithm for factoring polynomials over finite fields," *Mathematics of Computation*, vol. 36, no. 154, pp. 587–592, 1981.

[18] V. Shoup, *A Computational Introduction to Number Theory and Algebra*. Cambridge University Press, 2008.

[19] Z.-H. Sun, "Cubic and quartic congruences modulo a prime," *Journal of Number Theory*, vol. 102, no. 1, pp. 41–89, 2003.

[20] H. Postl, "Fast evaluation of Dickson polynomials," *Contributions to General Algebra*, vol. 6, pp. 223–225, 1988.

[21] FIPS 186-3, "Digital Signature Standard", Federal Information Processing Standards Publication 186, June 2009, http://csrc.nist.gov/publications/fips/fips186-3/fips_186-3.pdf.

[22] M. Khabbazian, T. Gulliver, and V. Bhargava, "Double point compression with applications to speeding up random point multiplication," *IEEE Transactions on Computers*, vol. 56, no. 3, pp. 305–313, 2007.