Multiple Point Compression on Elliptic Curves

by

Adilet Otemissov

Nazarbayev University
2015

Certificate of Approval

This is to certify that the accompanying thesis by Adilet Otemissov has been accepted.

Francesco Sica, Ph.D.

Kira Adaricheva, Ph.D.

Nazarbayev University
April 20, 2015

ABSTRACT OF THESIS

MULTIPLE POINT COMPRESSION ON ELLIPTIC CURVES

The paper aims at developing new point compression algorithms which are useful in mobile communication systems where Elliptic Curve Cryptography is employed to achieve secure data storage and transmission. Compression algorithms allow elliptic curve points to be represented in the form that balances the usage of memory and computational power. The two- and three-point compression algorithms developed by Khabbazian, Gulliver and Bhargava [4] are reviewed and extended to generic cases of four and five points. The proposed methods use only basic operations (multiplication, division, etc.) and avoids square root finding. In addition, a new two-point compression method which is heavy in compression phase and light in decompression is developed.

Adilet Otemissov
Nazarbayev University
April 2015

Acknowledgments

I thank all who helped me with my Capstone Thesis. Special thanks to Francesco Sica who provided a shorter proof of Theorem 1 which is presented in this paper.

Contents

1 Introduction

Cryptosystems based on elliptic curves are becoming more widespread nowadays. They are used in various communication systems for protecting and securing information from a third party. Elliptic curve cryptography (ECC) is based on the group structure of elliptic curves and its security stems from the difficulty of solving the discrete logarithm problem. The main advantage of elliptic curve cryptosystems compared to others is a small key size. For example, ECC with 233-bit key size is as secure as RSA with 2048-bit key. The table below compares the public key sizes of ECC and RSA given the same security level [5].

	Commensurable key sizes (bits)				
ECC	163	233	283	409	571
RSA	1024	2048	3072	7680	15360

The small key size makes ECC applicable in mobile communication systems which have limited computational power and memory [1, 2, 6]. Mobile devices such as smart cards, RFID tags, handheld PCs and many others find ECC as the best solution to secure data transmission.

Implementation of ECC usually requires transmission and storage of elliptic curve points which may be limited in resource-constrained environments. Therefore, point compression methods exist to represent elliptic curve points

in a way that finds a balance between the use of memory and computational power to ease transmission and storage. Previously, the work on the compact representation of two and three elliptic curve points were done by Khabbazian, Gulliver and Bhargava in [4]. This paper extends their work to four and five points and suggest a new two-point compression algorithm. All proposed algorithms only use basic operations such as multiplication, squaring, etc. and avoid square root extractions (refer to Appendix A and B).

In Section 2, I provide definition and properties of an elliptic curve followed by the previous work done by Khabbazian, Gulliver and Bhargava in the following section. Section 4 introduces new two-, four- and five-point compression algorithms and the work is concluded in Section 5.

2 Elliptic Curves

Before plunging into compression algorithms it is worth first to define what elliptic curves are and mention their properties.

An elliptic curve E over a field $\mathbb{F}$ is defined as a genus 1 curve with $\mathbb{F}$-rational points. It can be written in the generalized Weierstrass form

$$y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6 \tag{1}$$

where a_i's are constants in $\mathbb{F}$. In cryptography, the field $\mathbb{F}$ is usually considered to be a finite field $\mathbb{F}_p$ for some large prime p or $\mathbb{F}_{2^n}$.

It can be shown that if the characteristic of the field $Char(\mathbb{F}) \neq 2$, then (1) can be reduced to the following form

$$y^2 + xy = x^3 + ax^2 + b \tag{2}$$

For the case $Char(\mathbb{F}) \neq 2, 3$ Weierstrass equation simplifies to

$$y^2 = x^3 + ax + b \tag{3}$$

Elliptic curves are not allowed to have multiple roots. That is, they do not have cusps and self-intersections and this restriction for the elliptic curve (3) can be written as

$$4a^3 + 27b^2 \neq 0$$

Furthermore, we define $E(\mathbb{F})$ to be the set of elliptic curve points $P = (x, y)$ with a point at infinity (∞). Mathematically speaking,

$$E(\mathbb{F}) = \{\infty\} \cup \{(x, y) \in \mathbb{F}^2 : y^2 = x^3 + ax + b\}$$

The infinity point(∞) is thought to be located both at the top and at the bottom of y-axis. Hence, a line that connects any elliptic curve point (x_1, y_1)

and the point at infinity is a vertical line $x = x_1$.

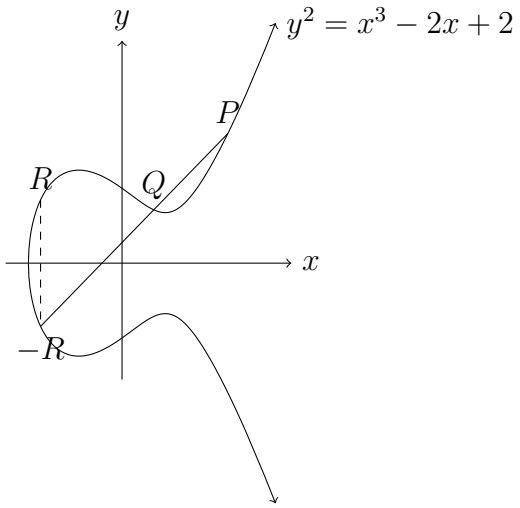
The set of elliptic curve points together with an addition operation form an abelian group and the point at infinity (∞) serves as an identity element. This property makes elliptic curves useful in cryptography. Let $P = (x_1, y_1)$ and $Q = (x_2, y_2)$ be elliptic curve points. Addition between points P and Q is performed and the resulting point $R = (x_3, y_3) = P + Q$ is obtained in the following way.

$Char(\mathbb{F}) = 2$	$P = Q$	$s = \frac{y_1}{x_1} + x_1$ $x_3 = s^2 + s + a$ $y_3 = (x_1 + x_3)s + y_1 + x_3$
	$P \neq \pm Q$	$s = \frac{y_2 + y_1}{x_2 + x_1}$ $x_3 = s^2 + s + x_1 + x_2 + a$ $y_3 = (x_1 + x_3)s + y_1 + x_3$
$Char(\mathbb{F}) \neq 2, 3$	$P = Q$	$s = \frac{3x_1^2 + a}{2y_1}$ $x_3 = s^2 - 2x_1$ $y_3 = (x_1 - x_3)s - y_1$
	$P \neq \pm Q$	$s = \frac{y_2 - y_1}{x_2 - x_1}$ $x_3 = s^2 - x_1 - x_2$ $y_3 = (x_1 - x_3)s - y_1$

The addition operation is illustrated in the pictures below for the curve $y^2 = x^3 - 2x + 2$.

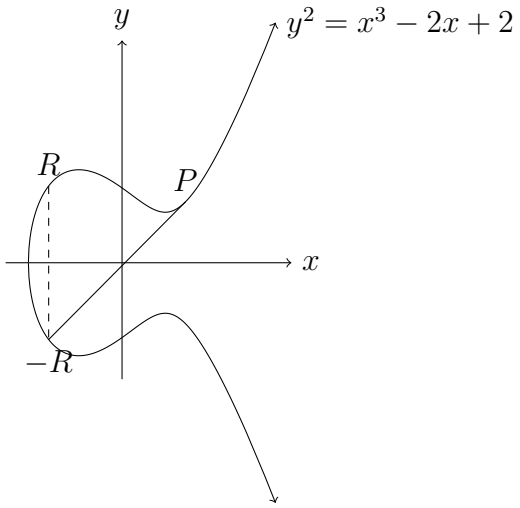
Case 1: $P \neq \pm Q$

A line that connects points P and Q is drawn and its intersection with the curve at a third point is reflected with respect to x-axis.



Case 2: $P = Q$

A tangent line at a point P is drawn and its intersection with the curve at a second point is reflected with respect to x-axis



3 Previous Work

Multiple point compression algorithms were first introduced by Khabbazian, Gulliver, and Bhargava [4]. They developed compression methods for two and three elliptic curve points. The general idea of their methods is to store values of x-coordinates and an additional value of $\alpha = y_1 + ... + y_n$. In this section, Khabbazian, Gulliver, and Bhargava's two- and three- point compression algorithms are described for the case $Char(\mathbb{F}) \neq 2, 3$.

3.1 Two-Point Compression Algorithm

Having an elliptic curve $y^2 = x^3 + ax + b$ and elliptic curve points $P_1 = (x_1, y_1), P_2 = (x_2, y_2)$ the points can be represented in a compact way by three field elements and one extra bit. The points are represented by $(x_1, x_2, \alpha = y_1 + y_2)$ and extracted through the following decompression algorithm:

We have

$$y_1^2 = x_1^3 + ax_1 + b$$

$$y_2^2 = x_2^3 + ax_2 + b$$

Then,

$$\begin{aligned} y_1^2 - y_2^2 &= (x_1 - x_2)(x_1^2 + x_1x_2 + x_2^2 + a) \\ &= \frac{(x_1 - x_2)(3(x_1 + x_2)^2 + (x_1 - x_2)^2 + 4a)}{4} \end{aligned}$$

On the other hand,

$$y_1^2 - y_2^2 = (y_1 + y_2)(y_1 - y_2) = \alpha(y_1 - y_2)$$

Therefore, y_1 and y_2 can be derived by linear algebra

$$y_1 = \frac{\alpha}{2} + (8\alpha)^{-1}(x_1 - x_2)(3(x_1 + x_2)^2 + (x_1 - x_2)^2 + 4a)$$

and

$$y_2 = \alpha - y_1$$

The special case when $y_1 + y_2 = 0$ is treated differently. In this case, the values (x_1, x_2, y_1) are stored and y_2 can be found simply by $y_2 = -y_1$. An additional bit is required to distinguish between these two cases: 1 and 0 are stored when $\alpha \neq 0$ and $\alpha = 0$, respectively.

The overall computational complexity of the decompression algorithm is $2M + 2S + I$.

3.2 Three-Point Compression Algorithm

Khabbazian, Gulliver and Bhagarva also showed that three elliptic curve points $P_1 = (x_1, y_1), P_2 = (x_2, y_2)$ and $P_3 = (x_3, y_3)$ can be decompressed storing four field elements $(x_1, x_2, x_3, \alpha = y_1 + y_2 + y_3)$ and two extra bits. In this case, the decompression algorithm works as follows:

Let $\beta = \alpha^2 + y_3^2 - y_1^2 - y_2^2$. Then,

$$y_3 = y_3 + \frac{(2y_1y_2)^2 - 4y_1^2y_2^2}{4\alpha\beta} = y_3 + \frac{(\alpha^2 + y_3^2 - y_1^2 - y_2^2 - 2\alpha y_3)^2 - 4y_1^2y_2^2}{4\alpha\beta}$$

$$= \frac{(\beta - 2\alpha y_3)^2 + 4\alpha\beta y_3 - 4y_1^2y_2^2}{4\alpha\beta} = \frac{\beta^2 + 4\alpha^2 y_3^2 - 4y_1^2y_2^2}{4\alpha\beta}$$

Note that y_i^2 can be obtained from the elliptic curve equation $y_i^2 = x_i(x_i^2 + a) + b$. After y_3 is found we obtain

$$y_1 + y_2 = \alpha - y_3$$

Then, y_1 and y_2 can be derived using the two-point decompression method described in 3.1. The overall computational cost of the algorithm is $I + 12M + 6S$.

The above algorithm fails when $\beta = 0$ or $\alpha = 0$ requiring special cases to be considered separately. The treatment of special cases is summarised in the table below:

Case	Store	Comments
$y_i^2 \neq y_j^2$ for $i \neq j$ and $\alpha = 0$	$(x_1, x_2, x_3, y_1 + y_2)$	y_1 and y_2 are found using two-point decompression algorithm, y_3 is calculated as $y_3 = -y_1 - y_2$
$y_1^2 = y_2^2 \neq y_3^2$	$(x_1, x_2, x_3, y_2 + y_3, b_1)$ where b_1 is a bit	y_2 and y_3 are found using two - point decompression algorithm. If $y_1 = y_2$ store $b_1 = 1$, otherwise set $b_1 = 0$
$y_1^2 = y_2^2 = y_3^2$	$(x_1, x_2, x_3, y_1, b_1, b_2)$	If $y_1 = y_2$ store $b_1 = 1$, otherwise set $b_1 = 0$. If $y_1 = y_3$ store $b_2 = 1$, otherwise set $b_2 = 0$.

4 New Point Compression Methods: $n = 2, 4, 5$

It is worth mentioning that if multiple point compression algorithms are not employed, then elliptic curve points can be represented in two ways. The first method (ordinary) suggests to store $(x_1, x_2, ...x_n)$ and extract values of y-coordinates using the elliptic curve equation thereby performing n square root operations. The second (trivial) stores all values of x and y coordinates: $(x_1, x_2, ...x_n, y_1, y_2, ..., y_n)$. The ordinary method requires twice less memory than the second, but it is more computationally costly. The trivial method does not perform any computations but demands large amount of memory.

In this case, multiple point compression algorithms offer a balanced approach to point representation. Compared to the ordinary method, multiple point compression algorithms require slightly more memory but are considerably less computationally costly. However, when compared to the trivial method, multiple point compression algorithms are more computationally complex but demand considerably less amount of memory.

All the proposed algorithms in the following sections are developed for the case $Char(\mathbb{F}) \neq 2, 3$.

4.1 Alternative two-point compression algorithm

This section introduces a new two-point compression method different from what was developed by Khabbazian, Gulliver, and Bhargava. The method's distinguishing feature is its computational complexity in the compression phase while decompression stage requires only a few multiplications and square operations.

Given points $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$ and their representation by $(A = \dfrac{x_1 - x_2}{y_1 - y_2}, B = y_1 - y_2, x_2, b)$, where b is an additional bit, values of x_1, x_2, y_1, and y_2 can be obtained through the following computations:

x_1 can easily be found by $x_1 = AB + x_2$. Then, y-coordinates are obtained with the use of elliptic curve equations.

Since

$$y_1^2 = x_1^3 + ax_1 + b$$

$$y_2^2 = x_2^3 + ax_2 + b$$

We have the following

$$y_1^2 - y_2^2 = (x_1 - x_2)(x_1^2 + x_1x_2 + x_2^2 + a)$$

Hence,

$$y_1 + y_2 = \frac{x_1 - x_2}{y_1 - y_2}((x_1 + x_2)^2 - x_1x_2 + a) = A((x_1 + x_2)^2 - x_1x_2 + a)$$

We also have $y_1 - y_2 = B$. Then,

$$y_1 = \frac{A((x_1 + x_2)^2 - x_1x_2 + a) + B}{2}$$

$$y_2 = y_1 - B$$

The computational cost of the compression phase is I, while decompression phase requires only $3M$ and $1S$. Compared to the previous two-point compression method this algorithm is more costly in the compression stage while being computationally cheaper in the decompression.

The algorithm fails, however, when $y_1 - y_2 = 0$. In this case, values (x_1, x_2, y_1)

are stored with an additional bit to differentiate between two cases. y_2 is decompressed simply by $y_2 = y_1$.

4.2 Elementary Symmetric Polynomials: lemma and theorem

A lemma and a theorem are introduced in this section which are essential to developing four- and five-point compression algorithms. In the lemma, an algebraic identity involving elementary symmetric polynomials is derived. Then, the theorem establishes a relation between elementary symmetric polynomials and their indepedent variables.

Recall the definition of elementary symmetric polynomials:

$$e_{k,m} = \begin{cases} \sum_{1 \le i_1 < i_2 \ldots < i_k \le m} y_{i_1} y_{i_2} \ldots y_{i_k} & \text{if } k \le m \\ 1 & \text{if } k = 0 \\ 0 & \text{if } k > m \end{cases}$$

where $k \in \mathbb{Z}^*$ and $m \in \mathbb{N}$.

Lemma 1.

$$\sum_{j=1}^{k} (-1)^{j+1} e_{k+j,m} e_{k-j,m} = \frac{e_{k,m}^2 - f_{k,m}}{2} \quad \textit{for } k \le m \tag{4}$$

where $e_{k,m}$ is an elementary symmetric polynomial and $f_{k,m}$ is defined as follows:

$$f_{k,m} = \begin{cases} \sum_{1 \le i_1 < i_2 \ldots < i_k \le m} y_{i_1}^2 y_{i_2}^2 \ldots y_{i_k}^2 & \text{if } k \le m \\ 1 & \text{if } k = 0 \\ 0 & \text{if } k > m \end{cases}$$

Proof. The principle of mathematical induction is used on m to prove the lemma.

Base case ($m = 1$):

k can only be equal 1. Hence, LHS of (4) is

$$\sum_{j=1}^{1} e_{1+j,1} e_{1-j,1} = e_{2,1} e_{0,1} = 0$$

On the other hand, RHS of (4) is

$$\frac{e_{1,1}^2 - f_{1,1}}{2} = \frac{y_1^2 - y_1^2}{2} = 0$$

Therefore, equation (4) is true for $m = 1$.

Before starting inductive step, first, notice that the following equations

$$\sum_{1 \le i_1 < \ldots < i_k \le m+1} y_{i_1} y_{i_2} \ldots y_{i_k} = \sum_{1 \le i_1 < \ldots < i_k \le m} y_{i_1} y_{i_2} \ldots y_{i_k} + \sum_{1 \le i_1 < \ldots < i_{k-1} \le m} y_{i_1} \ldots y_{i_{k-1}} y_{m+1}$$

$$\sum_{1\le i_1<\ldots<i_k\le m+1} y_{i_1}^2 y_{i_2}^2 \ldots y_{i_k}^2 = \sum_{1\le i_1<\ldots<i_k\le m} y_{i_1}^2 y_{i_2}^2 \ldots y_{i_k}^2 + \sum_{1\le i_1<\ldots<i_{k-1}\le m} y_{i_1}^2 \ldots y_{i_{k-1}}^2 y_{m+1}^2$$

are true for $1 \le k \le m$. The above equations are equivalent to

$$e_{k,m+1} = e_{k,m} + y_{m+1} e_{k-1,m} \text{ for } 1 \le k \le m \tag{5}$$

$$f_{k,m+1} = f_{k,m} + y_{m+1}^2 f_{k-1,m} \text{ for } 1 \le k \le m \tag{6}$$

(5) and (6) are also true for the case $k = m + 1$ since, by definition, $e_{m+1,m}$ and $f_{m+1,m}$ are equal zero.

Inductive step:

Assume that (4) is true for all positive integers $m \le n$ and for all $k \le m$. We must show that the equation is true for $n + 1$ and for all $k \le n + 1$. In other words, we must show that the following equation is true.

$$\sum_{j=1}^{k} (-1)^{j+1} e_{k+j,n+1} e_{k-j,n+1} = \frac{e_{k,n+1}^2 - f_{k,n+1}}{2} \tag{7}$$

Using equation (5), the LHS of the above equation becomes

$$\sum_{j=1}^{k}(-1)^{j+1}e_{k+j,n+1}e_{k-j,n+1} = \sum_{j=1}^{k-1}(-1)^{j+1}e_{k+j,n+1}e_{k-j,n+1} + (-1)^{k+1}e_{2k,n+1}$$

$$= \sum_{j=1}^{k-1}(-1)^{j+1}(e_{k+j,n} + y_{n+1}e_{k+j-1,n})(e_{k-j,n} + y_{n+1}e_{k-j-1,n}) + (-1)^{k+1}e_{2k,n+1}$$

$$= \sum_{j=1}^{k-1}(-1)^{j+1}e_{k+j,n}e_{k-j,n} + y_{n+1}\sum_{j=1}^{k-1}(-1)^{j+1}(e_{k+j-1,n}e_{k-j,n} + e_{k+j,n}e_{k-j-1,n})$$

$$+y_{n+1}^2\sum_{j=1}^{k-1}(-1)^{j+1}e_{k-1+j,n}e_{k-1-j,n} + (-1)^{k+1}(e_{2k,n} + y_{n+1}e_{2k-1,n})$$

Consider the above sums separately. The following equations

$$\sum_{j=1}^{k-1}(-1)^{j+1}e_{k+j,n}e_{k-j,n} + (-1)^{k+1}e_{2k,n} = \sum_{j=1}^{k}(-1)^{j+1}e_{k+j,n}e_{k-j,n} = \frac{e_{k,n}^2 - f_{k,n}}{2} \tag{8}$$

$$y_{n+1}^2\sum_{j=1}^{k-1}(-1)^{j+1}e_{k-1+j,n}e_{k-1-j,n} = y_{n+1}^2\frac{e_{k-1,n}^2 - f_{k-1,n}}{2} \tag{9}$$

are true by inductive hypothesis. The remaining sum

$$y_{n+1}\sum_{j=1}^{k-1}(-1)^{j+1}(e_{k+j-1,n}e_{k-j,n} + e_{k+j,n}e_{k-j-1,n})$$

telescopes, leaving

$$y_{n+1}(e_{k,n}e_{k-1,n} + (-1)^k e_{2k-1,n})$$

Therefore,

$$y_{n+1}\sum_{j=1}^{k-1}(-1)^{j+1}(e_{k+j-1,n}e_{k-j,n}+e_{k+j,n}e_{k-j-1,n})+(-1)^{k+1}y_{n+1}e_{2k-1,n}=y_{n+1}e_{k,n}e_{k-1,n} \tag{10}$$

Combining equations (8),(9), and (10) we obtain

$$\begin{aligned}\sum_{j=1}^{k}(-1)^{j+1}e_{k+j,n+1}e_{k-j,n+1} &= \frac{e_{k,n}^2-f_{k,n}}{2}+y_{n+1}e_{k,n}e_{k-1,n}+y_{n+1}^2\frac{e_{k-1,n}^2-f_{k-1,n}}{2}\\ &= \frac{(e_{k,n}+y_{n+1}e_{k-1,n})^2-(f_{k,n}+y_{n+1}^2f_{k-1,n})}{2}\\ &= \frac{e_{k,n+1}^2-f_{k,n+1}}{2}\end{aligned}$$

This completes the inductive step. □

The following theorem demonstrates how y-coordinates can be obtained without square root operations when squares of y-coordinates and values of elementary symmetric polynomials are known.

Theorem 1. *Let $e_{k,n}$ be an elementary symmetric polynomial in variables $y_1,...,y_n$. Then for odd values of n:*

$$y_i=\frac{e_{n,n}+y_i^2e_{n-2,n}+...+y_i^{n-1}e_{1,n}}{e_{n-1,n}+y_i^2e_{n-3,n}+...+y_i^{n-3}e_{2,n}+y_i^{n-1}} \quad \textit{for } i=1,...,n \tag{11}$$

For even values of n:

$$y_i = \frac{e_{n,n} + y_i^2 e_{n-2,n} + ... + y_i^{n-2} e_{2,n} + y_i^n}{e_{n-1,n} + y_i^2 e_{n-3,n} + ... + y_i^{n-2} e_{1,n}} \ \ for \ \ i = 1, ..., n \tag{12}$$

Proof.

$$\prod_{i=1}^{n} (x + y_i) = x^n + e_{1,n} x^{n-1} + e_{2,n} x^{n-2} + ... + e_{n-1,n} x + e_{n,n}$$

Substituting $x = -y_i$ we have the following

$$0 = (-1)^n y_i^n + (-1)^{n-1} e_{1,n} y_i^{n-1} + ... - e_{n-1,n} y_i + e_{n,n}$$

Then, for odd values of n it follows that

$$y_i(y_i^{n-1} + y_i^{n-3} e_{2,n} + ... + y_i^2 e_{n-3,n} + e_{n-1,n}) = y_i^{n-1} e_{1,n} + ... + y_i^2 e_{n-2,n} + e_{n,n}$$

Hence,

$$y_i = \frac{e_{n,n} + y_i^2 e_{n-2,n} + ... + y_i^{n-1} e_{1,n}}{e_{n-1,n} + y_i^2 e_{n-3,n} + ... + y_i^{n-3} e_{2,n} + y_i^{n-1}}$$

For even values of n:

$$y_i(y_i^{n-2} e_{1,n} + ... + y_i^2 e_{n-3,n} + e_{n-1,n}) = y_i^n + y_i^{n-2} e_{2,n} ... + y_i^2 e_{n-2,n} + e_{n,n}$$

Thus,

$$y_i = \frac{e_{n,n} + y_i^2 e_{n-2,n} + ... + y_i^{n-2} e_{2,n} + y_i^n}{e_{n-1,n} + y_i^2 e_{n-3,n} + ... + y_i^{n-2} e_{1,n}}$$

□

On the basis of Lemma 1 and Theorem 1 four- and five-point compression algorithms are developed in the next sections.

4.3 Four-Point Compression Algorithm

A generic four-point compression algorithm can easily be derived with the help of previously established theorem and lemma. In this case, points $P = (x_i, y_i)$ for $i = 1, 2, 3, 4$ are represented by $(x_1, x_2, x_3, x_4, e_{1,4} = y_1 + y_2 + y_3 + y_4)$. First, we find values of elementary symmetric polynomials using Lemma 1 and then obtain the values of y-coordinates according to Theorem 1.

For simplicity of notation let $e_i = e_{i,4}$ and $f_i = f_{i,4}$ for i = 1,2,3,4. Then, from the lemma we have

$$e_2 = \frac{e_1^2 - f_1}{2} \tag{13}$$

$$e_1 e_3 - e_4 = \frac{e_2^2 - f_2}{2} \tag{14}$$

$$e_2 e_4 = \frac{e_3^2 - f_3}{2} \tag{15}$$

Note that since y_i^2's are all known, f_i's can be easily found; therefore, from (13) we can compute e_2. Denote the RHS of (14) as A. Then, from the same equa-

tion we have

$$e_3 = \frac{A + e_4}{a_1} \tag{16}$$

Next, substitute e_3 in (15) by (16). We obtain the quadratic equation in e_4

$$2e_1^2 e_2 e_4 = A^2 + 2Ae_4 + e_4^2 - e_1^2 f_3 \tag{17}$$

e_4^2 is known because $e_4^2 = f_4 = y_1^2 y_2^2 y_3^2 y_4^2$. Hence,

$$e_4 = \frac{A^2 + e_4^2 - e_1^2 f_3}{2(e_1^2 e_2 - A)} \tag{18}$$

Next, substitute e_4 in (16) by the above expression. This yields

$$e_3 = \frac{2e_1^2 e_2 A - A^2 + e_4^2 - e_1^2 f_3}{2e_1(e_1^2 e_2 - A)} \tag{19}$$

According to Theorem 1, we have

$$y_i = \frac{e_4 + y_i^2 e_2 + y_i^4}{e_3 + y_i^2 e_1}, \text{ for } i = 1, 2, 3, 4. \tag{20}$$

Combining equations (18), (19) and (20) we obtain the following equation for y_i

$$y_i = \frac{e_1(A^2 + e_4^2 - e_1^2 f_3 + 2(e_1^2 e_2 - A)(y_i^2 e_2 + y_i^4))}{2e_1^2 e_2 A - A^2 + e_4^2 - e_1^2 f_3 + 2e_1^2 y_i^2 (e_1^2 e_2 - A)} \tag{21}$$

This method requires 55 multiplications, 11 square operations and 1 inver-

sion. For more detailed information on the computation of the number of operations refer to Appendix A.

4.4 Five-Point Compression Algorithm

A five-point compression algorithm is developed in a similar fashion. Having elliptic points $P = (x_i, y_i)$ for $i = 1, 2, 3, 4, 5$ we store $(x_1, x_2, x_3, x_4, x_5, e_{1,5} = y_1 + y_2 + y_3 + y_4 + y_5)$.

Again, to simplify notation let $e_i = e_{i,5}$ and $f_i = f_{i,5}$ for $i = 1, 2, 3, 4, 5$. According to Lemma 1 we have

$$e_2 = \frac{e_1^2 - f_1}{2} \tag{22}$$

$$e_3 e_1 - e_4 = \frac{e_2^2 - f_2}{2} \tag{23}$$

$$e_4 e_2 - e_5 e_1 = \frac{e_3^2 - f_3}{2} \tag{24}$$

$$e_5 e_3 = \frac{e_4^2 - f_4}{2} \tag{25}$$

Because e_1 and f_1 are known, e_2 is easily found from (22). Let RHS of (23) be denoted as A. Then, from (23) a linear combination between e_3 and e_4 is established

$$e_3 = \frac{A + e_4}{e_1} \tag{26}$$

Next, substituting the above expression into (24) it yields

$$e_4 e_2 - e_5 e_1 = \frac{A^2 + 2Ae_4 + e_4^2 - e_1^2 f_3}{2e_1^2} \tag{27}$$

From which we express e_4 to plug it in (25)

$$e_4^2 = e_1^2(2e_4 e_2 - 2e_5 e_1 + f_3) - A^2 - 2Ae_4 \tag{28}$$

Combining (28), (26) and (25) we get

$$e_5 \frac{A + e_4}{e_1} = \frac{e_1^2(2e_4 e_2 - 2e_5 e_1 + f_3) - A^2 - 2Ae_4 - f_4}{2} \tag{29}$$

from which we can express e_4 as follows

$$e_4 = \frac{e_5(2e_1^4 + 2A) + e_1(A^2 + f_4 - f_3 e_1^2)}{e_1(2e_1^2 e_2 - 2A) - 2e_5} \tag{30}$$

Denote

$$\begin{aligned} B &= e_1(2e_1^2 e_2 - 2A) \\ C &= 2e_1^4 + 2A \\ D &= e_1(A^2 + f_4 - f_3 e_1^2) \end{aligned}$$

Therefore,

$$e_4 = \frac{Ce_5 + D}{B - 2e_5} \tag{31}$$

Then, (26) becomes

$$e_3 = \frac{A + (Ce_5 + D)/(B - 2e_5)}{e_1} = \frac{A(B - 2e_5) + Ce_5 + D}{e_1(B - 2e_5)} \tag{32}$$

Combination of the above two equations and (25) gives us a quadratic equation in e_5. Notice that e_5^2 can be computed because $e_5^2 = y_1^2 y_2^2 y_3^2 y_4^2 y_5^2$.

$$e_5 e_3 = \frac{e_4^2 - f_4}{2} \Rightarrow e_5 \frac{A(B - 2e_5) + Ce_5 + D}{e_1(B - 2e_5)} = \frac{((Ce_5 + D)/(B - 2e_5))^2 - f_4}{2}$$

$$\Rightarrow e_5 = \frac{-C^2 e_1 e_5^2 - D^2 e_1 - 8ABe_5^2 + 2BCe_5^2 - 4De_5^2 + B^2 e_1 f_4 + 4e_1 f_4 e_5^2}{-2AB^2 - 8Ae_5^2 - 2BD + 4Ce_5^2 + 2CDe_1 + 4Be_1 f_4}$$

Let

$$\begin{aligned} E &= -C^2 e_1 e_5^2 - D^2 e_1 - 8ABe_5^2 + 2BCe_5^2 - 4De_5^2 + B^2 e_1 f_4 + 4e_1 f_4 e_5^2 \\ F &= -2AB^2 - 8Ae_5^2 - 2BD + 4Ce_5^2 + 2CDe_1 + 4Be_1 f_4 \end{aligned}$$

Then $e_5 = E/F$ and e_4 becomes

$$e_4 = \frac{CE + DF}{FD - 2E}$$

Denote $G = CE + DF$ and $H = FD - 2E$. From (26) we get

$$e_3 = \frac{AH + G}{He_1}$$

According to Theorem 1, for $n = 5$ we have

$$y_i = \frac{e_5 + y_i^2 e_3 + y_i^4 e_1}{e_4 + y_i^2 e_2 + y_i^4} \quad \text{for } i = 1, 2, 3, 4, 5$$

Combining all the above expressions for e_3, e_4 and e_5, we get the following equation for y_i

$$y_i = \frac{EHe_1 + Fy_i^2(AH + G + e_1^2 y_i^2 H)}{Fe_1(G + y_i^2 H(e_2 + y_i^2))} \tag{33}$$

Five-point compression algorithm requires $116M + 12S + I$ operations to extract all elliptic curve points. If each y_i is calculated by the given equation (33) separately the algorithm would require five inversions. However, there exists a method to obtain all y-coordinates using only one inversion operation. This method and computation of the number of operations are discussed in a greater detail in Appendix B.

5 Conclusion

The work done in this paper can be extended to six or higher number of points. However, the development of the algorithm for six points with the use of proposed lemma and theorem becomes harder but still possible. In this contribution, the generalization to n points was made by X. Fan, A. Ote-

missov, F. Sica, and A. Sidorenko in [3] but it is only of theoretical interest and the method suggested is not practical. There are still unanswered questions about the existence of practical algorithms which are computationally heavy in compression phase and light in decompression for one elliptic curve point when only one value is stored.

A Computational complexity: 4-point compression algorithm

Calculations are all based on two basic assumptions:

1) Addition and subtraction operations are neglected

2) Multiplication by constants are not counted

According to (21), in order to find all y_i's algorithm requires four inversions. Since inversions are computationally costly the number of inversions in the algorithm should be minimized. As it was mentioned before, the algorithm requires only one inversion.

Let

$$\begin{aligned} B &= e_1(A^2 + e_4^2 - e_1^2 f_3) \\ C &= 2e_1(e_1^2 e_2 - A) \\ D &= 2e_1^2 e_2 A - A^2 + e_4^2 - e_1^2 f_3 \\ E &= 2e_1^2(e_1^2 e_2 - A) \end{aligned}$$

Then,

$$y_i = \frac{B + C(e_2 y_i^2 + y_i^4)}{D + E y_i^2}$$

In order to avoid four inversions, express each y_i in the following way

$$y_1 = \frac{(B + C(a_2 y_1^2 + y_1^4))(D + E y_2^2)(D + E y_3^2)(D + E y_4^2)}{(D + E y_1^2)(D + E y_2^2)(D + E y_3^2)(D + E y_4^2)},$$

$$y_2 = \frac{(B + C(a_2 y_2^2 + y_2^4))(D + Ey_1^2)(D + Ey_3^2)(D + Ey_4^2)}{(D + Ey_1^2)(D + Ey_2^2)(D + Ey_3^2)(D + Ey_4^2)},$$

$$y_3 = \frac{(B + C(a_2 y_3^2 + y_3^4))(D + Ey_1^2)(D + Ey_2^2)(D + Ey_4^2)}{(D + Ey_1^2)(D + Ey_2^2)(D + Ey_3^2)(D + Ey_4^2)},$$

$$y_4 = \frac{(B + C(a_2 y_4^2 + y_4^4))(D + Ey_1^2)(D + Ey_2^2)(D + Ey_3^2)}{(D + Ey_1^2)(D + Ey_2^2)(D + Ey_3^2)(D + Ey_4^2)}.$$

The common denominator in all four expressions suggest that only one inversion is required. Calculations of multiplication, square and inversion operations are performed in the table below.

Expression	Operations	Comments
$y_1^2, y_2^2, y_3^2, y_4^2$	$4S + 4M$	Since $y^2 = x(x^2 + a) + b$
$f_2 = y_1^2y_2^2 + y_1^2y_3^2 + y_1^2y_4^2 + y_2^2y_3^2 + y_2^2y_4^2 + y_3^2y_4^2$	$6M$	Store $y_1^2y_2^2, y_1^2y_3^2, y_1^2y_4^2, y_2^2y_3^2$
$f_3 = y_1^2y_2^2y_3^2 + y_1^2y_2^2y_4^2 + y_1^2y_3^2y_4^2 + y_2^2y_3^2y_4^2$	$4M$	Store $y_1^2y_2^2y_3^2$
$e_4^2 = y_1^2y_2^2y_3^2y_4^2$	$1M$	Multiplication of $y_1^2y_2^2y_3^2$ by y_4^2
$y_1^4, y_2^4, y_3^4, y_4^4$	$4S$	y_i^2's are squared
e_1^2	1S	e_1 is squared
e_2	0	only subtraction, addition and multiplication by a constant

e_2^2	$1S$	
A	0	only subtraction, addition and multiplication by a constant
A^2	$1S$	
$B = e_1(A^2 + e_4^2 - e_1^2 f_3)$	$2M$	
$C = 2e_1(e_1^2 e_2 - A)$	$2M$	
$D = 2e_1^2 e_2 A - A^2 + e_4^2 - e_1^2 f_3$	$3M$	
$E = 2e_1^2(e_1^2 e_2 - A)$	$2M$	
$D + Ey_1^2$	$1M$	
$D + Ey_2^2$	$1M$	
$D + Ey_3^2$	$1M$	
$D + Ey_4^2$	$1M$	
$(D + Ey_1^2)(D + Ey_2^2)(D + Ey_3^2)(D + Ey_4^2)$	$3M$	
$((D + Ey_1^2)(D + Ey_2^2)(D + Ey_3^2)(D + Ey_4^2))^{-1}$	$1I$	
$B + C(e_2 y_1^2 + y_1^4)$	$2M$	
$B + C(e_2 y_2^2 + y_2^4)$	$2M$	
$B + C(e_2 y_3^2 + y_3^4)$	$2M$	
$B + C(e_2 y_4^2 + y_4^4)$	$2M$	
Final computation of y_1, y_2, y_3, y_4	$4M \cdot 4 = 16M$	

In total, the computational complexity of the algorithm is $55M + 11S + 1I$. For 256-bit elliptic curves the cost of square root operation and inversion, on average, are equal (Square root $\approx I \approx 127M + 254S$) [3]. Therefore,

compared to the ordinary method, four-point compression is 70% faster requiring 25% more memory. Compared to the trivial method, it saves 37.5% of memory.

B Computational complexity: 5-point compression algorithm

First, we describe the method to obtain y-coordinates with one inversion.

Recall equation (33)

$$y_i = \frac{EHe_1 + Fy_i^2(AH + G + e_1^2 y_i^2 H)}{Fe_1(G + y_i^2 H(e_2 + y_i^2))}$$

Rewrite the above as follows

$$y_i = \frac{EHe_1 + Fy_i^2(AH + G + e_1^2 y_i^2 H)}{Fe_1 \prod_{j=1}^{5}(G + y_j^2 H(e_2 + y_j^2))} \prod_{j \neq i}(G + y_j^2 H(e_2 + y_j^2)) \tag{34}$$

This form brings down all y_i's to a common denominator.

As mentioned before, the algorithm requires 116 multiplications, 12 square operations and 1 inversion. The counting process is summarised in the table below.

Expression	Cost	Comments
y_1^2, y_2^2, y_3^2, y_4^2, y_5^2	$5S + 5M$	Since $y^2 = x(x^2 + a) + b$
$f_2 = y_1^2y_2^2 + y_1^2y_3^2 + \cdots + y_4^2y_5^2$	$10M$	Store $y_1^2y_2^2$, $y_1^2y_3^2$, $y_2^2y_3^2$, $y_4^2y_5^2$
$f_3 = y_1^2y_2^2y_3^2 + \cdots + y_3^2y_4^2y_5^2$	$10M$	Store $y_1^2y_2^2y_3^2, y_1^2y_4^2y_5^2, y_2^2y_3^2y_4^2$
$f_4 = y_1^2y_2^2y_3^2y_4^2 + y_1^2y_2^2y_3^2y_5^2 + \cdots + y_2^2y_3^2y_4^2y_5^2$	$5M$	
$f_5 = e_5^2 = y_1^2y_2^2y_3^2y_4^2y_5^2$	M	
e_1^2	S	
e_2	0	only subtraction, addition and multiplication by a constant
e_2^2	S	
A	0	only subtraction, addition and multiplication by a constant
$B = 2e_1(e_1^2e_2 - A)$	$2M$	
e_1^4	S	
$C = 2e_1^4 + 2A$	0	
A^2	S	
$D = (A^2 + f_4 - f_3e_1^2) \cdot e_1$	$2M$	
B^2	S	
$F = A(-2B^2 - 8e_5^2) + e_1(2CD + 4Bf_4) + 4Ce_5^2 - 2BD$	$6M$	B^2,e_5^2 have been precomputed

$E = e_1(-C^2e_5^2 - D^2 + B^2f_4 + 4f_4e_5^2) + B(-8Ae_5^2 + 2Ce_5^2) - 4De_5^2$	8M+2S	B^2, e_5^2 have been precomputed
$H = FD - 2E$	M	
$G = CE + DF$	$2M$	
$He_1^2y_i^2$, $i = 1, 2, 3, 4, 5$	$2M \cdot 5 = 10M$	e_1^2, y_i^2 have been precomputed
Fy_i^2	$M \cdot 5 = 5M$	
$AH + G$	M	
EHe_1	$2M$	
$EHe_1 + Fy_i^2(AH + G + He_1^2y_i^2)$, $i = 1, 2, 3, 4, 5$	$M \cdot 5 = 5M$	
$G + y_i^2H(e_2 + y_i^2)$, $i = 1, 2, 3, 4, 5$	$2M \cdot 5 = 10M$	
Fe_1	M	
$Fe_1 \prod_{j=1}^{5}(G + y_j^2H(e_2 + y_j^2))$	$5M$	Common denominator
$(Fe_1 \prod_{j=1}^{5}(G + y_j^2H(e_2 + y_j^2)))^{-1}$	I	Inversion of the common denominator
Final computation of y_i, $i = 1, 2, 3, 4, 5$	$5M \cdot 5 = 25M$	See (34)

The total computational complexity of the algorithm is $116M + 12S + I$. This algorithm is faster by 72% but demands 20% more memory in contrast with the ordinary method. On the other hand, it is more computationally costly than the trivial method but saves 40% of memory.

References

[1] S. Blake-Wilson, N. Bolyard, V. Gupta, C. Hawk, and B. Moeller. Elliptic curve cryptography (ECC) cipher suites for transport layer security (TLS). *IETF Internet Draft*, May 2006. http://tools.ietf.org/html/rfc4492.

[2] M. Campagna and G. Zaverucha. A cryptographic suite for embedded systems (suite E). *IETF Internet Draft*, October 2012. http://tools.ietf.org/html/draft-campagna-suitee-04.

[3] X. Fan, A. Otemissov, F. Sica, and A. Sidorenko. Multiple compression on elliptic curves. preprint.

[4] M. Khabbazian, T. A. Gulliver, and V. K. Bhargava. Double point compression with applications to speeding up random point multiplication. *IEEE Trans. Computers*, 56(3): 305-313, 2007.

[5] A. Lenstra and E. Verheul, Selecting Cryptographic Key Sizes, *Journal of Cryptology* 14: 255-293, 2001. http://www.cryptosavvy.com/.

[6] T. Wollinger, J. Pelzl, C. Paar, G. Saldamli, and C. K. Koc. Elliptic and hyper-elliptic curves on embedded μp. *ACM Trans. on Embedded Computing Systems*, 3(3):509-533, August 2004.