DOI: 10.2478/awutm-2014-0014

DE GRUYTER OPEN

Analele Universităţii de Vest,
Timişoara
Seria Matematică – Informatică
LII, 2, (2014), 111– 125

Point Compression and Coordinate Recovery for Edwards Curves over Finite Field

Benjamin Justus

Abstract. We present two computational approaches for the purpose of point compression and decompression on Edwards curves over the finite field $\mathbb{F}_p$ where p is an odd prime. The proposed algorithms allow compression and decompression for the x or y affine coordinates. We also present a x-coordinate recovery algorithm that can be used at any stage of a differential addition chain during the scalar multiplication of a point on the Edwards curve.

AMS Subject Classification (2000). 94A60; 14H45
Keywords. Point Compression, IEEE Std 1363-2000, Elliptic Curve, Finite Field, Edwards Curve

1 Introduction

The first part of the present paper proposes compression and decompression algorithms for points on an Edwards curve over the finite field $\mathbb{F}_p$ where p is an odd prime. Edwards curves have played prominent roles in recent ECM (Elliptic Curve Method) applications. The most notable is the GMP-EECM software [5] based on which the authors claim to set speed records in ECM.

Point compression allows efficient storage and saves communication bandwidth in applications related to elliptic curve cryptography. The affine coordinate compression and decompression of a single point on a Weierstraß curve are described in the IEEE Standard **IEEE Std 1363-2000** for Public-Key Cryptography [4]. It specifies x-coordinate compression and decompression algorithms for binary fields $\mathbb{F}_{2^m}$, and y-coordinate compression and decompression algorithms for both binary field and $\mathbb{F}_p$ where p is an odd prime. Using cubic residue theorems, [2] is able to compress and decompress the x-coordinate of a single point on Weierstraß curves over $\mathbb{F}_p$ where $p > 3$. Double point compression and decompression algorithms for the y-coordinates are proposed in [3] for Weierstraß curves.

Technically, an Edwards curve is not an elliptic curve due to its singularities. However, every Edwards curve is birationally equivalent to an elliptic curve in Weierstraß form. Due to symmetry of the Edwards curve, our proposed compression and decompression algorithms work for both x-coordinates or y-coordinates. This becomes an advantage when one encounters a point with one coordinate size much larger than the other.

We present two approaches in this paper. The first method follows the standard approach as suggested in [4]. The compression of a point involves keeping a binary signature of the coordinate. The decompression stage is a little different in the sense: one needs to compute a square root and an inversion for completing the decompression process. This approach allows a natural generalization to n points compression, see section 3. When p is large with respect to n, one is capable of achieving a near 50% compression ratio. Although taking square roots in finite field can be achieved by polynomial time algorithms, our benchmark tests show that the square root operation in practice can be quite time consuming when the field size p is large. The second method avoids taking square roots at the cost of extracting the compressed coordinates from the curve equation and prescribed algebraic relations, see section 4. This method was pioneered in [3] for compressing the y-coordinates of points on Weierstraß curves. We present the analogies for Edwards curves in sections 4.1 and 4.2. Our benchmark tests show that the second method is much more efficient than the square root approach when p is large.

To speed up elliptic curve arithmetics, it is possible to add (or double) points omitting one of the coordinates. This is the case for Montgomery curves which are featured in the GMP-ECM software [8]. One can carry out points addition, doubling on Montgomery curves using only (X, Z) projective coordinates. The scalar multiplication of a point can be achieved by using a differential addition chain (sometimes known as Lucas chain). Following this approach, one is able to avoid completely the arithmetic cost incurred by one of the coordinates during all stages of a computation. Differential arithmetic formula also exists for the Edwards curve setting (using only (Y, Z) coordinates), see [1, 12].

In the second part of the paper, we investigate the following question: using a differential addition chain, how can one recover the missing coordinate at the final stage of a scalar multiplication computation? This question is investigated in [9] for Montgomery curves, and [1] for Edwards curves. Let $[n]P$ be the n-fold of a point P on a curve. In recovering the missing coordinate of $[n]P$, all the above approaches require partial coordinate information of the points $P, [n]P, [n+1]P$ in addition to the curve parameters. To obtain information of the point $[n+1]P$ during the computation of $[n]P$ is not practical as the computation of $[n+1]P$ does not come generally as a byproduct during any stage of computing $[n]P$.

In section 5, we present a novel x-coordinate recovering algorithm for Edwards curves. The algorithm is able to bypass information about $[n+1]P$ during the recovery of the missing x-coordinate of $[n]P$.

The plan of the paper is the following. We recall basic facts about Edwards curves in section 2. The two compression/decompression approaches are presented in section 3 and section 4, respectively. The probabilistic coordinate recovering algorithm is proved in section 5. Section 6 contains timing benchmarks for the decompression algorithms described in the previous sections.

1.1 Notation

We use the following notations in tabulating the cost incurred by the arithmetic operations. The notations $\mathbf{SQ}, \mathbf{S}, \mathbf{M}, \mathbf{I}$ signify the field operations: taking square root, squaring, multiplication, and inversion, respectively.

2 Background on Edwards Curves

Edwards curves are given by equations of the form

$$E_{c,d} : x^2 + y^2 = c^2(1 + dx^2y^2),$$

where c, d are curve parameters in a field k of characteristic different from 2, and $c, d \neq 0$ and $dc^4 \neq 1$.

The addition law is defined by the formula:

$$(x_1, y_1), (x_2, y_2) \mapsto \left(\frac{x_1y_2 + y_1x_2}{c(1 + dx_1x_2y_1y_2)}, \frac{y_1y_2 - x_1x_2}{c(1 - dx_1x_2y_1y_2)} \right). \tag{2.1}$$

For this addition law, the point $(0, c)$ is the neutral element. The inverse of a point $P = (x, y)$ is $-P = (-x, y)$. In particular, $(0, -c)$ has order 2; $(c, 0)$

and $(-c, 0)$ are the points of order 4. When the curve parameter d is not a square in k, then the addition law (2.1) is complete (i.e. defined for all inputs). Every Edwards curve is bi-rationally equivalent to an elliptic curve in Weierstraß form.

For the rest of the paper, we shall restrict our discussion to those curves with $c = 1$,

$$E_d = E_{1,d} : x^2 + y^2 = 1 + dx^2y^2. \tag{2.2}$$

All the algorithms in the paper work for both $E_{c,d}$ and E_d.

3 Square Root Approach

The algorithms 1 and 2 describe for n points compression and decompression. The compression/decompression is performed on the x-coordinates and can be adapted to the y-coordinates compression/decompression exactly the same way. For example when $n = 1$, to optimize the compression ratio, one should align the compression choice with the larger of the two coordinates. For a single point, our compression method follows the standard approach as described in [4]. The generalization to n points is quite natural, the idea is to concatenate each bit of binary signature to form a n-bits string. The compression algorithm 1 during the loop keeps the signature string S updated by adding the current bit of binary signature.

Algorithm 1 x-coordinate compression for n points

```
Input: $(x_1, y_1), ..., (x_n, y_n)$ on $E_d$
Output: $(S, y_1, y_2, ..., y_n)$ in compressed format
1: $S \leftarrow NULL$
2: for $i = 1$ to $n$ do
3:     $\overline{x} \leftarrow x_i \pmod 2$
4:     $S = S \,||\, \overline{x}$
5: end for
6: return $(S, y_1, y_2, ..., y_n)$
```

Algorithm 2 x-coordinate decompression for n points

```
Input: $(S, y_1, y_2, ..., y_n)$
Output: $(x_1, y_1), ..., (x_n, y_n)$ on $E_d$
1: for $i = 1$ to $n$ do
2:   $A \leftarrow \frac{1-y_i^2}{1-dy_i^2} \pmod p$
3:   $x_i \leftarrow \sqrt{A} \pmod p$
4:   if $x_i \pmod 2 \neq S[i]$ then
5:     $x_i = -x_i \pmod P$
6:   end if
7: end for
8: return $(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)$
```

Decompression Type	Max Storage Space	Arithmetic Cost
single point	$\lceil \log_2 p \rceil + 1$	1**SQ**+ 1**S**+ 1**M**+ 1**I**
n points	$n\lceil \log_2 p \rceil + n$	n**SQ**+ n**S**+ n**M**+ n **I**

Table 1: Cost for single and multiple points decompressions

The decompression algorithm 2 is correct because given the y-coordinate of a point (x, y) on the Edwards Curve E_d, we may solve for x^2 using the definition of the curve (2.2)

$$A = \frac{1 - y^2}{1 - dy^2}.$$

The field inversion is valid because the point is well defined. Taking square root of A in $\mathbb{F}_p$ gives rise to 2 solutions. Furthermore the 2 solutions are distinct because the field characteristic is odd. Thus the binary bit $\overline{x}$ uniquely identifies x.

The cost of the decompression algorithm is recorded in Table 1. These are the worst case scenarios which happen when n points given are distinct. Regarding the max storage space, one should keep in mind the max storage required for a single uncompressed point (x, y) (resp. n uncompressed points) is $2\lceil \log_2 p \rceil$ (resp. $2n\lceil \log_2 p \rceil$).

4 Second Approach

We present in this section a second approach. Section 4.1 describes compression and decompression algorithms for two points. Section 4.2 describes algorithms in the case of three points. They are analogous to the Weierstraß curve approaches as presented in [3].

4.1 Double Points Compression

The compression algorithm 3 takes two points on Edwards curve as input, and outputs a triple in the compressed format. There are two cases to consider. In the case $x_1 + x_2 = 0$ (line 1 - 3): we have $y_1 = \pm y_2$ because $y_1^2 = y_2^2$. So we need only to store the binary signature $\overline{y_2}$, and the decompression of y_2 is identical to the single point decompression case described in algorithm 2. Finally we recover x_2 as $x_2 = -x_1$.

In the other case: $x_1 + x_2 \neq 0$ (line 4 - 6), we store the triple (S, y_1, y_2). The decompression from the triple is described in algorithm 4. It relies on the arithmetic identity:

$$x_1 = \frac{x_1 + x_2}{2} + \frac{x_1^2 - x_2^2}{2(x_1 + x_2)} = \frac{S}{2} + \frac{A_1 - A_2}{2S} \tag{4.1}$$

where the parameters S, A_1, A_2 are as defined in algorithm 4. Finally, we can recover x_2 as $x_2 = -x_1$.

Algorithm 3 x-coordinate compression for two points

```
Input: $(x_1, y_1), (x_2, y_2)$ on $E_d$
Output: a triple in compressed format
1: if $x_1 + x_2 = 0 \pmod p$ then
2:     $S = y_2 \pmod 2$
3:     return $(S, x_1, y_1)$
4: else
5:     $S \leftarrow x_1 + x_2 \pmod p$
6:     return $(S, y_1, y_2)$
7: end if
```

Algorithm 4 x-coordinate decompression for two points assuming $x_1 + x_2 \neq 0$

Input: (S, y_1, y_2) in compressed format
Output: $(x_1, y_1), (x_2, y_2)$ on E_d

```
1: $A_1 \leftarrow \frac{1-y_1^2}{1-dy_1^2} \pmod p$
2: $A_2 \leftarrow \frac{1-y_2^2}{1-dy_2^2} \pmod p$
3: $x_1 = \frac{S}{2} + \frac{A_1 - A_2}{2S} \pmod p$
4: $x_2 = S - x_1 \pmod p$
5: return $(x_1, y_1), (x_2, y_2)$
```

Decompression Type	Max Storage Space	Arithmetic Cost
$x_1 + x_2 = 0$	$2\lceil \log_2 p \rceil + 1$	1**SQ**+ 1**M**+ 1**I**
$x_1 + x_2 \neq 0$	$3\lceil \log_2 p \rceil$	2**S**+ 3**M**+ 3**I**

Table 2: Cost for Double Points Decompression

4.2 Triple Points Compression

The compression algorithm 5 takes three points on an Edwards curve as input, and outputs a point in the compressed format. There are three cases to consider. If $x_1^2 = x_2^2 = x_3^2$ (line 1 - 4): we have without loss of generality $x_2 = x_3 = \pm x_1$. So the recovery of x_2, x_3 as before requires only the binary signatures $\overline{x_2}, \overline{x_3}$ in addition to their respective y-coordinates. If only two of x_i's satisfy $x_i^2 = x_j^2$ (line 5 - 16), the recovery of x_j requires only the binary signature $\overline{x_j}$, and we are reduced to the case of double points compression (algorithm 3).

Algorithm 5 x-coordinate compression for three points

```
Input: $(x_1, y_1), (x_2, y_2), (x_3, y_3)$ on $E_d$
Output: a point in compressed format
1: if $x_1^2 = x_2^2 = x_3^2$ then
2:     $\overline{x_2} = x_2 \pmod 2$
3:     $\overline{x_3} = x_3 \pmod 2$
4:     return $(x_1, y_1, y_2, y_3, \overline{x_2}, \overline{x_3})$
5: else if $x_1^2 = x_2^2$, $x_2^2 \neq x_3^2$ then
6:     $\overline{x_1} = x_1 \pmod 2$
7:     $A \leftarrow x_2 + x_3$
8:     return $(A, y_1, y_2, y_3, \overline{x_1})$
9: else if $x_1^2 = x_3^2$, $x_1^2 \neq x_2^2$ then
10:    $\overline{x_3} = x_3 \pmod 2$
11:    $A \leftarrow x_1 + x_2$
12:    return $(A, y_1, y_2, y_3, \overline{x_3})$
13: else if $x_2^2 = x_3^2$, $x_1^2 \neq x_3^2$ then
14:    $\overline{x_2} = x_2 \pmod 2$
15:    $A \leftarrow x_1 + x_3$
16:    return $(A, y_1, y_2, y_3, \overline{x_2})$
17: else if $x_1^2 \neq x_2^2 \neq x_3^2$ and $x_1 + x_2 + x_3 \neq 0$ then
18:    $A \leftarrow x_1 + x_2 + x_3$
19:    return $(A, y_1, y_2, y_3)$
20: else
21:    $A \leftarrow x_2 + x_3$
22:    return $(A, y_2, y_3)$
23: end if
```

In the last case $x_1^2 \neq x_2^2 \neq x_3^2$ (line 17 - 22), there are two sub-cases. When $x_1 + x_2 + x_3 \neq 0$, we may define $A = x_1 + x_2 + x_3$ and returns (A, y_1, y_2, y_3) as an output of the compression algorithm. To decompress (algorithm 6), we first calculate B_1, B_2, B_3 (Alg 6: line 1 - 3). The x_1, x_2, x_3 (Alg 6: line 4 - 6) are well defined because

$$B_i = A^2 + x_i^2 - \sum_{j \neq i} x_j^2 = 2 \prod_{j \neq i} (x_i + x_j) \neq 0.$$

The correctness of the identities (Alg 6: line 4 -6) can be verified straightforward as:

$$\begin{aligned} x_i &= \left(2\prod_{j\neq i} x_j\right)^2 - 4\prod_{j\neq i} x_j^2 + x_i \\ &= (A^2 - \sum_j x_j^2 - 2(Ax_i - x_i^2))^2 - 4\prod_{j\neq i} x_j^2 + x_i \\ &= (B_i - 2Ax_i)^2 - 4\prod_{j\neq i} x_j^2 + x_i \\ &= B_i^2 - 4AB_i x_i + 4A^2x_i^2 - 4\prod_{j\neq i} x_j^2 + x_i \end{aligned}$$

which implies

$$x_i = \frac{B_i^2 + 4A^2x_i^2 - 4\prod_{j\neq i} x_j^2}{4AB_i}, \quad i = 1, 2, 3.$$

Algorithm 6 x-coordinate decompression for three points assuming $x_1^2 \neq x_2^2 \neq x_3^2, x_1 + x_2 + x_3 \neq 0$

```
Input: $(A, y_1, y_2, y_3)$
Output: $(x_1, y_1), (x_2, y_2), (x_3, y_3)$ on $E_d$
1: $B_1 = A^2 + x_1^2 - x_2^2 - x_3^2$
2: $B_2 = A^2 + x_2^2 - x_1^2 - x_3^2$
3: $B_3 = A^2 + x_3^2 - x_1^2 - x_2^2$
4: $x_1 = \frac{B_1^2+4A^2x_1^2-4x_2^2x_3^2}{4AB_1}$
5: $x_2 = \frac{B_2^2+4A^2x_2^2-4x_1^2x_3^2}{4AB_2}$
6: $x_3 = \frac{B_3^2+4A^2x_3^2-4x_1^2x_2^2}{4AB_3}$
7: return $(x_1, y_1), (x_2, y_2), (x_3, y_3)$
```

This finishes the subcase $x_1 + x_2 + x_3 \neq 0$. In the subcase $x_1 + x_2 + x_3 = 0$ (line 20 - 22): we have $x_1 = -(x_2 + x_3)$. So we are again reduced to the double points case: we can extract (algorithm 4) $(x_2, y_2), (x_3, y_3)$ from (A, y_2, y_3). Finally we recover x_1 as $x_1 = -(x_2 + x_3)$.

5 Point Recovery

In this section, we present the x-coordinate recovery algorithm for Edwards curves. Without the aid of any algorithm, given the y-coordinate of a point

Decompression Type	Max Storage Space	Arithmetic Cost
$x_1^2 = x_2^2 = x_3^2$	$4\lceil \log_2 p \rceil + 2$	**2SQ**+ 2**M**+ 2**I**
$x_1^2 = x_2^2$, $x_2^2 \neq x_3^2$	$4\lceil \log_2 p \rceil + 1$	1**SQ**+ 2**S**+ 4**M**+ 4**I**
$x_1^2 = x_3^2$, $x_1^2 \neq x_2^2$	$4\lceil \log_2 p \rceil + 1$	1**SQ**+ 2**S**+ 4**M**+ 4**I**
$x_2^2 = x_3^2$, $x_1^2 \neq x_3^2$	$4\lceil \log_2 p \rceil + 1$	1**SQ**+ 2**S**+ 4**M**+ 4**I**
$x_1^2 \neq x_2^2 \neq x_3^2, x_1 + x_2 + x_3 \neq 0$	$4\lceil \log_2 p \rceil$	7**S**+ 9**M**+ 3**I**
$x_1^2 \neq x_2^2 \neq x_3^2, x_1 + x_2 + x_3 = 0$	$4\lceil \log_2 p \rceil$	2**S**+ 3**M**+ 3**I**

Table 3: Cost for three points Decompression

on an Edwards curve, one is able to determine the x-coordinate up to a sign (i.e. 50% chance of guessing it right). We first prove in the following:

Proposition 5.1. *Let E_d be an Edwards curve defined over $\mathbb{F}_p$ where p is odd prime. Suppose $d \neq 0, 1$, and d is not a square in $\mathbb{F}_p$. Let $P = (x, y)$ be a point whose order does not divide 4. Let x_n, y_n be the affine coordinates of the points $[n]P$. Let $A = 1 - dy^2$, $B = y^2 - 1$, and $D = \gcd(Ay_n^2 + B, dBy_n^2 + A)$, then we have*

$$dxyx_ny_n \equiv -1 \pmod{(dBy_n^2 + A)/D}. \tag{5.1}$$

Proof. We derive the above proposition from Theorem 2 of [1] (specialized to the case $c = 1$). The theorem allows one to express the affine x-coordinate of $[n]P$ in terms of the coordinates of the points $P, [n]P, [n+1]P$. Precisely, we have

$$x_n = \frac{2yy_ny_{n+1}(dBy_n^2 + A) - (Ay_n^2 + B) - y_{n+1}^2(dBy_n^2 + A)}{dxyy_n\left(Ay_n^2 + B - y_{n+1}^2(dBy_n^2 + A)\right)} \tag{5.2}$$

where A, B, y_n are as defined in the proposition, and y_{n+1} is the affine y-coordinates of the point $[n+1]P$. We may reduce (5.2) modulo $dBy_n^2 + A$ to obtain

$$dxyx_ny_n(Ay_n^2 + B) = -(Ay_n^2 + B) \pmod{dBy_n^2 + A}. \tag{5.3}$$

Dividing away $D = \gcd(Ay_n^2 + B, dBy_n^2 + A)$ in (5.3) gives

$$dxyx_ny_n\frac{Ay_n^2 + B}{D} = -\frac{Ay_n^2 + B}{D} \pmod{\frac{dBy_n^2 + A}{D}}.$$

And since $\gcd((Ay_n^2 + B)/D, (dBy_n^2 + A)/D) = 1$, we have

$$dxyx_ny_n \equiv -1 \pmod{(dBy_n^2 + A)/D}.$$

□

Equation 5.1 gives a necessary condition for x_n to be the x-coordinate of $[n]P$. And this gives us an alternative way of distinguishing the sign of x_n. Algorithm 7 describes the proposed x-coordinate recovery algorithm. The input of the algorithm is a point $P = (x, y)$ and the y-coordinate of the point $[n]P$ on the Edwards curve E_d. The algorithm outputs the x-coordinate of $[n]P$.

Algorithm 7 x-coordinate recovery

```
Input: $P = (x, y)$, $y_n$ = $y$-coordinates of $[n]P$ on $E_d$
Output: $x_n$ = $x$-coordinates of $[n]P$ if successful.
1: $A \leftarrow 1 - dy^2 \pmod p$
2: $B \leftarrow y^2 - 1 \pmod p$
3: $D = \gcd(Ay_n^2 + B, dBy_n^2 + A)$
4: $C \leftarrow \frac{1-y_n^2}{1-dy_n^2} \pmod p$
5: $x_n \leftarrow \sqrt{C} \pmod p$
6: if $dxyx_ny_n \neq -1 \pmod{(dBy_n^2 + A)/D}$ then
7:     $x_n = -x_n \pmod p$
8: end if
9: return $x_n$
```

6 Benchmark

We perform timing benchmark for the algorithms described in the previous sections. Specifically in the following, Table 4 and 5 contain benchmark results on the single point compression/decompression algorithm (algorithm 1 and 2 for $n = 1$), and Table 6 and 7 contain benchmark results on the double points compression/decompression algorithm (algorithm 3 and 4). Table 8 and 9 contain benchmark results on the triple points compression/decompression algorithm (algorithm 5 and 6). Negligible computation time[1] are registered as NEG in all the tables. The decompression algorithms are timed on batches as well as per decompression operation.

For the benchmark test, we first randomly generate a prime p of a specified bit-length. For a fixed E_d over $\mathbb{F}_p$, random points on E_d are generated upon which the compression and decompression algorithms are benchmarked. To see the impact of the curve parameter d on the benchmark results, a set of d different in size is used. The timing is performed by repeating the

[1]The CPU clock registers 0 sec time lapse

compression/decompression a fixed number of times. The current set of tests are run on a DELL Precision M6600 Laptop with OS Ubuntu 13.04 32 bits, Intel Core i7-2860QM CPU @ 2.50GHz $\times$ 4. The test suites are written using the GMP library [6].

The test results show that for both approaches, the compression algorithms require negligible CPU time compared to the decompression algorithms. Furthermore, the complexity of the compression/decompression algorithms directly depends on the size of the base field, and the curve parameter d.

The choices of the field size though are much larger than those used in ECC, we have included the test results here as a matter of reference. A full test for small devices using the standard curve sizes (192 - 521 bits) is planned in the near future.

7 Future Work

An important question here is about optimization: how one may incorporate the compression of points in an optimized way into a particular addition chain. Lastly, we plan to test the algorithms in this paper on devices with limited memory and constrained computation power.

Acknowledgements

The author wishes to thank the referees, whose comments helped to improve this paper. Particular thanks go to Dr. Daniel Loebenberger who provided an improvement to the original version of the coordinate recovery algorithm.

Appendix A. Timing Benchmark

$\mathbb{F}_p$	Time (sec)
512	NEG
1024	NEG
2048	NEG
4096	NEG

(a) Compression single point 1000 times using E_d with $d = 3$

$\mathbb{F}_p$	Time (sec)	Time/Decomp. (sec)
512	0.348	0.0003
1024	10.02	0.010
2048	23.34	0.023
4096	128.97	0.129

(b) Decompression single point 1000 times using E_d with $d = 3$

Table 4: Benchmark for Algorithm 1 and 2

$\mathbb{F}_p$	Time (sec)
512	NEG
1024	0.004
2048	NEG
4096	NEG

(a) Compression single point 1000 times using E_d with $d = p - 1$

$\mathbb{F}_p$	Time (sec)	Time/Decomp. (sec)
512	0.348	0.0003
1024	9.904	0.0099
2048	23.472	0.023
4096	129.796	0.130

(b) Decompression single point 1000 times using E_d with $d = p - 1$

Table 5: Benchmark for Algorithm 1 and 2

$\mathbb{F}_p$	Time (sec)
512	0.028
1024	0.032
2048	0.044
4096	0.068

(a) Compression two points 50000 times using E_d with $d = 3$

$\mathbb{F}_p$	Time (sec)	Time/Decomp. (sec)
512	1.132	0.001
1024	3.748	0.004
2048	10.536	0.011
4096	30.16	0.030

(b) Decompression two points 50000 times using E_d with $d = 3$

Table 6: Benchmark for Algorithm 3 and 4

$\mathbb{F}_p$	Time (sec)
512	0.024
1024	0.032
2048	0.044
4096	0.068

(a) Compression two points 50000 times using E_d with $d = p - 1$

$\mathbb{F}_p$	Time (sec)	Time/Decomp. (sec)
512	1.188	0.001
1024	4.244	0.004
2048	11.492	0.011
4096	33.424	0.033

(b) Decompression two points 50000 times using E_d with $d = p - 1$

Table 7: Benchmark for Algorithm 3 and 4

$\mathbb{F}_p$	Time (sec)
512	0.036
1024	0.040
2048	0.056
4096	0.088

(a) Compression three points 50000 times using E_d with $d = 3$

$\mathbb{F}_p$	Time (sec)	Time/Decomp. (sec)
512	3.162	0.003
1024	10.652	0.011
2048	27.000	0.027
4096	74.936	0.075

(b) Decompression three points 50000 times using E_d with $d = 3$

Table 8: Benchmark for Algorithm 5 and 6

$\mathbb{F}_p$	Time (sec)
512	0.036
1024	0.044
2048	0.06
4096	0.084

(a) Compression three points 50000 times using E_d with $d = p - 1$

$\mathbb{F}_p$	Time (sec)	Time/Decomp. (sec)
512	3.732	0.004
1024	11.384	0.011
2048	28.584	0.029
4096	79.776	0.080

(b) Decompression two points 50000 times using E_d with $d = p - 1$

Table 9: Benchmark for Algorithm 5 and 6

References

[1] **Benjamin Justus and Daniel Loebenberger**, *Differential Addition in Generalized Edwards Coordinates*, IWSEC, 2010, 316-325

[2] **Alina Dudeanu and George-Razvan Oancea and Sorin Iftene**, *An x-Coordinate Point Compression Method for Elliptic Curves over F_p*, SYNASC, 2010, 65-71

[3] **Majid Khabbazian and T. Aaron Gulliver and Vijay K. Bhargava**, Double

Point Compression with Applications to Speeding Up Random Point Multiplication, *IEEE Trans. Computers*, **56** (3), (2007), 305-313

[4] *IEEE 1363-2000: Standard Specification For Public Key Cryptorgraphy*, American National Standards Institute, IEEE Computer Society, 2000.

[5] **Daniel J. Bernstein and Peter Birkner and Tanja Lange and Christiane Peters**, ECM using Edwards curves, *Math. Comput.*, **82** (282), (2013)

[6] **Torbjörn Granlund and the GMP development team**, *GMP-5.0.5*, 2012.

[7] **Daniel J. Bernstein and Tien-Ren Chen and Chen-Mou Cheng and Tanja Lange and Bo-Yin Yang**, *ECM on Graphics Cards*, EUROCRYPT, 2009, 483-501

[8] **Paul Zimmermann and Bruce Dodson**, *20 Years of ECM*, ANTS, 2006, 525-542

[9] **Katsuyuki Okeya and Kouichi Sakurai**, *Efficient Elliptic Curve Cryptosystems from a Scalar Multiplication Algorithm with Recovery of the y-Coordinate on a Montgomery-Form Elliptic Curve*, CHES, Vol. Generators, 2001, 126-141

[10] **Eric Brier and Marc Joye**, *Weierstraß Elliptic Curves and Side-Channel Attacks*, Public Key Cryptography, 2002, 335-345

[11] **Eric Bach and Jeffrey Shallit**, *Algorithmic Number Theory*, MIT Press, **1**, 1996

[12] **Wouter Castryck and Steven D. Galbraith and Reza Rezaeian Farashahi**, Efficient arithmetic on elliptic curves using a mixed Edwards-Montgomery representation, *IACR Cryptology ePrint Archive*, **2008**, (2008), 218

Benjamin Justus

Laboratoire PRiSM, Groupe Crypto, UVSQ, 78035 Versailles Cedex, FRANCE
Laboratory for Safe and Secure Systems LaS^3, Standorte Amberg/Regensburg, GERMANY

Received: 25.06.2014

Accepted: 10.12.2014