Implementation of Elliptic Curve Cryptography in Binary Field

Implementation of Elliptic Curve Cryptography in Binary Field

D R Susantio, I Muchtadi-Alamsyah

Algebra Research Group, Faculty of Mathematics and Natural Sciences, Institut Teknologi Bandung, Jalan Ganesha no. 10, Bandung, 40132, Indonesia.

E-mail: ntan@math.itb.ac.id

Abstract. Currently, there is a steadily increasing demand of information security, caused by a surge in information flow. There are many ways to create a secure information channel, one of which is to use cryptography. In this paper, we discuss the implementation of elliptic curves over the binary field for cryptography. We use the simplified version of the ECIES (Elliptic Curve Integrated Encryption Scheme). The ECIES encrypts a plaintext by masking the original message using specified points on the curve. The encryption process is done by separating the plaintext into blocks. Each block is then separately encrypted using the encryption scheme.

1. Introduction

Information security has become one of the oft-discussed problem in the wake of the 21st century. Most information channels that we use are insecure: they are vulnerable to leakage. Therefore, it is essential that we look for ways to secure the insecure information channel. Cryptography is one of the many ways to secure the channels.

Neal Koblitz and Victor Miller independently introduced elliptic curve cryptography (ECC) [5]. ECC uses the set of points on an elliptic curve along with an addition rule. The unique mathematical structure of the points with the addition rule enables us to perform encryption and decryption of plaintexts. Another reason that supports the feasibility of ECC is the fact that it uses significantly smaller key sizes than the RSA Cryptosystem. For a comparison between RSA and ECC key sizes, see [6].

In the previous research [3], we presented the implementation of binary field arithmetic operation algorithms. In this paper, we discuss the implementation of elliptic curve cryptography using elliptic curves over binary field. We will address several issues, among them will be case-handling for the point operations, the key generation process, and the encryption.

2. Elliptic Curves and the ECIES

An elliptic curve over $GF(2^n)$ is defined by the simplified Weierstrass equation $y^2 + xy = x^3 + ax^2 + b$, where $a \neq 0$ and $b \neq 0$ [2]. It is possible to define several operations on the points of the elliptic curve, namely point negation, addition, and doubling.

We define the point operations as follows. First, let E be an elliptic curve over $GF(2^n)$ and $P(x, y)$ a point on E. The negative of P is defined to be $-P(x, x + y)$. Now let $P(x_1, y_1)$ and $Q(x_2, y_2)$ be distinct points on E. The result of adding P and Q is the point $P + Q$ (x_3, y_3) where $x_3 = \lambda^2 + \lambda + x_1 + x_2 + a$, $y_3 = \lambda(x_1 + x_3) + x_3 + y_1$, and $\lambda = (y_2 + y_1)/(x_2 + x_1)$. In

the case that P and Q are not distinct, the operation is called a point doubling. The double of point P is the point $2P$ (x_3, y_3) where $x_3 = \lambda^2 + \lambda + a$, $y_3 = x_1^2 + \lambda x_3 + x_3$, and $\lambda = x_1 + y_1/x_1$. It is also possible to define a scalar multiplication on elliptic curves. Given a positive integer n the scalar multiple of a point is defined as

$$nP = \underbrace{P + P + P + \cdots + P}_{n \text{ times}}.$$

The set of all points of an elliptic curve E over $GF(2^n)$ forms a commutative group with respect to point addition. Furthermore, according to the Mordell-Weil Theorem, this group is finitely generated [7] [8] . The group structure of the elliptic curves over $GF(2^n)$ allows its use for cryptography. In this paper, we discuss the implementation of the simplified version of Elliptic Curve Integrated Encryption Scheme (ECIES).

The simplified ECIES is a modification of the ElGamal scheme, which is based on the elliptic curve discrete logarithm problem described as follows. [9]

Definition 2.1. *Let E be an elliptic curve over $GF(2^n)$, P be a point on E with order n, and $Q \in \langle P \rangle$, the finite group generated by P. The discrete logarithm problem is the problem of determining an integer m, $0 \leq m \leq n-1$ which satisfies $Q = mP$. The integer m is then called the discrete logarithm of Q.*

Let $\mathcal{P}$ the set of all possible plaintexts, $\mathcal{C}$ the set of all possible ciphertexts, and $\mathcal{K}$ the set of all keys in the encryption scheme. We now define the simplified ECIES as follows. [9]

Definition 2.2. *Let E be an elliptic curve over $GF(2^N)$ such that E contains a cyclic subgroup $H = \langle P \rangle$ of order n in which the elliptic curve discrete logarithm problem is infeasible.*

(i) The plaintext space is $\mathcal{P} = GF(2^N)$ and the ciphertext space is $\mathcal{C} = E(GF(2^N)) \times GF(2^N)$.

(ii) The key space is $\mathcal{K} = \{(E, P, Q, m, n) : Q = mP\}$, where E, P, Q, and n are public keys and m a private key.

(iii) For every $K \in \mathcal{K}$, plaintext $x \in \mathcal{P}$, and ciphertext $(U, c) \in \mathcal{C}$, and a (secret) random number $k \in [0, n-1]$, define the encryption and decryption as follows.

(a) The encryption function is $e_K(x, k) = (kP, x \cdot x_0)$, where $kQ = (x_0, y_0)$, and $x_0 \neq 0$.

(b) The decryption function is $d_K(U, c) = c(x_0)^{-1}$, where $(x_0, y_0) = mU$.

3. Implementation

In this section we will describe several algorithms to perform elliptic curve operations, as well as some modifications made to the simplified ECIES.

Elliptic Curve Operations

We first describe the algorithm for point negation. Given a point $P(x, y)$ on an elliptic curve E, the algorithm simply takes the coordinates of P as input, then returns $-P(x, x + y)$ (see Algorithm 1).

Algorithm 1 Point negation

INPUT: point $P(x_1, y_1)$.
OUTPUT: point $-P$.
1: $X \leftarrow x_1$
2: $Y \leftarrow x_1 \oplus y_1$
3: **return** (X, Y)

Next, we describe the algorithm for point doubling. We first consider the case when $P = -P$ and P is the identity. In this case, the algorithm returns the identity. Otherwise, we compute $2P$ using the doubling formula (see Algorithm 2).

Algorithm 2 Point doubling

INPUT: point $P(x_1, y_1)$.
OUTPUT: point $2P$.
if $P = -P$ or $P = \mathcal{O}$ **then**
  return $\mathcal{O}$
else
  $\lambda \leftarrow x_1 \oplus y_1/x_1$
  $X \leftarrow \lambda^2 \oplus \lambda \oplus a$
  $Y \leftarrow x_1^2 \oplus \lambda \cdot X \oplus X$
  return (X, Y)
end if

We now describe the point addition algorithm. We first consider whether $P = Q$. If this is the case, we utilize the point doubling algorithm. Otherwise, we have several sub-cases to consider. The first sub-case is the case when $P = -Q$. In this case, return $\mathcal{O}$. The second sub-case is the case when $P = \mathcal{O}$ or $Q = \mathcal{O}$. If $P = \mathcal{O}$, return Q. If $Q = \mathcal{O}$, return P. Should all of these sub-cases fail, we conclude that P and Q are distinct and are not the identity. Therefore, we compute $P + Q$ using the addition formula.

Algorithm 3 Point addition

INPUT: points $P(x_1, y_1)$, $Q(x_2, y_2)$.
OUTPUT: point $P + Q$.
if $P \neq Q$ **then**
  if $P = -Q$ **then**
    return $\mathcal{O}$
  else if $P = \mathcal{O}$ **then**
    return Q
  else if $Q = \mathcal{O}$ **then**
    return P
  else
    $\lambda \leftarrow (y_2 \oplus y_1)/(x_2 \oplus x_1)$
    $X \leftarrow \lambda^2 \oplus \lambda \oplus x_1 \oplus x_2 \oplus a$
    $Y \leftarrow \lambda \cdot (x_1 \oplus X) \oplus X \oplus y_1$
    return (X, Y)
  end if
else
  return $2P$
end if

Finally, to calculate the scalar multiple of a point, we use the double-and-add method. This method is analogous to the square-and-multiply algorithm for calculating powers. Let P be a point on an elliptic curve and let s be a positive integer. First, let $b_n b_{n-1} \cdots b_0$ be the binary representation of s. We observe the value of b_0. If $b_0 = 1$, we begin with an initial value of P. Otherwise, we begin with $\mathcal{O}$. Then, for each integer i, where $1 \leq i \leq n$, observe b_i. If $b_i = 1$, then add $(2^i)P$. For example, suppose we want to compute $5P$. The binary representation of 5 is 101. Therefore, using this method, instead of calculating $P + P + P + P + P$, we calculate $4P + P$. The double-and-add method is more efficient than the naive repeated addition method to compute scalar multiples [9].

ECIES Parameter Generation
To begin the encryption and decryption process using the simplified ECIES, we first generate the curve. This is simply done by choosing a random number between 1 and $2^N - 1$, where N is the degree of the chosen binary field. We then take the binary representation of the random number as the curve parameters a and b.

Next, we generate the point P. We first randomly choose the x-coordinate for the point using a similar procedure to the curve parameter generation. Substituting for x in the curve equation yields an equation in the form $y^2 + by + d = 0$. We then determine whether the equation has a solution using Theorem 3.1 [4].

Algorithm 4 Scalar multiple

INPUT: point P, integer n.
OUTPUT: point nP.
1: $A \leftarrow P$
2: $R \leftarrow \mathcal{O}$
3: **while** $n > 0$ **do**
4: **if** $n \equiv 1 \bmod 2$ **then**
5: $R \leftarrow R + A$
6: **end if**
7: $n \leftarrow n \gg 1$
8: $A \leftarrow 2A$
9: **end while**
10: **return** R

Theorem 3.1. *The polynomial* $t^2 + t + d \in GF\left(2^N\right)[t]$ *has a root* u *in* $GF\left(2^N\right)$*, if and only if* $\mathrm{Tr}(d) = 0$. $\mathrm{Tr}(d)$ *is called the trace of an element* d*, which is defined to be*

$$\mathrm{Tr}(d) = d + d^2 + d^{2^2} + \cdots + d^{2^{N-1}}.$$

If the equation has a root, we are then able to calculate the roots using Theorem 3.2 [1]. Otherwise, select another x-coordinate and repeat the equation checking process.

Theorem 3.2. *(Kugurakov) Let* $\delta \in GF\left(2^N\right)$ *and* $\mathrm{Tr}(\delta) = 0$. *Then the equation* $y^2 + y = \delta$ *has the explicit root* $y = 1 + \sum_{j=1}^{m-1}\left(\delta^{2^j}\left(\sum_{k=0}^{j-1} u^{2^k}\right)\right)$ *where* $u \in GF\left(2^N\right)$ *and* $\mathrm{Tr}(u) = 1$. *The other root is* $y + 1$.

To choose the element u such that $\mathrm{Tr}(u) = 1$, we select a random element of $GF\left(2^N\right)$, then calculate its trace. This process is repeated until an element with trace 1 is found.

The next step is to calculate the order of P. We use Hasse's theorem [8].

Theorem 3.3. *(Hasse) Let* $E/\mathbb{F}_q$ *be an elliptic curve over* $GF(q)$. *Then,*

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

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

As a consequence of this theorem, we have $(\sqrt{q} - 1)^2 \leq \#E(\mathbb{F}_q) \leq (\sqrt{q} + 1)^2$. Therefore, there exists an integer $M \in [(\sqrt{q} - 1)^2, (\sqrt{q} + 1)^2]$ such that $MP = \mathcal{O}$. We can then compute n, the order of P, as the factor of M.

Finally, we can generate the private key m and point $Q = mP$. The private key m is randomly chosen from integers between 1 and $n - 1$, inclusive. Since, by Definition 2.2, the point P generates a cyclic group of order n, it suffices to choose m in this manner. After the private key is determined, we compute $Q = mP$ using the double-and-add algorithm.

Encryption and Decryption

Encryption is accomplished by first converting the plaintext into an element, or elements, of $GF(2^N)$. We take the binary representation of the ASCII code of each character in the plaintext. In this paper, it is assumed that every character has an ASCII value of 0 – 127.

The next step is to group the characters, which are already in their binary representation, into blocks. To do this, first take note that due to the assumption of the ASCII value, the size of the ASCII binary representation is at most 7 bits. We then do a padding process so that every character has a 7-bit binary representation. Once padded, the binary representations are concatenated, then split into blocks where each block has a maximum of $\lfloor N/7 \rfloor$ characters of the original plaintext.

The last step is to encrypt each block of the plaintext using the encryption function defined before.

We provide the proof of correctness for the encryption algorithm (Algorithm 5) as follows.

Algorithm 5 Encryption with the simplified ECIES

INPUT: plaintext x.
OUTPUT: ciphertext $(U(x_1, y_1), y)$.
1: **for** $char \in x$ **do**
2: $char \leftarrow$ BINARYASCII($char$)
3: $char \leftarrow$ PADDING($char$)
4: APPEND(x',$char$)
5: **end for**
6: $blocklength \leftarrow \lfloor N/7 \rfloor$
7: $x' \leftarrow$ BLOCK(x', $blocklength$)
8: $k \leftarrow$ RANDOM($[1, n-1]$)
9: $U(x_1, y_1) \leftarrow kP$
10: $V(x_2, y_2) \leftarrow kQ$
11: **for** $char \in x'$ **do**
12: $cipher \leftarrow char \cdot x_2$
13: APPEND(y, $cipher$)
14: **end for**
15: **return** (U, y)

Theorem 3.4. *The encryption algorithm with the simplified ECIES is correct.*

Proof. We first split the algorithm as described into three blocks: lines 1 – 5, 6 – 10, and 11 – 15. We will establish the correctness of each block.

Lines 1 – 5. Let x be a string and x' be an array of binary strings of length 7. For the purposes of the algorithm, we allow x' to be empty. We also note that a string can be viewed as an array of characters. We assert that x' is a loop invariant. Note that at the beginning of the loop, x' is an empty array, which we allow. At each iteration, the algorithm will convert the nth character of x into its ASCII value in binary form by the BINARYASCII function. Its binary form is then padded by the PADDING function. Therefore, the character is now a binary string of length 7. This binary string is then appended to x', which establishes the fact that by the end of the execution of the loop, x' is an array of binary strings of length 7, verifying our assertion that x' is indeed a loop invariant. Finally, since we have a 'for' loop, and the length of x is finite, the loop is guaranteed to terminate. We have established the correctness of this block.

Lines 6 – 10. For this block, let x', the array of binary strings of length 7 and the points P and Q, the public keys, be our initial assertions. The final assertion will be x', the array of binary strings of length at most N, and points kP and kQ. The algorithm on this block first calculates the block length, which is the maximum of characters of the original plaintext that can be concatenated by the BLOCK function. Next, the BLOCK function manipulates x' by replacing each of the elements with blocks of concatenated binary representations of at most $\lfloor N/7 \rfloor$ characters from the original plaintext. Since $\lfloor N/7 \rfloor \leq N/7$, each block can only have a binary string of length at most N. The algorithm then randomly computes an integer within the $[1, n-1]$ interval. Finally, by the point scalar multiple algorithm, we obtain kP and kQ. We have completed the proof of correctness for this block.

Lines 11 – 15. Let x' be an array of blocks as previously described, U be kP, the first part of the ECIES ciphertext, and y be an array of ciphertexts which make up the second part of the ECIES ciphertext, which we allow to be empty. We first consider the 'for' loop. As with the first block, we assert that y is a loop invariant. We note the fact that y is empty at the beginning of the loop. At the execution of the loop, each element of the array is multiplied with x_2, the x-coordinate of kQ. It is then appended to y. Therefore, at the end of the execution of the loop, y is indeed an array of ciphertexts. Again, since we have a 'for' loop, the loop is guaranteed to terminate. Finally, the algorithm returns (U, y), which is the complete form of an ECIES ciphertext, establishing the correctness of this block.

Since we have established the correctness of the blocks, we have also established the correctness of the encryption algorithm. □

The decryption process is essentially the reverse of the encryption process. We first decrypt

each block of the ciphertext using the decryption function. Then, we make sure that the length of each block of the decrypted ciphertext is divisible by 7, padding in if necessary. Each block is then split into sub-blocks of length 7. We can then convert each sub-block into the original character of the plaintext.

Algorithm 6 Decryption with the simplified ECIES

INPUT: ciphertext $(U(x_1, y_1), y)$.
OUTPUT: plaintext x.
1: $V(x_2, y_2) \leftarrow mU$
2: **for** $cipher \in y$ **do**
3: $char \leftarrow cipher/x_2$
4: $char \leftarrow$ PADDING($char$)
5: APPEND(x, $char$)
6: **end for**
7: $x \leftarrow$ SPLIT(x)
8: **for** $char \in x$ **do**
9: $char \leftarrow$ DEASCII($char$)
10: APPEND(x',$char$)
11: **end for**
12: **return** x'

We now provide the proof of correctness for the decryption algorithm (Algorithm 6).

Theorem 3.5. *The decryption algorithm with the simplified ECIES is correct.*

Proof. We now split the algorithm into two blocks: lines 1 – 6 and 7 – 12.

Lines 1 – 6. Let $(U(x_1, y_1))$ be the ECIES ciphertext and m be the private key. The first line of this block calculates $V = mU$, which is used to decrypt the ciphertext. Next, we observe the 'for' loop. Let x be an array of binary representation blocks of the plaintext which length is divisible by 7 and has a maximum length of N. We allow x to be empty. We assert that x is a loop invariant. At the beginning of the loop, x is indeed empty. At each iteration, we divide an element of y with x_2, the x-coordinate of V. By Definition 2.2, this decrypts that element to a block of binary representation of the plaintext. The block is then padded so that each block has a length that is divisible by 7 and appended to x. Therefore, at the end of the execution of this loop, we have an array of binary representation blocks of the plaintext which length is divisible by 7 and has a maximum length of N. Again, since we are dealing with a 'for' loop, this loop is guaranteed to terminate, completing this block's proof of correctness.

Lines 7 – 12. The first line of this block splits the array of plaintext blocks, x, into binary strings of length 7. Next, again we face a 'for' loop. Let x' be the original plaintext. Since x' is a string, we can treat x' as an array of characters, which, again, we allow to be empty. We assert that x' is a loop invariant. On loop entry, x' is empty. At each iteration, each element of the array is translated back from a binary string ASCII code into the character represented by the ASCII code. This character is appended into x'. Therefore, x' is an array of characters of the original plaintext, proving that x' is indeed a loop invariant. Since we are dealing with a 'for' loop, the loop is guaranteed to terminate, and by the termination of the loop, we have recovered all characters of the original plaintext. We have established the correctness of this block.

Since we have established the correctness of each block, we have also established the correctness of the decryption algorithm. □

4. Results

We implemented the aforementioned algorithms in an encryption-decryption simulation program written in Python. A sample output of the program is given in Figure 1. Basically, the program first randomizes the curve parameters, then picks a point and calculates its order. Afterwards, the user is asked to provide a plaintext. Once the plaintext is set, the program encrypts the plaintext, then displays the resulting ciphertext. Next, the program prompts the user to enter

the ciphertext. Once the ciphertext is set, the program decrypts the ciphertext, then displays the recovered plaintext.

Several encryption and decryption simulations were conducted using the program. During the simulation, we attempt to encrypt the following plaintext:

> Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aliquam in nulla quis mi maximus suscipit at sed ipsum. Maecenas tincidunt turpis diam, sed ultricies justo vulputate hendrerit. Donec eget metus nec tortor finibus varius a mattis turpis. In facilisis convallis hendrerit. Praesent pellentesque eros sed metus feugiat dictum. Donec vel nibh mollis, dignissim magna ac, tempus nisl. Vestibulum eleifend pretium lorem at consectetur. Proin lacinia pharetra ipsum, vel ornare urna hendrerit eu. Maecenas sed libero pretium, condimentum nulla tincidunt, egestas est. Sed placerat tortor a nisi lobortis vulputate. Vestibulum dignissim justo ac lectus varius, vulputate elementum sapien rhoncus. Suspendisse et condimentum nibh. Morbi eleifend nisl vel magna elementum euismod. Nunc turpis dolor, cursus vel eros at, mollis vestibulum urna.

which is a *lorem ipsum* text created by a *lorem ipsum* generator. It consists of 842 bytes of information. For the purposes of the simulation, we restrict the program to work in $GF(2^{20})$.

The results of the simulation is shown in Table 1. It is worth noticing that for all cases, the point generation time (PtGenTime), encryption time (EncTime), and decryption time (DecTime) is relatively stable and fast, given the amount of text encrypted (and decrypted). On the other hand, the point order calculation times (PtOrdTime) are extremely varied and relatively slow. This is due to the way Hasse's theorem (Theorem 3.3) is implemented in the program. The program simply tries every possible integer in the Hasse interval until it finds an integer M which satisfies $MP = \mathcal{O}$. The factorization is performed by checking from a predetermined list of primes, which is also rather inefficient.

5. Conclusion

We have established several algorithms for implementing the simplified ECIES over binary field. Several cases must be handled when adding points, which are the case where the point one (or both) points are points at infinity, and the case where the points are the same. The point generation process can be done effectively using Theorems 3.1 and 3.2. The encryption and decryption process can also be done efficiently using the provided algorithm. On the other hand, there is room for improvement to the point order calculation.

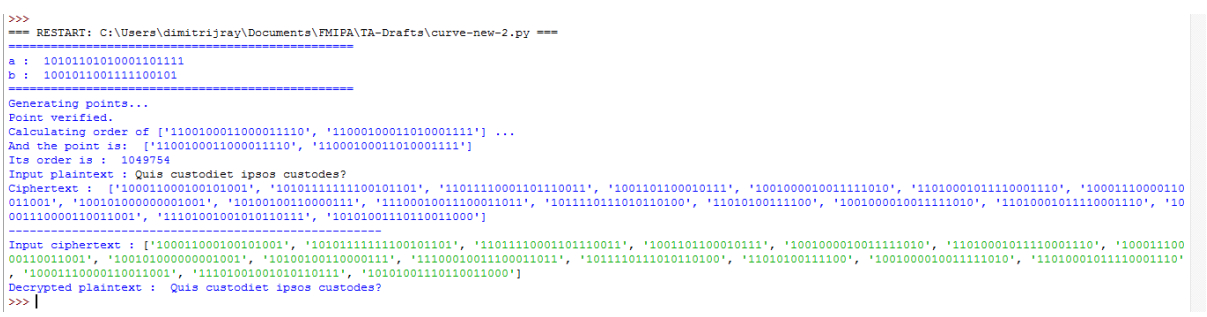

Figure 1. Encryption-decryption program, executed via the Python IDLE GUI

No.	Curve Parameters		Point chosen		Order	PtGenTime	PtOrdTime	EncTime	DecTime
	a	b	x	y					
1	11010100010100001	11111011110011100000	11010100010100001	11111011110011100000	1048870	0.0102	22.1138	0.3178	0.2799
2	1000110000101000101	11000010111011100110	11100000110011110111	10110010100000111110	1048392	0.0102	17.9539	0.3177	0.2693
3	1011111001000010011	11010100010010000100	10110101000101011	10111010010111001100	1049464	0.0099	28.3902	0.3238	0.2721
4	11010011101001101000	11011001000010110110	10100100111111010100	10000110011100001010	1050416	0.0926	38.0370	0.2978	0.3185
5	11101000101101011101	10000111110000100111	100101000110100100	1110011011000100011	1047226	0.1215	9.1655	0.3091	0.4407
6	111101100111101000	11100001011111110011	10100110111100000000	11000011100101010000	1048222	0.1661	28.4010	0.3173	0.3306
7	11100011101010001101	1010100000011100101	10000011011100000101	100100000111110111	1048202	0.0127	30.2198	0.2818	0.3142
8	11110101111011110	11011001001011001001	100111110110100100	110101001110100000	210008	0.0193	57.3799	0.2630	0.5272
9	11011000101100011110	11011101110000111010	10100010111110100110	10011000001111111100	1047440	0.1303	15.7834	0.3129	0.4970
10	1110011010110111101	11011001010110010001	11000101010010010	10101011010110111110	1049842	0.0118	33.2842	0.2506	0.3056
11	111011100111010001	10001101110110100011	1010011011010101010	11111101111101000111	1047130	0.0968	7.0309	0.2690	0.3076
12	101000110101010011110	11110100110101111111	11010100010101001110	10011111010111010010	1050166	0.0121	37.1911	0.2341	0.2961
13	10001110100000111010	111000001110010000	10110011010001000110	11111010000001011010	1049396	0.0218	29.2950	0.2611	0.3748
14	111110100111010011	10111000100000000110	101111010100010001	10110000011011111011	1046846	0.0208	3.7891	0.2569	0.3157
15	110111100110100	10110001000000010101	11111010010111110111	1001101100000101100	116292	0.1288	1.2743	0.2677	0.2636
16	1100010100011100	1000001010100111010	1110111110001001	11110010101011111101	116328	0.0309	4.6491	0.2794	0.3814
17	100000111001011011	100000000100011000	11010110111111101001	100000101000110111	349762	0.0954	29.8303	0.2478	0.2640
18	11100111001000010001	11011000111000001011	11000010101001100	11010000100111101010	209482	0.1013	8.5569	0.2518	0.2582
19	10010100011011101011	1111001111100010110	100000110001101100	110101000100111011	95332	0.0238	26.3968	0.2518	0.2466
20	10101101101110011101	11101100110101011011	11100110100010011111	10011101111001111100	13978	0.0963	19.3744	0.2600	0.2874
21	1011010011110100011	1100100010100000101	11010000111010100110	11010111011010011000	349548	0.1195	21.5548	0.2499	0.2529
22	10011011011000110011	11010100011010100010	11111111100010101111	10101001010110100010	1047328	0.0992	7.2352	0.2393	0.2555
23	100000000101111000	1101000011000111111	11110110011110000111	10100111011111101011	149818	0.1355	25.2752	0.2933	0.2811
24	10110111000000011100	100000010000001010	110100110011011001	100001011000000101	1048438	0.1017	21.1759	0.2504	0.2529
25	1010100010101111000	11100011100000101111	10011110000110001	11010001110110000011	209364	0.1075	2.6863	0.2926	0.2808

Table 1. Experiment results with the encryption-decryption program. The PtGenTime column shows the time required to generate a valid point in the curve. The PtOrdTime column shows the time required to calculate the chosen point's order. The EncTime and DecTime column shows the time required to process the encryption and decryption, respectively.

References

[1] Cherly J, Gallardo L, Leonid V and Wheland E 1998 Solving quadratic equations over polynomial rings of characteristic two *Publicacions Matemàtiques* **42**
[2] Hankerson D, Menezes A and Vanstone S 2004 *Guide to Elliptic Curve Cryptography* (New York: Springer-Verlag)
[3] Maulana M, Senjaya W F, Rahardjo B, Muchtadi-Alamsyah I and Paryasto M W 2014 Implementation of finite field arithmetic operations for polynomial and normal basis representations *3rd Int. Conf. on Computation for Science and Technology (ICCST-3)*
[4] Pommerening K 2012 Quadratic equations in finite fields of characteristic 2 retrieved from `www.staff.uni-mainz.de/pommeren/MathMisc/QuGlChar2.pdf`
[5] Paar C and Pelzl J 2010 *Understanding Cryptography* (New York: Springer)
[6] Shanmugalakshmi R and Prabu M 2009 Research issues on Elliptic Curve Cryptography and its applications *International Journal of Computer Science and Network Security* **9**
[7] Silverman J and Tate J 1994 *Rational Points on Elliptic Curves* (New York: Springer)
[8] Silverman J 2009 *The Arithmetic of Elliptic Curves* (New York: Springer)
[9] Stinson D 2006 *Cryptography: Theory and Practice 3rd Edition*, (Boca Raton: Chapman & Hall/CRC)