Middle East Technical University Department Of Cyber Security

CSEC 519

Blockchain and Cryptocurrency Technologies

Spring 2024-2025

First Assignment

Name Surname: Alkım Doğan
Student ID: 2521482

1

Z_8^* is the set of elements $1, 2, 3, 4, 5, 6, 7$ with group (or maybe not) operator of multiplication $(\cdot)$.

Lets check for group axioms one by one.

1. **Associativity**: We have associativity in our case since the group operator is multiplication.
   $a \cdot (b \cdot c) = (a \cdot b) \cdot c$ holds.
2. **Identity element**: As we not from basic mathematics, 1 is our identity element since any number multiplied with 1 is the number itself. This axiom also holds.
   $1 \cdot 1 \equiv 1 \mod 8$
   $2 \cdot 1 \equiv 2 \mod 8$
   $3 \cdot 1 \equiv 3 \mod 8$
   $4 \cdot 1 \equiv 4 \mod 8$
   $5 \cdot 1 \equiv 5 \mod 8$
   $6 \cdot 1 \equiv 6 \mod 8$
   $7 \cdot 1 \equiv 7 \mod 8$
3. **Inverse element**: Each element must have an inverse element to form a group. For each element e from the set, gcd(e,8)=1 must hold. But 2 does not have an inverse because gcd(2,8)$\neq$1. Therefore, the elements 2,4,6 do not have any inverse. We can conclude this axiom does not hold.

Conclusion: not a group because the 3rd axiom does **NOT** hold.

2

Z_{13}^* is the set of elements $1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12$ with group operator of multiplication $(\cdot)$. There are 12 elements in this set.

Lets try each element one by one.

1. 1 cannot be the generator because it is the identity element.

2. Lets try the element 2
   $2^1 \equiv 2 \text{ mod } 13$
   $2^2 \equiv 4 \text{ mod } 13$
   $2^3 \equiv 8 \text{ mod } 13$
   $2^4 \equiv 3 \text{ mod } 13$
   $2^5 \equiv 6 \text{ mod } 13$
   $2^6 \equiv 12 \text{ mod } 13$
   $2^7 \equiv 11 \text{ mod } 13$
   $2^8 \equiv 9 \text{ mod } 13$
   $2^9 \equiv 5 \text{ mod } 13$
   $2^{10} \equiv 10 \text{ mod } 13$
   $2^{11} \equiv 7 \text{ mod } 13$
   $2^{12} \equiv 1 \text{ mod } 13$
   $2^{13} \equiv 2 \text{ mod } 13$ (We come to the beginning again by finding 2)

3. We do not need to try the remaining numbers since we have found the generator.

As we can see from the above, the element **2** has order of **12**. Therefore, 2 is one of the possible generators if there exists more than one generator.

3

- F_{13} is a field since 13 is a prime.
- We have $q = p = 13$ in our case which is totally fine with the elliptic curve standards.
- $y^2 = x^3 + ax + b$. We have this form and also $a, b \in F_{13}$ since $a = 1$ and $b = 3$.
- In our case, we have $a = 1$ and $b = 3$. $4a^3 + 27b^2 \not\equiv 0$ mod p where p is 13 must hold. $4 + 27(3^2) \equiv 0$. So, since this inequality does not hold, which is necessary for elliptic curves, we can conclude the given equality is **NOT** an elliptic curve.

4

I coded very basic script for this purpose and provided the code at the end. I will provide the output for the all of the points and the number of points.

```
[(1, 2), (1, 11), (2, 5), (2, 8), (6, 4), (6, 9), (7, 1), (7, 12), (9, 5), (9, 8),
(12, 0), (inf, inf)]
Of Points :12
```

5

I have found each of the generators. The output for finding the generators is below.

```
(6,4) is a generator
(6,9) is a generator
```

```
(7,1) is a generator
(7,12) is a generator
```

Now, I will generate the points from the first generator **(6,4)** by hand one by one to proof my code.

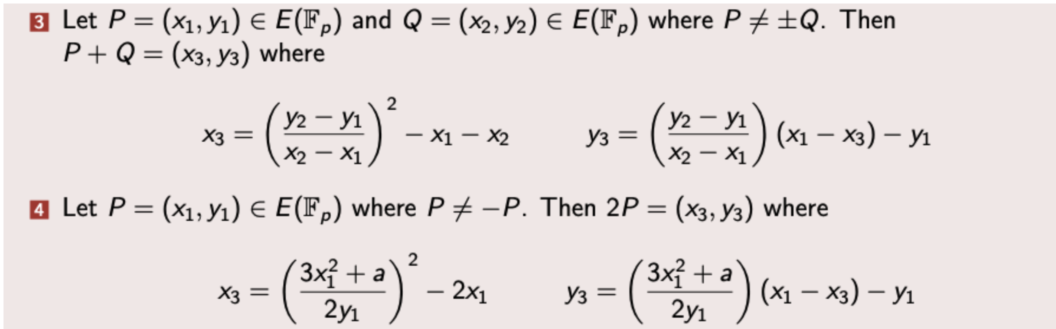

Figure 1: Formulas I have made use of from our lecture slides

1. $$P = (6,4)$$

2. $$2P_x = ((4+1) * 8^{-1})^2 - 12 = 2$$
$$2P_y = ((4+1) * 8^{-1})(6-2) - 4 = 5$$
$$2P = (2,5)$$

3. $$3P_x = (1 * 9^{-1})^2 - 6 - 2 = 1$$
$$3P_y = (1 * 9^{-1})(6-1) - 4 = 11$$
$$3P = (1,11)$$

4. $$4P_x = (7 * 8^{-1})^2 - 6 - 1 = 9$$
$$4P_y = (7 * 8^{-1})(6-9) - 4 = 8$$
$$4P = (9,8)$$

5. $$5P_x = (4 * 3^{-1})^2 - 6 - 9 = 7$$
$$5P_y = (4 * 3^{-1})(6-7) - 4 = 12$$
$$5P = (7,12)$$

6. $$6P_x = (8 * 1^{-1})^2 - 6 - 7 = 12$$
$$6P_y = (8 * 1^{-1})(6-12) - 4 = 0$$
$$6P = (12,0)$$

7.
$$7P_x = (9 * 6^{-1})^2 - 6 - 12 = 7$$
$$7P_y = (9 * 6^{-1})(6 - 7) - 4 = 1$$
$$7P = (7, 1)$$

8.
$$8P_x = (10 * 1^{-1})^2 - 6 - 7 = 9$$
$$8P_y = (10 * 1^{-1})(6 - 9) - 4 = 5$$
$$8P = (9, 5)$$

9.
$$9P_x = (1 * 3^{-1})^2 - 6 - 9 = 1$$
$$9P_y = (1 * 3^{-1})(6 - 1) - 4 = 2$$
$$9P = (1, 2)$$

10.
$$10P_x = (11 * 8^{-1})^2 - 6 - 1 = 2$$
$$10P_y = (11 * 8^{-1})(6 - 2) - 4 = 8$$
$$10P = (2, 8)$$

11.
$$11P_x = (4 * 9^{-1})^2 - 6 - 2 = 6$$
$$11P_y = (4 * 9^{-1})(6 - 6) - 4 = 9$$
$$11P = (6, 9)$$

12.
$$12P_x = (5 * 0^{-1})^2 - 6 - 6 = \infty$$
$$12P = (\infty, \infty)$$

This is the algorithm and proof of my code.

6 CODE (I will also submit .py file explicitly)

```
def gcd_extended(a, b, x, y):
  if a == 0:
    x[0] = 0
    y[0] = 1
    return b

  x1, y1 = [0], [0]
  gcd = gcd_extended(b % a, a, x1, y1)

  x[0] = y1[0] - (b // a) * x1[0]
  y[0] = x1[0]
  return gcd

def find_gcd(a, b):
  x, y = [1], [1]
```

```python
  return gcd_extended(a, b, x, y)

def get_inverse(A, M):
  x, y = [1], [1]
  g = gcd_extended(A, M, x, y)

  if g == 1:
    return (x[0] % M + M) % M

  return None


def get_all_points(field_size, a, b):
  points = []
  for i in range(field_size):
    rhs = (i * i * i + a * i + b) % field_size
    for j in range(field_size):
      if ( j * j) % field_size == rhs:
        points.append((i,j))
        if j != 0:
          points.append((i, field_size-j))
        break
  points.extend([(float("inf"),float("inf"))])
  return points


def get_double(p, field_size, a, b):

  x,y = p[0], p[1]

  if y == 0:
    return None

  payda = get_inverse(2 * y, field_size)
  x3 = ( 3 * x * x  + a ) * payda
  x3 *= x3
  x3 = (x3 - 2 * x ) % field_size

  y3 = (3 * x * x + a ) * payda * (x - x3) - y
  y3 = y3 % field_size

  return x3,y3

def get_sum(p1, p2, field_size, a , b):
  x1, y1, x2, y2 = p1[0], p1[1], p2[0], p2[1]

  if x1 == x2:
    return None

  pay = (y2 - y1) % field_size
  payda = (x2 - x1) % field_size

  payda = get_inverse(payda, field_size)

  if payda is None:
    return None
```

```python
    x3 = pay * payda
    x3 = (x3 * x3 - x1 - x2) % field_size

    y3 = (pay * payda * ( x1 - x3 ) - y1) % field_size

    return x3, y3


def order(p, field_size, a, b):

    cnt = 1
    var = get_double(p, field_size, a, b)

    if var is None:
        return cnt

    cnt += 1

    while True:

        var = get_sum(p, var, field_size, a , b)

        if var is None:
            break

        cnt += 1

    cnt += 1
    return cnt



def find_generator(all_points, field_size, a, b):
    for p in all_points:
        if p[0] == float("inf"):
            continue
        if order(p, field_size, a, b) == len(all_points):
            print(f'({p[0]},{p[1]}) is a generator')

def main():
    field_size = 13
    a = 1
    b = 2
    all_points = get_all_points(field_size, a, b)

    print(all_points)

    print(f'# Of Points :{len(all_points)}')

    find_generator(all_points, field_size, a, b)


if __name__ == '__main__':
    main()
```