Politechnika Krakowska
im. Tadeusza Kościuszki

Dokumentacja – Algorytm kryptograficzny - ECC.

Gabriela Rączka

Nr albumu:147085

gr nr 02

1. Temat

Elliptic Curve Cryptography (ECC) to kryptografia klucza publicznego opracowana niezależnie przez Victora Millera i Neala Koblitza w roku 1985. W kryptografii krzywych eliptycznych będziemy używać równania krzywej postaci

$y^2 = x^3 + ax + b$

które jest znane jako równanie Weierstrassa, gdzie a i b są stałymi z

$4a^3 + 27b^2 =/= 0$

Ten program implementuje protokół ECDH (Elliptic Curve Diffie-Hellman) w celu generowania kluczy prywatnych i publicznych, a także w celu wymiany „sekretu" między dwoma użytkownikami.

2. Opis matematyczny analizowanego algorytmu, opierając się na tym jakie działania arytmetyczne są w nim wykonywanie.

Krzywa eliptyczna to krzywa opisana równaniem, które można zapisać w następującej postaci:

$y^2 = x^3 + az + b$

Do kryptografii krzywych eliptycznych używamy nazwanych krzywych. Oznacza to, że krzywym i wykonywanym na nich obliczeniom towarzyszą wartości a i b oraz dodatkowe parametry (do wykorzystania w obliczeniach).

Weźmy znany przykład: dla krzywej w naszym w punkcie (1,1)

$y^2 = x^3 + x + 1$

Aby użyć krzywych do kryptografii, potrzebujemy kilku dodatkowych parametrów powiązanych z nazwaną krzywą.

To są:

- Punkt generatora (G)
  Jest to punkt na krzywej, którego wszyscy używają podczas tworzenia (lub weryfikowania) zaszyfrowanej wiadomości.
- parametr p definiujący koniec ciała skończonego (Fp)
  Ciało skończone to pole liczbowe zawierające liczby całkowite między dwoma punktami końcowymi. Fp jest skończonym polem złożonym z liczb całkowitych od 0 do p-1, matematycznie opisanym jako:

  $F_p = \{0, \ldots, p-1\}$

  Ponieważ nasza krzywa jest używana na polu skończonym, wzór na tę krzywą jest również zmodyfikowany:

  $y^2 = x^3 + ax + b \pmod p$

  $y^2 = x^3 + x + 1 \pmod p$

- parametr **n** określający maksymalną wartość klucza
  Określa maksymalną wartość klucza prywatnego. Wartość klucza prywatnego musi mieścić się w przedziale od **0** do **n-1** Ta wartość **n** jest rzędu G.

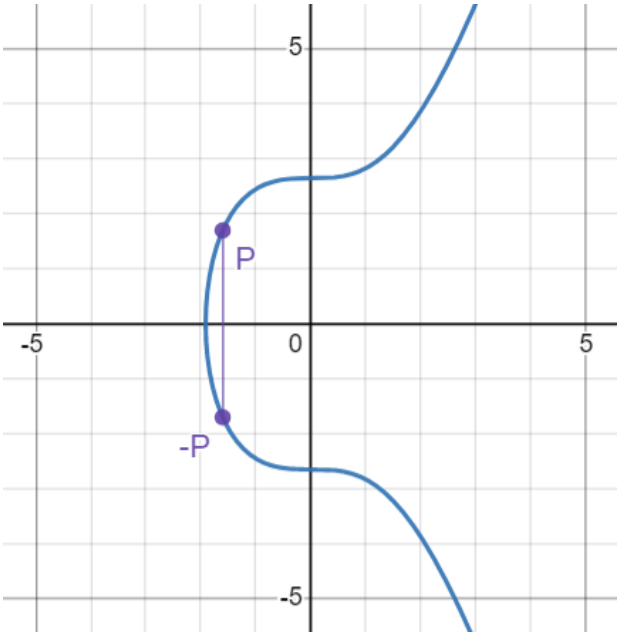
Aby zrozumieć, jak działa kryptografia krzywych eliptycznych, musimy znać podstawowe operacje matematyczne, które można wykonać na krzywej eliptycznej.

Podstawowe operacje, które można wykonać to:

- negacja punktowa
- dodanie dwóch punktów
- powielanie punktu (dodawanie punktu do siebie)
- mnożenie przez liczbę (wartość skalarną) można osiągnąć przez powtarzanie dodawania.

Ważną właściwością krzywych eliptycznych jest to, że operacji dzielenia nie można zinterpretować.

Negacja punktu to nic innego jak jego odbicie na osi poziomej.



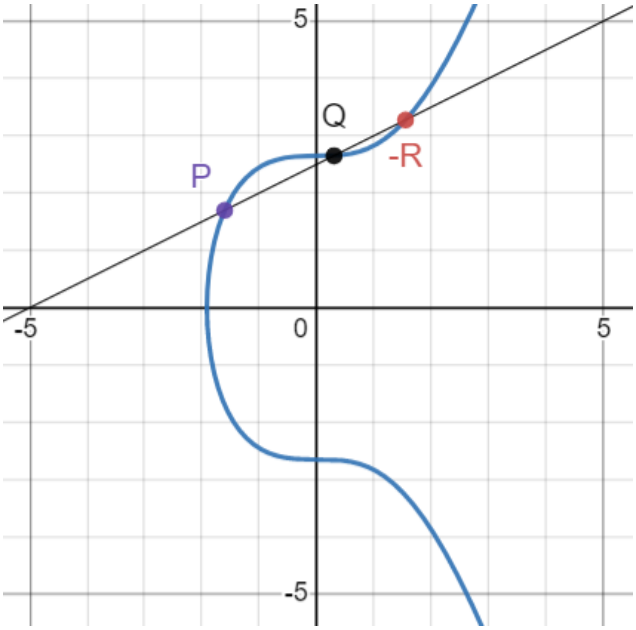
Dodanie dwóch punktów

Z definicji połączenie dwóch punktów linią daje trzeci punkt ze znakiem ujemnym.

$P+Q= -R$

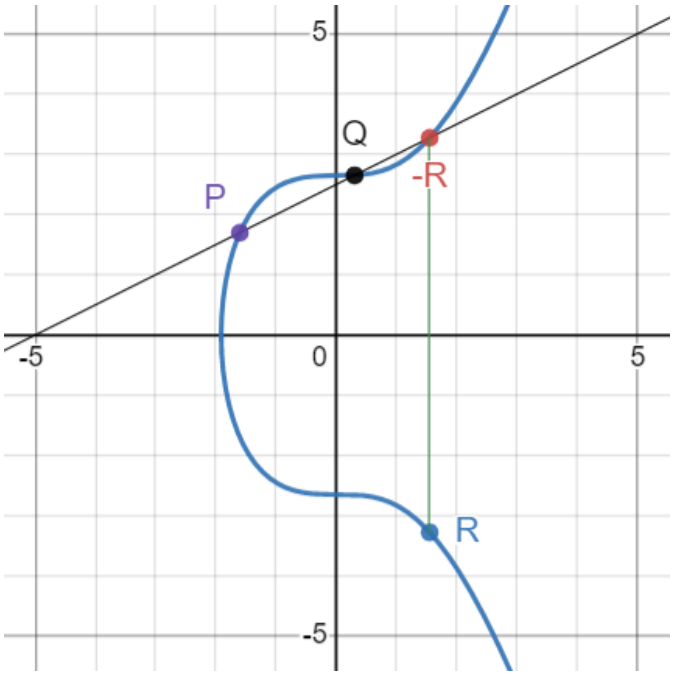
(Definicja dodawania krzywych eliptycznych różni się od tradycyjnego dodawania).

To wygląda tak:



Aby uzyskać +R, należy wykonać odbicie na osi poziomej (negację).

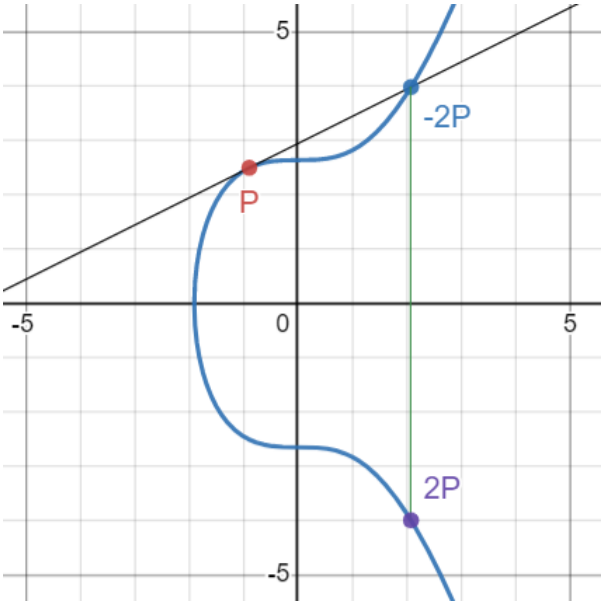
Wynikiem jest punkt R.



Powielanie punktu

Jest to szczególny przypadek dodawania punktów, kiedy punkt jest dodawany do siebie.

W tym przypadku dwa punkty są w rzeczywistości zbieżne, więc należy narysować linię styczną. Linia przecina krzywą w punkcie -2P, który musi zostać odbity lustrzanie, aby uzyskać duplikat punktu, 2P, na krzywej eliptycznej.

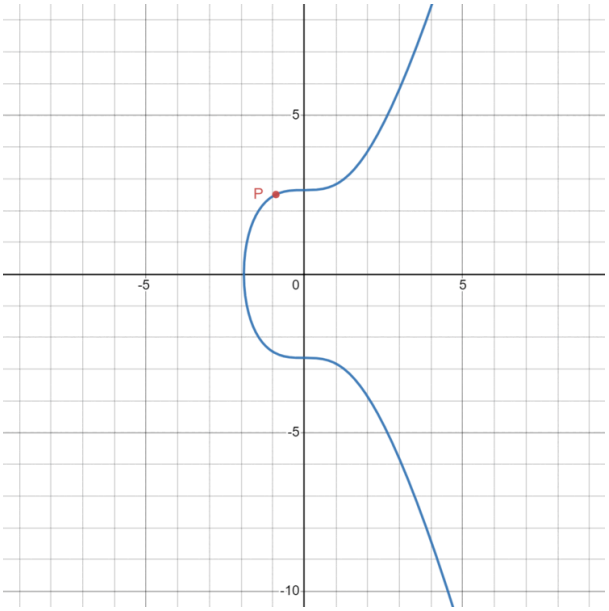


Mnożenie punktu przez wartość liczbową

Ponieważ możemy dodać dwa punkty razem lub zduplikować punkt (dodając go do siebie), możemy to łączyć wiele razy. Oznacza to, że możemy pomnożyć nasz punkt przez dowolną liczbę.

W poniższym przykładzie punkt P jest mnożony przez 3:

1. rysowana jest styczna do punktu ***P*** , a wynikowe przecięcie ***-2P*** jest odbijane lustrzanie, aby uzyskać ***2P*** .
2. punkty ***P*** i ***2P*** są połączone, a wynikowe przecięcie ***-3P*** jest odzwierciedlone w celu uzyskania ***3P***



W tym przykładzie wykonaliśmy pomnożenie przez 3, ale pomnożenie przez większą wartość oznacza oczywiście, że operacje dodawania i powielania muszą zostać powtórzone.

W kryptografii krzywych eliptycznych liczba używana do mnożenia ma długość 256 bitów (2^{255}) i jest to klucz prywatny.

W jaki sposób tworzona jest para kluczy? Dlaczego jest bezpieczny?

Klucz prywatny (*klucz prywatny*) to losowa liczba całkowita z przedziału od ***0*** do ***n-1*** , to właśnie generujemy.

Kluczem publicznym jest punkt generatora (G) pomnożony przez tę wygenerowaną liczbę losową na krzywej eliptycznej.

*klucz publiczny = klucz prywatny * G*

Oznacza to, że musimy wykonać mnożenie z punktem G, jak opisano w poprzedniej sekcji.

Brak interpretacji podziału na krzywej eliptycznej uniemożliwia wyznaczenie liczby losowej reprezentującej klucz prywatny z klucza publicznego, który jest punktem na krzywej.

3. Pseudo kod algorytmu

Algorytm ECC (Elliptic Curve Cryptography) to sposób szyfrowania danych z wykorzystaniem krzywych eliptycznych. Jednym z najczęściej wykorzystywanych algorytmów z tej rodziny jest ECDH (Elliptic Curve Diffie-Hellman

W tym kodzie zdefiniowana jest klasa **EllipticCurve** opisująca krzywą eliptyczną na skończonym ciele. Krzywa ta jest określona przez dwie zmienne: **a** i **b**. Zmienne te są stałymi prywatnymi danymi składowymi klasy **EllipticCurve** i są używane do wyznaczania punktów na krzywej.

Klasa **Point** opisuje punkt na krzywej eliptycznej. Zawiera dwie zmienne prywatne danego składowe klasy **Point**: **x_** i **y_**, które odpowiadają współrzędnym x i y punktu. Ponadto klasa **Point** ma wskaźnik na krzywą eliptyczną, do której punkt należy.

Funkcja **addDouble** dokonuje podwójnego dodawania punktu, co oznacza, że suma punktu jest dodawana do samej siebie wielokrotnie, zależnie od podanej liczby **m**. Wynik jest przypisywany do punktu **acc**. (W przypadku dodawania dwóch identycznych punktów wykorzystywany jest wzór na podwajanie punktu.)

Funkcja **scalarMultiply** dokonuje mnożenia skalarowego punktu, co oznacza, że punkt jest dodawany do siebie wiele razy, zależnie od podanej liczby **k**. Wynik jest zwracany jako nowy punkt.

Funkcja **add** dodaje dwa punkty na krzywej eliptycznej. Wynik dodawania jest zwracany jako nowy punkt. Funkcja obsługuje specjalne przypadki, takie jak dodawanie z zerem, a także dodawanie punktu do jego odbicia względem osi x.

Funkcja **frand()** zwraca losową liczbę typu **float** z przedziału [0, 1].

Funkcja **irand()** zwraca losową liczbę typu **int.**

Funkcja **CalculatePoints():**

```
x_val[P]
y_val[P]
Dla n od 0 do P wykonaj:
    nsq = n*n
    x_val[n] = ((n*nsq) + a_.i() * n + b_.i()) % P
    y_val[n] = nsq % P

Dla n od 0 do P wykonaj:
    Dla m od 0 do P wykonaj:
        Jeśli x_val[n] równa się y_val[m] to:
            m_table_.push_back(Point(n,m,*this))

table_filled_ = true
```

Funkcja "CalculatePoints" ma za zadanie obliczyć wszystkie punkty na krzywej eliptycznej, które zawierają się w polu skończonym zdefiniowanym przez liczbę pierwszą P. Najpierw tworzone są tablice x_val[] i y_val[], które przechowują obliczone wartości dla każdej wartości n. Następnie, dla każdej pary (n,m), sprawdzane jest, czy x_val[n] równa się y_val[m]. Jeśli tak, to punkt (n,m) jest dodawany do wektora m_table_, który przechowuje wszystkie punkty na krzywej. Na końcu funkcja ustawia zmienną table_filled_ na true, aby oznaczyć, że tablica punktów została wypełniona.

Funkcja**: scalarMultiply**(int k, const Point& a)

```
Jesli k = 0 to
    Zwroc punkt (0, 0, *ec_)
Jesli k = 1 to
    Zwroc punkt a

Inicjuj:
    acc = a
    res = punkt (0, 0, *ec_)
    i = 0
    j = 0
    b = k
```

```
Dopoki b != 0:
    Jesli b & 1 != 0 to
        addDouble(i - j, acc)
        res += acc
        j = i
    b >>= 1
    i += 1

Zwroc res
```

Jest to funkcja klasy obliczająca iloczyn skalarny k * a. W funkcji **scalarMultiply**, punkt a jest mnożony przez k (skalar). Zaczynamy od punktu **a** i składamy go z samym sobą. Następnie, jeśli bit k jest ustawiony na 1, dodajemy punkt **acc** do punktu **res**, a następnie **acc** jest przesuwany o **i-j**. Powtarzamy ten proces dla wszystkich bitów k. Wynikiem jest zsumowanie punktów k razy.

Funkcja **add**(x1, y1, x2, y2, xR, yR)

```
// Specjalne przypadki dotyczące elementu neutralnego dodawania
Jeśl x1 == 0 i y1 == 0:
    Wykonaj : xR = x2
    yR = y
Jeśli x2 == 0 i y2 == 0:
  Wykonaj :  xR = x1
    yR = y1
Jeśli y1 == -y2:
  Wykonaj :  xR = yR = 0

// Dodawanie punktów
Jeśli x1 == x2 i y1 == y2:
    // 2P
   Wykonaj : s = (3 * x1^2 + a) / (2 * y1)
    xR = s^2 - 2 * x1
W przeciwnym razie:
    // P+Q
```

Wykonaj : s = (y1 - y2) / (x1 - x2)

xR = s^2 - x1 - x2

Jeśli s != 0:

Wykonaj : yR = -y1 + s * (x1 - xR)

W przeciwnym razie:

Wykonaj : xR = yR = 0

Powyższy pseudokod przedstawia funkcję **add**, która dodaje dwa punkty na krzywej eliptycznej. Funkcja **add** dodaje dwa punkty na krzywej eliptycznej. Jeśli któreś z punktów jest punktem **zerowym** (0,0), to wynikiem dodawania jest punkt niezerowy. Gdy punkty mają odwrotne y, wynikiem dodawania jest punkt zerowy. W innym przypadku obliczamy **s**, który jest współczynnikiem nachylenia prostej przechodzącej przez punkty.

4. Instrukcja obsługi – np. ze zrzutami ekranu prezentującymi działanie projektu.

Poniżej opisuję, jak działają niektóre fragmenty tego kodu.

Na początku definiowany jest obiekt myEllipticCurve klasy EllipticCurve. Jego konstruktor przyjmuje dwa argumenty, 1 i 1, co odpowiada parametrom krzywej eliptycznej y^2 = x^3 + ax + b. Wartości te określają krzywą eliptyczną, na której będzie operować algorytm ECC. W tym przypadku użyta jest krzywa o długości klucza 263 bitów.

Następnie algorytm generuje punkty na krzywej eliptycznej za pomocą funkcji myEllipticCurve.CalculatePoints(), a następnie wypisuje je na ekranie za pomocą funkcji myEllipticCurve.PrintTable(cout,5).

Dalej w kodzie tworzone są trzy pary kluczy prywatnych i publicznych. Każdy klucz prywatny to liczba całkowita (odpowiadająca punktowi na krzywej eliptycznej), a klucz publiczny to iloczyn punktu na krzywej eliptycznej i liczby całkowitej.

Pierwsza osoba szyfruje swoją wiadomość, dzieląc ją na dwie części, a następnie mnożąc każdą z części przez klucz publiczny drugiej osoby. Otrzymywane wyniki są następnie przesyłane do drugiej osoby, która odszyfrowuje wiadomość, mnożąc otrzymane punkty przez swój klucz prywatny i dzieląc uzyskane wartości przez odpowiednie współrzędne klucza publicznego pierwszej osoby.

Ostatni fragment kodu przed zakończeniem programu pokazuje, jak trzecia osoba próbuje przechwycić wiadomość i odszyfrować ją za pomocą swojego klucza prywatnego, jednak wynik jest zupełnie inny, ponieważ trzecia osoba nie ma prawidłowego klucza prywatnego.

```
A little Elliptic Curve cryptography example

The elliptic curve: y^2 mod 263 = (x^3+1x+1) mod 263

Points on the curve (i.e. the group elements):
(0, 1) (0, 262) (1, 23) (1, 240) (2, 96) (2, 167)
(3, 89) (3, 174) (4, 73) (4, 190) (6, 111) (6, 152)
(7, 80) (7, 183) (8, 28) (8, 235) (10, 119) (10, 144)
(14, 38) (14, 225) (15, 32) (15, 231) (21, 128) (21, 135)
(22, 64) (22, 199) (23, 57) (23, 206) (24, 35) (24, 228)
(27, 81) (27, 182) (29, 111) (29, 152) (30, 87) (30, 176)
(31, 34) (31, 229) (33, 27) (33, 236) (38, 101) (38, 162)
(39, 45) (39, 218) (40, 55) (40, 208) (44, 65) (44, 198)
(45, 35) (45, 228) (47, 81) (47, 182) (48, 60) (48, 203)
(49, 123) (49, 140) (51, 64) (51, 199) (57, 25) (57, 238)
(58, 5) (58, 258) (59, 6) (59, 257) (60, 123) (60, 140)
(61, 52) (61, 211) (66, 34) (66, 229) (67, 119) (67, 144)
(68, 74) (68, 189) (69, 108) (69, 155) (72, 85) (72, 178)
(74, 4) (74, 259) (75, 25) (75, 238) (77, 116) (77, 147)
(78, 53) (78, 210) (81, 0) (82, 27) (82, 236) (83, 114)
(83, 149) (87, 61) (87, 202) (88, 91) (88, 172) (99, 79)
(99, 184) (103, 131) (103, 132) (108, 83) (108, 180) (110, 130)
(110, 133) (111, 77) (111, 186) (112, 44) (112, 219) (115, 59)
(115, 204) (116, 125) (116, 138) (117, 18) (117, 245) (118, 106)
(118, 157) (123, 23) (123, 240) (127, 2) (127, 261) (131, 25)
(131, 238) (132, 70) (132, 193) (134, 29) (134, 234) (137, 107)
(137, 156) (138, 29) (138, 234) (139, 23) (139, 240) (141, 109)
(141, 154) (142, 26) (142, 237) (143, 37) (143, 226) (144, 69)
(144, 194) (145, 115) (145, 148) (146, 97) (146, 166) (147, 94)
(147, 169) (148, 27) (148, 236) (150, 129) (150, 134) (152, 124)
(152, 139) (154, 123) (154, 140) (157, 100) (157, 163) (159, 102)
(159, 161) (162, 104) (162, 159) (166, 34) (166, 229) (169, 107)
(169, 156) (170, 130) (170, 133) (173, 90) (173, 173) (174, 109)
(174, 154) (175, 20) (175, 243) (180, 122) (180, 141) (181, 59)
(181, 204) (182, 110) (182, 153) (184, 43) (184, 220) (185, 127)
(185, 136) (186, 119) (186, 144) (188, 70) (188, 193) (189, 81)
(189, 182) (190, 64) (190, 199) (192, 15) (192, 248) (194, 35)
(194, 228) (195, 7) (195, 256) (196, 116) (196, 147) (199, 2)
(199, 261) (200, 2) (200, 261) (206, 70) (206, 193) (207, 117)
(207, 146) (208, 24) (208, 239) (210, 79) (210, 184) (211, 109)
(211, 154) (216, 4) (216, 259) (217, 79) (217, 184) (218, 108)
(218, 155) (219, 118) (219, 145) (220, 107) (220, 156) (221, 33)
(221, 230) (222, 58) (222, 205) (223, 50) (223, 213) (224, 9)
(224, 254) (226, 99) (226, 164) (228, 111) (228, 152) (230, 59)
(230, 204) (236, 4) (236, 259) (239, 108) (239, 155) (240, 31)
(240, 232) (242, 72) (242, 191) (245, 48) (245, 215) (246, 130)
(246, 133) (249, 98) (249, 165) (251, 75) (251, 188) (253, 116)
(253, 147) (254, 29) (254, 234) (259, 14) (259, 249) (260, 51)
(260, 212)
```

```
some point P  = (1, 23), 2P = (87, 61)
some point Q = (1, 240), P+Q = (0, 0)
P += Q = (0, 0)
P += P = 2P = (87, 61)

EC message encryption example
==============================================

G = (148, 27), order(G) is 15
First person public key Pa = 51*(148, 27) = (3, 89)
Second person public key Pb = 212*(148, 27) = (61, 52)
Third person public key Pj = 153*(148, 27) = (195, 256)


Plain text message from first person to second person: (19, 72)
Encrypted message from first person to second person = {Pa,c1,c2} = {(3, 89), 193, 153}

        Second person decrypted message from first person = (19, 72)

Third person decrypted message from first person = (203, 191)
```