МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА
МЕХАНІКО-МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
КАФЕДРА АЛГЕБРИ І КОМП'ЮТЕРНОЇ МАТЕМАТИКИ

Алгоритми генерування еліптичних кривих, придатних для обчислення білінійного спарювання

Курсова робота

Студент-виконавець:
Авраменко Нікіта Васильович
III курс, 1 група
спеціальність:
"комп'ютерна математика"

Роботу захищено
з оцінкою: _____

Науковий керівник:
Олійник Андрій Степанович
Доктор, професор кафедри
алгебри і комп'ютерної
математики

Члени комісії:
____
____
____

____

(підпис)

Київ — 2021

Зміст

Вступ

В цій роботі розглянуто і реалізовано на мові Python алгоритм, запропонований у статті “Pairing-Friendly Elliptic Curves of Prime Order“ [1] Пауло Баррето та Майклом Наерігом. Він дозволяє генерувати еліптичні криві, які придатні до білінійного спарювання.

Не сингулярна еліптична крива над полем $\mathbb{F}_p$ є придатною до білінійного спарювання, якщо вона містить підгрупу порядку r із не дуже великим степенем занурення k. Тобто розрахунки в полі $\mathbb{F}_p$ обчислювально можливі. Такі криві простого порядку необхідні для систем на основі спарювання, таких як короткі цифрові підписи. Наприклад, довжина підпису BLS[2] дорівнює розміру заданого поля p. При 128-біт рівні безпеки схема повинна бути визначена на групі 256-біт порядку r і відображена на скінчене поле розміру p^k приблизно 3072-біт.

В першому розділі описана необхідна теорія за матеріалами статей. Коротко розглянуто еліптичні криві та основні їх властивості, які потім знадобляться у доведенні запопонованого алгоритму.

У другому розділі представлені результати роботи реалізованого алгоритму.

У розділі три підведені підсумки та узагальнені результати.

Після цього у додатку викладено програмний код реалізації алгоритму на мові Python 3.9. Було використано допоміжні бібліотеки sympy та дві стандартні бібліотеки math і time.

1 Необхідні допоміжні відомості

1.1 Еліптичні криві та дії над ними

Через те, що будемо працювати із еліптичними кривими, то потрібно спочатку коротко визначичи, що вони собою визначають. Більш детально це можна прочитати, наприклад, у [3].

1.1.1 Означення еліптичних кривих

Опишемо криві, із якими будемо працювати в подальшому.

Означення 1.1. [4] *Еліптична крива над полем K — це проєктиний многовид, визначений рівнянням Веєрштрассе:*

$$y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6,$$

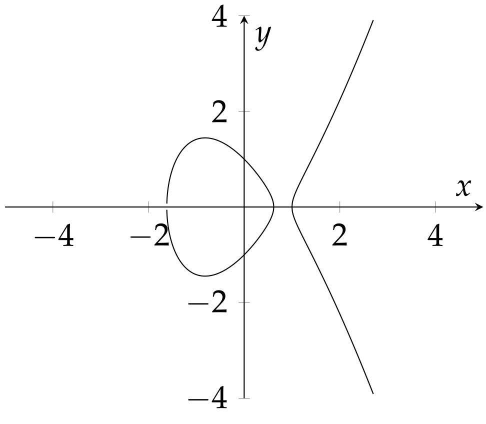
де $a_1, a_2, a_3, a_4, a_5, a_6 \in K$. Коли характеристика K не дорівнює 2 або 3, то можно спростити рівність до:

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

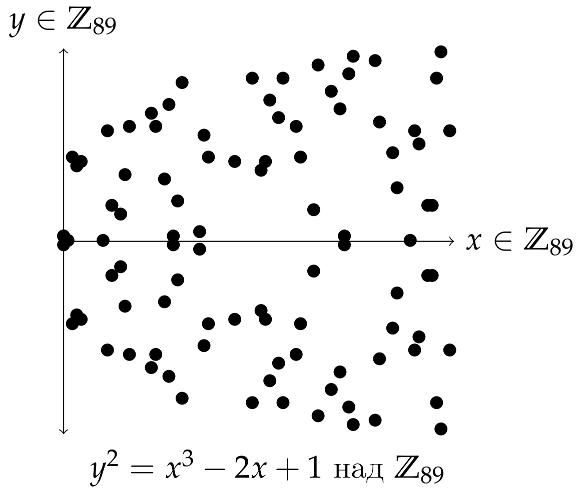
Ми будемо працювати над полем $\mathbb{F}_p$, тому можемо визначити множину точок еліптичної кривої таким чином:

$$E(\mathbb{F}_p) = \Big\{(x,y) \in (\mathbb{F}_p)^2 \mid y^2 \equiv x^3 + ax + b \pmod{p},\ 4a^3 + 27b^2 \not\equiv 0 \pmod{p}\Big\} \cup \{O\},$$

де $\Delta_E = 4a^3 + 27b^2 \not\equiv 0$ — дискрімінант кривої та умова несингулярності, а $\{O\}$ — точка в нескінченності і нейтральный елемент. Також еліптичні криві над $\mathbb{F}_p$ утворюють абелеву групу.



$y^2 = x^3 - 2x + 1$ над $\mathbb{R}$



$y^2 = x^3 - 2x + 1$ над $\mathbb{Z}_{89}$

1.1.2 Арифметика на еліптичних кривих

Нехай $P_1 = (x_1, y_1)$, $P_2 = (x_2, y_2)$, $P_1 + P_2 = P_3 = (x_3, y_3)$ точки на еліптичній кривій $E(\mathbb{F}_p)$. Маємо такі властивості:

- $-O = O$
- $P_1 + O = O + P_1 = P_1$
- $-P_1 = (x_1, -y_1 \pmod p)$
- $P_1 + (-P_1) = O$

В більш загальному випадку:

$$\begin{aligned} x_3 &= (m^2 - x_1 - x_2) && \bmod p \\ y_3 &= [y_1 + \lambda(x_3 - x_1)] && \bmod p \\ &= [y_2 + \lambda(x_3 - x_2)] && \bmod p \end{aligned}$$

Якщо $P_1 \neq P_2$, то нахил λ має вигляд:

$$\lambda = (y_1 - y_2)(x_1 - x_2)^{-1} \bmod p$$

Якщо $P_1 = P_2$, то нахил λ приймає форму:

$$\lambda = (3x_1^2 + a)(2y_1)^{-1} \bmod p$$

Тепер можемо визначити множення на скаляр: $V = nP = \overbrace{P + P + \cdots + P}^{n}$, $n \in \mathbb{Z}$.

Нам знадобиться це множення у програмній реалізації, тому розглянемо як його обчислити. Ми можемо покращити лінійну складність обчислення nP на логарифмічну використовуючи бінарний метод піднесення до степені із книги “Art of Computer Programming Seminumerical Algorithms“ [5] з невеликими змінами:

Algorithm 1: Бінарний метод обчислення nP

Input: $P \in E(\mathbb{F}_p), n \in \mathbb{N}$
Output: nP

$N \leftarrow n, Y \leftarrow 1, Z \leftarrow P$
while $N \neq 0$ **do**
  if N *непарне* **then**
    $Y \leftarrow Y + Z$
  end
  $Z \leftarrow Z + Z$
  $N \leftarrow \lfloor N/2 \rfloor$
end
return Y

Також додатково відмітимо випадки $0P = O$ і $(-n)P = n(-P) = -(nP)$.

У [6] такий підхід обчислення називается методом "double-and-add" і береться як базовий алгоритм, але зазначається, що у нього є різні більш ефективні модифікації.

1.1.3 Степінь занурення еліптичної кривої

Означення 1.2. [7] *Кажемо, що підгрупа G еліптичної кривої $E(\mathbb{F}_p)$ має степінь занурення або множник безпеки k, якщо підгрупа порядку r є дільником $p^k - 1$, але не є дільником $p^i - 1$ для $0 < i < k$.*

Спарювання Тейта [6, 8, 9] (або спарюванн Вейля[10]) відображає дискретний логарифм в G у дискретний логарифм в $\mathbb{F}_{p^k}$, що є основою атаки Фрея-Рюка [11].

Проблема полягає в тому, щоб побудувати таку еліптичну криву, аби її k був досить великим для запобігання атаки Фрея-Рюка, але достатньо малим для ефективного обчислення спарювання Тейта.

1.2 Ізогенія та комплексне множення

Означення 1.3. [4] *Ізогенія — це морфізм кривих, що залишає точку O фіксованою.*

Можна визначити множення на m відображенням $m : E(\mathbb{F}_p) \to E(\mathbb{F}_p)$ як:

$$P \to mP = P + P + \cdots + P, \forall P \in E(\mathbb{F}_p) \text{ та } m \geq 0.$$

Враховуючи те, що це морфізм кривих і відображення фіксує базову точку O, то маємо ізогенію. Це також сюр'єкція. [12] Отже, ми можемо визначити ядро $E[n]$, яке містить точки $E(\mathbb{F}_p)$ порядку n.

$$E[n] = \{P \in E(\mathbb{F}_p) : nP = O\}$$

Означення 1.4. [13] *Ендоморфізм еліптичної кривої E є ізогенією E в саму себе. Кільце ендоморфізмів E, яке позначимо $End(E)$, є множиною всіх ендоморфізмів E, причому додавання визначається точково, а множення — композицією.*

Тобто, якщо ϕ_1 та ϕ_2 ендоморфізми E, то ми можемо визначити ендоморфізм $(\phi_1 + \phi_2)$:

$$(\phi_1 + \phi_2) : E \to E,\ \ (\phi_1 + \phi_2)(P) = \phi_1(P) + \phi_2(P).$$

Також отримаємо ендоморфізм взявши композицію:

$$(\phi_1\phi_2) : E \to E,\ \ (\phi_1\phi_2)(P) = \phi_1(\phi_2(P)).$$

Тому безліч ендоморфізмів замкнуто при додаванні і множенні.

Означення 1.5. [13] *Еліптична крива E має комплексне множення (CM), якщо*

$$End(E) \supsetneq \mathbb{Z}.$$

Або іншими словами має нетривіальний ендоморфізм (ендоморфізм E, який не є відображенням множення на n). Якщо E не має комплексного множення, тоді кільце ендоморфізмів ізоморфно $\mathbb{Z}$.

1.3 Границя Гассе

Спочатку встановимо слід відображення Фробеніуса t, як $t = p + 1 - N$, де $N = \#E(\mathbb{F}_p)$.

Лема 1.6. [4] *Степеневе відображення $deg : End(E) \to \mathbb{Z}$ зодовільняє відношення*

$$|\deg(\phi - \psi) - \deg(\psi) - \deg(\phi)| \leq 2\sqrt{\deg(\psi)\deg(\phi)}$$

Доведення наведено у [12]

Теорема 1.7. [14] *Нехай* $E(\mathbb{F}_p)$ *еліптична крива, тоді:*

$$|t| \leq 2\sqrt{p}$$

Доведення. Визначимо відображення Фробеніуса $\psi : E(\mathbb{F}_p) \to E(\mathbb{F}_p)$ як

$$(x, y) \to (x^p, y^p).$$

За малою теоремою Ферма знаємо, що $x^p \equiv x \pmod{p}$, тоді відображення поточково фінксує на E

$$\psi(P) = P.$$

Звідси $\psi(P) - P = 0$, тобто $(\psi - 1)(P) = 0$.

$$P \in \ker(\psi - 1)$$

Такоим чином E ізоморфно ядру відображення $(\psi - 1)$. Ізоморфізм дає

$$N = \#\ker(\psi - 1) = \deg(\psi - 1)$$

За лемою 1.6

$$|\deg(\psi - 1) - \deg(\psi) - \deg(1)| \leq 2\sqrt{\deg(\psi)\deg(1)}.$$

$\deg(\psi - 1) = N,\ \deg(\psi) = p$ і $\deg 1 = 1$, тоді

$$|N - p - 1| = |t| \leq 2\sqrt{p}.$$

□

1.4 Тест Бейлі-Померанца-Селфріджа-Уогстаффа

В алгоритмі, який буде описано в наступному підрозділі, використовується перевірка числа на простоту. Для еліптичних кривих нам потрібно саме просте число, не псевдопросте і тому подібне. Гарним підходом було б використовувати детермінований тест на простоту. Але через те, що в програмі використовуються числа досить великої бінарної довжини, то реалізація на Python занадто довго працює. Через це було використано тест на простоту із пакету sympy, який використовуює Тест Бейлі-Померанца-Селфріджа-Уогстаффа (БПСУ).

Це ймовірносний тест. Вважається, что він має нескінченно багато складених чисел, що пройдуть перевірку на простоту, але досі не відомо таких прикладів[15]. Тому, хоч вибір ймовірностного тесту не є повністю правильним, серед інших варінтів, які було знайдено, він є допустимим.

1.5 Запропонований алгоритм побудови кривої із степенем занурення $k = 12$

Будь-яка еліптична крива E над полем $\mathbb{F}_p$ задовільняє теорему Гассе, яка полягає в тому, що слід відображення Фробеніуса t, який залежить від N та p у рівнянні $N = p + 1 - t$, виконує нерівність $|t| \leq 2\sqrt{p}$.

Означення 1.8. [16] *Кажемо, що* m *є квадратичним лишком за модулем* n*, якщо існує ціле число* x*, для якого* $x^2 \equiv m \pmod{n}$*.*

У реалізації алгоритму перевірка "чи є число квадратичним лишком" та "знахождення розв'язку квадратичного лишку" взято із бібліотеки sympy.

Означення 1.9. [17] *Нехай* K *поле характеристики* p*,* n *— невід'ємне ціле число, що не ділиться на* p*. Тоді:*

$$\Phi_n(x) = \prod_{\substack{s=1 \\ \gcd(s,n)=1}}^{n} (x - \xi^s)$$

називається циклічним многочленом над полем K*.*

Теорема 1.10. [17] *Нехай* K *поле характеристики* p*,* n *— невід'ємне ціле число, що не ділиться на* p*. Тоді* $x^n - 1 = \prod_{d|n} \Phi_d(x)$

З $N = hr$ для деякого h і $q = (t-1) + hr$ випливає $q^u - 1 = (t-1)^u - 1 \pmod{r}$ для $u > 0$. Тоді будь-яке відповідне r повинно задовільняти $r|(t-1)^k - 1$ і $r \nmid (t-1)^i - 1$ для $0 < i < k$. Маємо таку лему:

Лема 1.11. [7] *Будь-яке придатне просте* r *задовільняє* $r|\Phi_k(t-1)$ *і* $r \nmid \Phi_k(t-1)$ *для* $0 < i < k$*.*

Доведення. Оскільки r просте і $r|(t-1)^k - 1$, то тоді $r|\Phi_d(t-1)$ для таких d, які є дільниками k. Єдиним варіантом буде $d = k$. Звідси $r|\Phi_k(t-1)$. □

Далі буде доведено алгоритм генерування еліптичних кривих зі степеню занурення 12, який запропонований у статті "Pairing-Friendly Elliptic Curves of Prime Order"' [1] Пауло Баррето та Майклом Наерігом.

Теорема 1.12. [1] *Існує ефективний алгоритм побудови еліптичної кривої простого порядку (майже довільної бінарної довжини) із степенем занурення* $k = 12$ *над полем простого порядку.*

Доведення. Будемо притримуватись стратегії параметризації $p(x), n(x), t(x)$ та використовувати властивість $n|\Phi_k(t-1)$ (Лема 1.11). Через те, що Φ_{12} квартичний і з границі Гассе маємо $n \sim p \sim t^2$ ми повинні взяти $t(x)$ як такий квадратний многочлен, що $n(x)$ є квартичним фактором $\Phi_{12}(t-1)$.

Гелбрейт в [18] показав, що тільки квадратні многочлени $u(x) = 2x^2$ та $u(x) = 6x^2$ розбивають Φ_{12} на два незвідних квартичні фактори. Взявши слід відображення Фробеніуса як $t(x) = 6x^2 + 1$ ми отримаємо

$$\Phi_{12}(t(x) - 1) = n(x)n(-x),$$

де $n(x) = 36x^4 + 36x^3 + 18x^2 + 6x + 1$. З відношення $n = p + 1 - t$ ми отримаємо незвідний многочлен $p(x) = n(x) + t(x) - 1 = 36x^4 + 36x^3 + 24x^2 + 6x + 1$. Рівняння норми CM стане

$$DV^2 = 4p - t^2 = 3(6x^2 + 4x + 1)^2.$$

Нехай для деякого x_0 обчислимо $n = n(x_0)$ і $p = p(x_0)$, які будуть простими числами. Тоді CM метод для дискримінанта $D = 3$ [19, 20] виробляє криві вигляду

$$E(\mathbb{F}_p) : y^2 = x^3 + b, b \neq 0$$

.

Тепер необхідно знайти b. Потрібно взяти найменше $b \neq 0$ таке, що $b + 1$ буде квадратичним лишком за модулем p і точка $G = (1, \sqrt{b+1} \bmod p)$ задовільняє $nG = O$ (порядок кривої — велике просте число, тому не існує точок вигляду $(0, y)$ із порядком 3). Цей метод спрощення підходу, описаного в [21], і швидко збігається до потрібного b.

Бінарна довжина m порядку еліптичної кривої легко встановлюється вибором x_0. Потрібно почати з найменшого $x \sim 2^{m/4}$ такого, що $n(x)$ буде мати бінарну довжину m, і збільшувати його поки $n(x)$ та $p(x)$ не стануть простими. $\square$

Отже, в результаті маємо такі параметризації:

$$\begin{aligned}
t &= 6x^2 + 1, \\
n &= 36x^4 + 36x^3 + 18x^2 + 6x + 1, \\
p &= 36x^4 + 36x^3 + 24x^2 + 6x + 1, \\
DV^2 &= 108x^4 + 144x^3 + 84x^2 + 24x + 3 = 3(6x^2 + 4x + 1)^2.
\end{aligned}$$

В реалізації алгоритму, для отримання x_0, було використано бінарний пошук на відрізку $[2^{\lfloor m/4 \rfloor - 2}, 2^{\lfloor m/4 \rfloor + 2}]$. Сам алгоритм, доведений у теоремі, виглядає так:

Algorithm 2: Побудова кривої простого порядку із $k = 12$

Input: Бінарна довжина m

Output: Параметри p, n, b, y, які утворюють криву $y^2 = x^3 + b$ із простим порядком n над полем $\mathbb{F}_p$ і породжуючою точкою $G = (1, y)$

```
Function GetSmallestX(m)
    L ← 2^⌊m/4⌋ − 2
    R ← 2^⌊m/4⌋ + 2
    while L < R do
        x ← ⌊(L + R)/2⌋
        c ← ⌈log_2(P(−x))⌉
        if c < m then
            L ← x + 1
        else
            R ← x
        end
    end
    return L
end
x ← GetSmallestX(m)
loop
    t ← 6x^2 + 1
    p ← P(−x)
    n ← p + 1 − t
    if p та n прості then
        break
    end
    p ← P(x)
    n ← p + 1 − t
    if p та n прості then
        break
    end
    x ← x + 1
end
b ← 0
repeat
    repeat
        b ← b + 1
    until b + 1 квадратичний лишок за модулем p
    Порахувати таке y, що y^2 = b + 1 mod p
    G ← (1, y) на кривій E : y^2 = x^3 + b
until nG = O
return p, n, b, y
```

В теоремі ми не розглядали $u(x) = 2x^2$, оскільки в такому випадку DV^2 розкладається як квадратний вільний добуток незвідних многочленів, що в загальному випадку призводить до дуже великого дискримінанту D.

Також щодо перевірки $nG = O$ в останньому циклі. CM конструкція гарантує, що порядок кривої, який задовільнє рівняння $3V^2 = 4p - t^2$, приймає одну з шости форм $\{p+1 \pm t, p+1 \pm (t \pm 2V)/2\}$[21]. Тобто не всі варіанти вибору b дадуть криву з потрібним порядком. Так як ймовірність того, що $b+1$ є квадратичним лишоком у рядку 32, буде приблизно $1/2$, а ймовірність правильного вибору b $1/6$, то очікується, що алгоритм буде перевіряти десь 12 можливих варіантів b до своєї зупинки.

В алгоритмі обирається найменше x, але це можно легко змінити для отримання іншого результату. Наприклад, у статті зроблена модифікація, яка максимізує p і n для заданої бінарної довжини та максимально спрощуючи інші параметри (b, G). Також x можно обирати випадковим чином на певному проміжку.

2 Основні результати

Наведені нижче криві згенеровані програмною реалізацією алгоритму. Вони задовільняють рівняння $E(\mathbb{F}_p) : y^2 = x^3 + b$ із простим порядком n, слідом відображення Фробеніуса t та породжуючим елементом $G = (1, y)$.

128 біт:

$$p = 170141201157549966723314726635601483473$$
$$n = 170141201157549966710270908127474967889$$
$$t = 13043818508126515585$$
$$b = 26$$
$$G(1, y) = (1, 23312569331926977698552035429200437876)$$

Час роботи get_pair: 0.0049 с

Час роботи get_curve: 0.2905 с

Загальний час роботи: 0.2955 с

160 біт:

$$p = 730750820779114522776002280017373265478724878629$$
$$n = 730750820779114522776001425177727027045004358733$$
$$t = 854839646238433720519897$$
$$b = 2$$
$$G(1, y) = (1, 338670257705860180385527562199106733311931085415)$$

Час роботи get_pair: 0.0628 с

Час роботи get_curve: 0.0588 с

Загальний час роботи: 0.1216 с

256 біт:

$$p = 5789604461865875109835728647999158233952285020785139786214682856582194276043 1$$
$$n = 57896044618658751098357286479991582339282234238683391992883452969489537512281$$
$$t = 240615969168005869263375596332405248151$$
$$b = 7$$
$$G(1, y) = (1,\ 8361881673513282932067712456830243220239610601706896947202393234200630622 18)$$

Час роботи get_pair: 0.6961 с

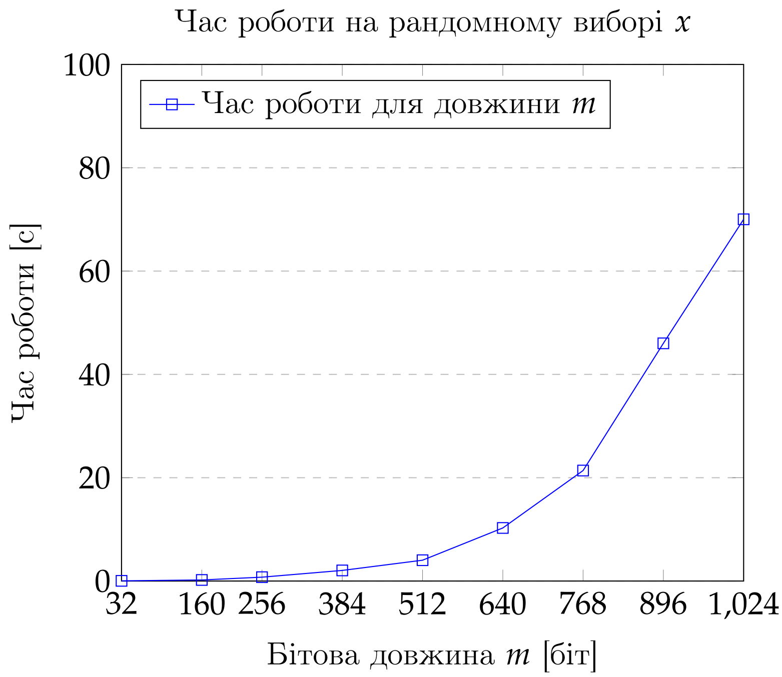
Час роботи get_curve: 0.3351 с

Загальний час роботи: 1.0312 с

Розглянемо час роботи алгоритму, коли ми обираємо найменше x.

На перший погляд цей стрибок на 896 біт виглядає дивно, але це через проблему розподілу p і n. Важко сказати як швидко ми знайдемо такі пари простих чисел. У випадку 896 біт нам не пощастило, тому p та n алгоритм шукав довше, ніж можна було очікувати.

Зробимо невелику модифікацію алгоритму. Тепер замість найменшого x будемо обирати рандомне значення з інтервалу [GetSmallestX(m), GetSmallestX(m+1) - ϵ], де ϵ деяка величина, щоб наші пари p і n при роботі алгоритму точно були бінарної довжини m. Для кожної запропонованої m згенеруємо криві на 20 випадкових значеннях x і виведемо середній час.



3 Висновки

В ході роботи було успішно реалізовано запропонований алгоритм. Теоретична частина була досить новою, але не занадто складною. Як і було сказано в статті[1], алгоритм генерування еліптичних кривих із степенем занурення $k = 12$ не є складним з точки зору програмної реалізації, але потребує певних теоретичних знань.

Щоб досягти сили симетричного ключа в 256 біт стандартному асиметричному алгоритму потрібен астрономічно великий ключ розміру 15 360 біт. Але для еліптичної кривої цей рівень сили досягається при 512 біт. Програмна реалізація генерування кривої працює досить швидко на бітовій довжині 256-512 біт, що достатньо для сучасних потреб.

Література

[1] Paulo S. L. M. Barreto and Michael Naehrig. Pairing-Friendly Elliptic Curves of Prime Order. *Selected Areas in Cryptography – SAC 2005*, 2006.

[2] D. Boneh and M. Franklin. Identity-based encryption from the Weil pairing. *SIAM Journal of Computing*, 2003.

[3] J. Silverman and J. Tate. Rational points on elliptic curves. *Springer-Verlag*, 2015.

[4] Christopher J. Swierczewski and William Stein. Connections Between the Riemann Hypothesis and theSato-Tate Conjecture. 2008.

[5] D. E. Knuth. Art of Computer Programming - Volume 2 (Seminumerical Algorithms). 1981.

[6] B. Lynn P.S.L.M. Barreto, H.Y. Kim and M. Scott. Efficient algorithms for pairing-based cryptosystems. *Advances in Cryptology - Crypto'2002*, 2002.

[7] Ben Lynn Paulo S. L. M. Barreto and Michael Scott. Constructing Elliptic Curves with Prescribed Embedding Degrees. *In Security in Communication Networks – SCN'2002*, 2002.

[8] M. Muller G. Fret and H. Ruck. The Tate Pairing and the Discrete Logarithm Applied to Elliptic Curve Cryptosystems. *IEEE Transactions on Information Theory*, 1999.

[9] K. Harrison S.D. Galbraith and D. Soldera. Implementing the Tate pairing. *Lecture Notes in Computer Science 2369*, 2002.

[10] A. Joux and K. Nguyen. Separating Decision Diffie-Hellman from Diffie-Hellman in Cryptographic Groups. *Journal of Cryptology*, 2003.

[11] G. Frey and H. Ruck. A Remark Concerning m-Divisibility and the Discrete Logarithm in the Divisor Class Group of Curves. *Mathematics of Computation*, 1994.

[12] J. Silverman. The arithmetic of elliptic curves. *Graduate Texts in Mathematics*, 1986.

[13] Andrew Lin. Complex Multiplication and Elliptic Curves.

[14] Igor Tolkov. Counting points on elliptic curves: Hasse's theorem and recent developments. 2009.

[15] Thomas R. Nicely. The Baillie-PSW primality test. 2012.

[16] Tristan Shin. Quadratic Residues. 2018.

[17] R. Lidl and H. Niederreiter. Introduction to finite fields and their applications. *Cambridge University Press*, 1986.

[18] J. McKee S. Galbraith and P. Valenca. Ordinary abelian varieties having small embedding degree. *Cryptology ePrint Archive*, 2004.

[19] G.J. Lay and H. G. Zimmer. Constructing elliptic curves with given group order over large finite fields. *Algorithmic Number Theory Symposium – ANTS-I*, 1994.

[20] F. Morain. Building cyclic elliptic curves modulo large primes. *Advances in Cryptology – Eurocrypt'1991*, 1991.

[21] USA IEEE Computer Society, New York. *IEEE Standard Specifications for PublicKey Cryptography – IEEE Std 1363-2000*, 2000.

A Програмний код

A.1 Допоміжний файл operations_over_EC.py

```
class Curve:
    """
    Клас еліптичної кривої, який містить її опис
    """
    def __init__(self, p, n, b, equation='y**2 - x**3 - self.b'):
        self.b = b
        self.n = n
        self.p = p
        self.equation = equation

    def eq(self, point):
        """
        Обчислює рівняння кривої для заданої точки.
        Використовується в перевірці чи належить точка кривій

        :param point: точка із координатами (x,y)
        :return: значення self.equation на цій точці
        """
        x, y = point
        return eval(self.equation)

    def __contains__(self, point):
        """
        Перевірка належності точки кривій.
        Якщо точка -- нейтральний елемент або рівняння на її координатах
        дорівнює нулю, то вона належить еліптичній кривій, інакше
        не належить.

        :param point: точка, яку ми збираємось перевірити
        :return: True якщо належить, інакше False
        """
        if isinstance(point, Point):
            return True if point.identity or \
                            self.eq((point.x, point.y)) % self.p == 0 \
                            else False
        else:
            return False

    def new_point(self, x, y):
        """
        Створення нової точки на кривій

        :param x: x координата точки
        :param y: y координата точки
        :return: Клас Point
        """
        return Point((x, y), self)


class Point:
    """
    Клас, що описує точки еліптичної кривої та дії над ними
```

```
    """
    def __init__(self, point, curve):
        self.curve = curve
        self.x, self.y = point
        if point == (0, 0):
            self.identity = True
        else:
            self.identity = False

        assert self in self.curve

    def __neg__(self):
        """
        Взяття оберненого елемента точки
        """
        assert self in self.curve

        if self.identity:
            return self

        self.y = -self.y % self.curve.p
        assert self in self.curve

        return self

    def __add__(self, other):
        """
        Обчислення суми двох точок
        """
        assert self in self.curve
        assert other in self.curve
        assert self.curve is other.curve

        if self.identity:
            return other.__copy__()
        if other.identity:
            return self.__copy__()

        if self.x == other.x and self.y != other.y:
            return Point((0, 0), self.curve)

        if self.x == other.x:
            m = (3 * self.x ** 2) * inverse_mod(2 * self.y, self.curve.p)
        else:
            m = (self.y - other.y) * inverse_mod(self.x - other.x, self.curve.p)

        x = m ** 2 - self.x - other.x
        y = self.y + m * (x - self.x)

        new_point = Point((x % self.curve.p, -y % self.curve.p), self.curve)
        assert new_point in self.curve
        return new_point

    def __str__(self):
        return f'{self.x, self.y}'

    def __copy__(self):
```

```
        return Point((self.x, self.y), self.curve)

    def __mul__(self, other: int):
        """
        Множення точки на ціле число
        """
        assert isinstance(other, int)
        assert self in self.curve

        if other < 0:
            return (-self).__mul__(-other)

        result = Point((0, 0), self.curve)
        addend = self

        while other:
            if other & 1:
                result = result + addend

            addend = addend + addend
            other >>= 1

        assert result in self.curve
        return result

    def __rmul__(self, other):
        return self.__mul__(other)


def egcd(a, b):
    """
    Алгоритм Евкліда. Використовується в inverse_mod
    """
    x, y, u, v = 0, 1, 1, 0
    while a != 0:
        q, r = b // a, b % a
        m, n = x - u * q, y - v * q
        b, a, x, y, u, v = a, r, u, v, m, n
        gcd = b
    return gcd, x, y


def inverse_mod(k, p):
    """
    Взяття оберненого елемента за модулем p
    """
    if k == 0: raise ZeroDivisionError('division by zero')
    if k < 0 : return p - inverse_mod(-k, p)
    gcd, x, y = egcd(k, p)

    assert gcd == 1
    assert (k * x) % p == 1

    return x % p


def get_test_curve():
    p = 1461501624496790265145448589920785493717258890819
```

```
    n = 1461501624496790265145447380994971188499300027613
    b = 3
    return Curve(p, n, b)


if __name__ == '__main__':
    ec = get_test_curve()
    G = ec.new_point(1, 2)
    print(f'Point G: {G}')
    print(f'G + G: {G + G}')
    print(f'nG: {ec.n * G}')
```

A.2 Головний файл main.py

```
from time import time
from math import log2, ceil
from sympy import isprime, is_quad_residue, sqrt_mod
from operations_over_EC import Curve, Point


def P(x):
    return 36 * x ** 4 + 36 * x ** 3 + 24 * x ** 2 + 6 * x + 1


def get_smallest_x(m):
    """
    :param m: бажана бінарна довжина p і n
    :return: найменше x
    """
    left = 2 ** (m // 4 - 2)
    right = 2 ** (m // 4 + 2)

    while left < right:
        x = (left + right) // 2
        c = ceil(log2(P(-x)))
        if c < m:
            left = x + 1
        else:
            right = x

    return left


def get_pair(m):
    """
    Пошук пари простих чисел p і n

    :param m: бінарна довжина m
    :return: p і n
    """
    x = get_smallest_x(m)

    while True:
        t = 6 * x ** 2 + 1
        p = P(-x)
        n = p + 1 - t
        if isprime(p) and isprime(n):
            return p, n, t
```

```
        p = P(x)
        n = p + 1 - t
        if isprime(p) and isprime(n):
            return p, n, t
        x += 1


def is_identity(point):
    return point.identity


def get_curve(n, p):
    """
    Знаходження кривої за параметрами n і p

    :return: параметр b у рівнянні кривої та породжуюча точка
    """
    b = 1
    while True:
        b += 1
        while not is_quad_residue(b + 1, p):
            b += 1
        y = sqrt_mod(b + 1, p)
        ec = Curve(p, n, b)
        G = ec.new_point(1, y)
        if G in ec and is_identity(n * G):
            break

    return b, Point((1, y), ec)


if __name__ == '__main__':
    m = int(input('m: '))
    start = time()
    p, n, t = get_pair(m)
    end1 = time()
    print(f'Час роботи get_pair: {end1 - start} c')
    print(f'p: {p} {len(bin(p)[2:])} біт')
    print(f'n: {n} {len(bin(n)[2:])} біт')
    print(f't: {t} {len(bin(t)[2:])} біт')
    start1 = time()
    b, point = get_curve(n, p)
    end = time()
    print(f'Час роботи get_curve: {end - start1} c')
    print(f'b: {b}')
    print(f'G(1,y): {point}')
    print(f'Загальний час роботи: {end - start} c')
    print()
```